Université de Montpellier

Projet-HLIN601

Generative Adversarial Network

CHEVALIER Baptiste
LEHOUAOUI Sara
LEMONNIER Adam
ZEDDAM Lylia
Encadrant :
M.Pascal Poncelet

GitLab : https ://git-lab.etu.umontpellier.fr /e20190006547/gan.git

Avril 2021

Nous souhaitons exprimer notre reconnaissance envers Mr Pascal Poncelet, notre encadrant sur ce TER, pour sa patience et sa disponibilité. Également pour ses conseils et ses directives, qui nous ont permis de réaliser ce travail.

Table des matières

1 Présentation du sujet et motivations **4**

1.1 Introduction 4

1.2 Organisation 4

1.2.1 Démarche 4

1.2.2 Outils 5

2 Introduction des concepts **6**

2.1 Rappels sur la classification 6

2.2 Rappels sur les Réseaux Neuronaux 7

2.2.1 Concept Général 7

2.2.2 Réseaux Convolutifs (CNN) 8

2.3 Rappels sur les réseaux utiles à la génération de texte 9

2.3.1 Réseaux de neurones récurrent (RNN) 9

2.3.2 Long Short Term Memory (LSTM) 10

2.3.3 Word Embedding 10

2.4 Présentation des Réseaux Génératifs Antagonistes (GAN) 11

2.5 Lexique 12

3 Présentation des travaux : Génération de données avec différentes architectures de GANs **13**

3.1 Objectif 13

3.2 Architectures 13

3.2.1 GAN Standard 13

3.2.2 GAN Conditionnel 14

3.3 Expérimentations et résultats 15

3.3.1 Jeu de donnée 15

3.3.2 Résultats et Comparaisons 15

3.3.3 Biais et problèmes 18

4 Présentation des travaux : Génération de texte avec différents modèles générateurs **19**

4.1 Objectif . . . 19

4.2 Architectures . . . 19

4.2.1 LSTM et Embeddings . . . 19

4.2.2 GAN . . . 20

4.3 Expérimentations et Résultats . . . 20

4.3.1 Jeu de données . . . 20

4.3.2 Explication des expérimentations . . . 21

4.3.3 Qualité des phrases générées . . . 21

4.3.4 Accuracy . . . 22

4.3.5 Entraînement d'un classifieur avec des phrases générées . . . 26

4.3.6 Biais . . . 26

5 Bibliographie **27**

Chapitre 1

Présentation du sujet et motivations

1.1 Introduction

Dans le cadre de ce TER, nous nous sommes intéressés aux modèles génératifs et plus spécifiquement aux GANs : Réseaux Génératifs Antagonistes. L'objectif était d'étudier et de comparer différentes architectures de GANs et de modèles de générateurs.

Dans un monde ou les modèles génératifs sont en pleine croissance et sont utilisés dans de plus en plus de domaines, il est intéressant de se pencher sur la récente technologie que sont les GANs. Cette technologie à déjà fait ses preuves notamment dans un cadre de génération de fausses images avec des réseaux profonds de convolution génératifs Antagonistes (DCGAN).

Le travail effectué dans ce projet se résume en plusieurs étapes :

- Compréhension général des réseaux de neurones
- Mise en place d'architectures sur des jeux de données
- Comparaison des différentes architectures et test de la qualité des données générés.
- Même chose avec des données textuelles nécessitant des réseaux plus complexes.

1.2 Organisation

1.2.1 Démarche

Pour notre première approche des GANs nous avons utilisé MNIST, un jeu de données d'images de chiffres. Nous nous sommes appuyés sur un article accessible ici [Gün], cette implémentation nous a permis d'y voir plus clair sur le fonctionnement de ce type de réseau. Nous nous sommes rapidement approprié le code et avons pu modifier à notre convenance le générateur en limitant la génération à un chiffre en particulier par exemple.

La seconde étape a été l'utilisation du jeu de donnée Iris (accessible sur le site UCI [Fis]). Cet autre approche nous a permis de mieux comprendre l'utilité des GANs dans le domaine de l'intelligence artificielle, comme outil de traitement de données. Nous avons commencé par nous approprier le jeu de données composé de propriétés précises sur 3 types d'Iris. Une fois bien compris il a fallu adapter le générateur du premier GAN pour qu'il génère des Iris réalistes.

Nous nous sommes ensuite dirigés vers la génération de texte afin d'appréhender différemment les GANs. Pour commencer, nous avons appris à générer du texte en utilisant un autre réseau neuronal, un LSTM. Pour entraîner ce dernier nous nous sommes servi du texte original de Lewis Carroll, "Alice in Wonderland". Après un pré-traitement du texte (remplacement des majuscules par des minuscules, retrait de la ponctuation,...) et quelques étapes d'entraînements, notre réseau pouvait générer la suite d'une phrase caractères par caractères. Cette première étape nous a beaucoup appris sur la génération de texte. Le second objectif était de générer des avis négatifs ou positifs. Sur le même principe que les expérimentations précédentes nous avons utilisé des données issues de plate-formes différentes (Twitter, avis sur Amazon, ...). Dans cette configuration, le générateur a dut être entraîné sur de nombreuses données avant de renvoyer des avis "réalistes".

1.2.2 Outils

Afin d'implémenter nos modèles nous avons choisit d'utiliser python qui est le langage le plus utilisé dans le domaine de l'intelligence artificielle et nous permets ainsi d'avoir accès à de nombreux outils et bibliothèques. Parmi celles ci nous pouvons citer Keras qui se base sur tensorflow et est une bibliothèque très complète pour écrire des réseaux neuronaux. D'autre bibliothèque auraient pu être utilisés comme PyTorch mais nous avons préféré utiliser principalement Keras de par sa simplicité de compréhension et de mise en oeuvre. Pour réaliser ce projet nous nous sommes principalement servis de Google Colab, un outils mis en place par Google et relié au Drive, permettant la lecture et rédaction de Notebooks en python. Les Notebooks ont été essentiels dans nos expérimentations et nous ont permis d'organiser convenablement le code de chaque partie, et de facilement compiler et visualiser nos résultats.
Dans notre code, nous avons utilisé de nombreuses bibliothèques en python permettant la mise en oeuvre de différents points. Pour la manipulation des données nous avons utilisé les dataframes de la bibliothèque Panda, ainsi que nltk (Natural Langage Toolkit) qui nous a été utile pour les différents pré-traitement appliqué aux données. Sklearn a également été utilisé pour l'implémentation de différents classifieurs. Afin de réaliser des graphiques et d'illustrer nos résultats nous nous sommes servis de Matplotlib.
Pour partager nos fichiers et centraliser nos travaux nous avons principalement utilisé Gitlab, mais également Google Drive.

Chapitre 2

Introduction des concepts

2.1 Rappels sur la classification

Les GANs dont traitent ce projet se placent dans un cadre de classification de données, c'est pourquoi il est important de rappeler le contexte.

Contexte

On dispose d'un jeu de données classifiées, c'est à dire réparties dans au moins deux classes. Par exemple, des images de chats et de chiens. On souhaite à partir des données dont la classe est connue, créer un modèle qui pourra déterminer la classe de futures données. Pour cela il va falloir "entraîner" notre modèle à partir des données réelles.

Types de classifieurs

Un exemple simple de classification est la régression linéaire, on peut également citer une régression linéaire multiple ou polynomiale. Cependant dans certains cas, il sera préférable d'utiliser des classifieurs moins triviaux. Voici une liste non exhaustive de quelques classifieurs.

- Régression Logistique
- Algorithme des K plus proches voisins (KNN)
- Arbre de décision
- Classification à vecteurs de support (SVC)

Évaluation d'un modèle

Nous disposons de plusieurs indices afin d'évaluer la qualité de notre modèle.
Le plus simple que nous utiliserons beaucoup dans ce projet est l'*accuracy*. Il s'agit tout simplement de calculer le ratio de validité de notre modèle en comparant les résultats obtenus sur un jeu de test. On peut citer également la matrice de confusion, le *recall* ou encore le *f1-score* qui sont d'autres indices permettant de déterminer la validité d'un modèle.

2.2 Rappels sur les Réseaux Neuronaux

2.2.1 Concept Général

Un réseau de neurones artificiels est un modèle s'inspirant des neurones biologiques dans un cadre de classification de données.

Principe

Les réseaux de neurones artificiels peuvent être décrits comme des systèmes composés d'au moins deux couches de neurones - une couche d'entrée et une couche de sortie - et comprenant généralement des couches intermédiaires "cachées". Plus le problème à résoudre est complexe, plus le réseau de neurones artificiels doit comporter de couches, lorsque le nombre de couches devient important on parle d'apprentissage profond *(Deep Learning)*.

Au sein d'un réseau de neurones artificiels, le traitement de l'information suit toujours la même séquence :
Les informations sont transmises aux neurones de la couche d'entrée.
Par la suite les neurones de chaque couche sont reliés exclusivement aux neurones de la couche qui succède, de plus on associe un poids à chaque liaison qui symbolisent la "force" de cette dernière.
On va multiplier la valeur des neurones de chaque couches par le poids de la liaison et sommer les résultats. Les sommes passent ensuite par une fonction d'activation qui est non-linéaire afin de normaliser les valeurs, les résultats seront les valeurs des neurones cachées pour la couche suivante. On répète l'opération jusqu'à la couche de sortie, cette étape est appelée propagation avant *(Forward Propagation)*
Note : Il existe de nombreuses fonctions d'activations à utiliser en fonction du contexte *(sigmoid, tanh, ReLU, SoftMax...)*

Dans l'étape suivante, on calcule une fonction de coût (ou de perte : coût cumulé), c'est à dire une fonction qui exprime l'écart entre le résultat obtenu et le résultats attendu. Il en existe plusieurs comme l'erreur quadratique ou l'entropie croisée.
Après cela on utilise l'algorithme de descente de gradient afin de déterminer les nouveaux poids, qui vont minimiser la fonction de coût, pour chaque connexions. Pour plus d'éléments relatifs à la descente de gradient se référer à [Wik]). Si l'algorithme du gradient nous donne la direction vers laquelle orienter nos poids, il ne nous oriente pas sur la façon de le faire. C'est pourquoi on introduit également le *learning rate* qui est le "pas" de variation des poids. Cette étape est appelée Rétro-propagation *Backward Propagation* (figure 2.1).

On répète le schéma propagation avant (*forward-propagation*) - rétro-propagation (*back-propagation*) pour un certains nombre d'*epoch* déterminé.

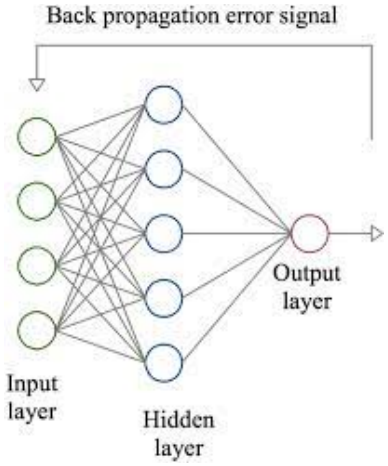

FIGURE 2.1 – Déroulement de la propagation avant et rétro-propagation

2.2.2 Réseaux Convolutifs (CNN)

Les réseaux de convolution ou CNN (*Convolutional Neural Network*) sont utilisés spécifiquement pour le traitement des images dans un but de classification. En effet, les images étant des objets de très grande dimension, elles nécessitent un pré-traitement avant d'être traitées par un réseau plus classique.

Les CNNs sont composés de différentes couches : (voir figure 2.2)
Premièrement les couches dites de *convolutions*. Ces couches agissent en appliquant des filtres à l'image afin d'en extraire les composantes principales. Ce pré-traitement va être très utile pour notre classification mais a pour défaut d'élargir encore les dimensions de notre entrée.
C'est pour cela que dans un second temps les couches de *pooling* ont pour rôle de "résumer" l'image en réduisant la dimensionnalité. Il existe différentes variantes, la plus utilisée étant le *max-pooling*.
Suite à cela une couche appelée *flatten* vient "aplanir" la matrice en la transformant en vecteur colonne.
Suivent ensuite de multiples couches standards dites "dense" à activation ReLU.
Enfin la couche de sortie utilise une activation sigmoïde si il y a deux classes ou softmax si il y en a plus.

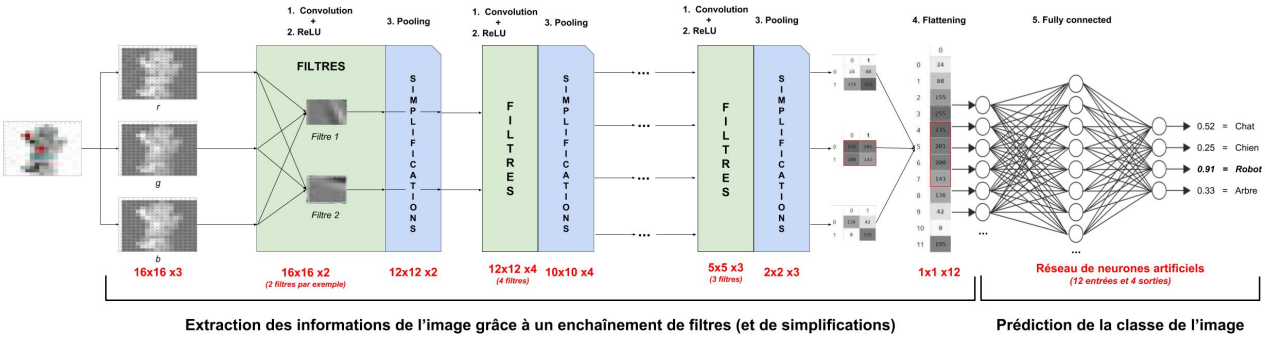

FIGURE 2.2 – Fonctionnement du CNN

2.3 Rappels sur les réseaux utiles à la génération de texte

2.3.1 Réseaux de neurones récurrent (RNN)

Les réseaux de neurones récurrent sont une approche des réseaux neuronaux où chaque prédiction se base sur les prédictions précédentes. Ils sont alors très utiles pour la génération de texte. Étant donné que le but de la génération de texte est de créer une certaine sémantique il faut que chaque nouveau caractère ou mot soit prédit en fonction de ceux qui le précèdent. La figure 2.3.1 illustre un exemple de RNN.

Pour plus d'informations sur les réseaux récurrent se référer à ce guide illustré [Phib].

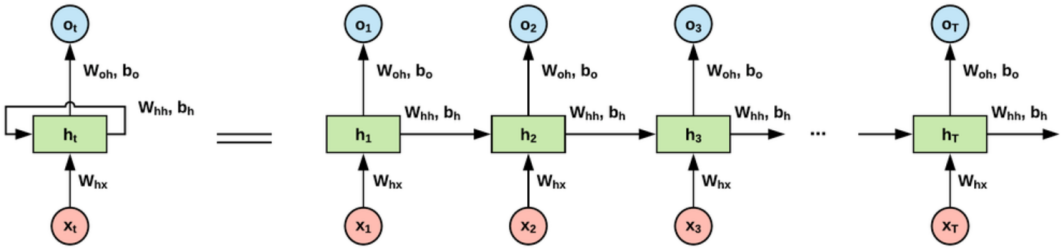

FIGURE 2.3 – Un exemple de réseaux de neurones récurrent

Cependant les RNNs ont un problème. Dans le cas du traitement de texte, l'importance des anciennes entrées décroît exponentiellement or, les mots/caractère du début d'une phrase ne sont pas forcement moins important que ceux de la fin.

2.3.2 Long Short Term Memory (LSTM)

Les LSTMs sont une architecture de réseau récurrent utilisée dans le domaine de l'apprentissage en profondeur notamment pour le traitement de texte. Contrairement aux réseaux de neurones récurrent standard, les LSTMs ont des connexions de rétroaction. Les prédictions sont faites non-seulement en se basant sur l'entrée et la sortie précédentes comme dans le cas des RNNs mais aussi sur une mémoire. Pour plus de détails sur leurs fonctionnement se reporter à cet article [Phia] illustrant le fonctionnement pas à pas de ce type de réseau. La figure 2.4 schématise la cellule d'un LSTM.

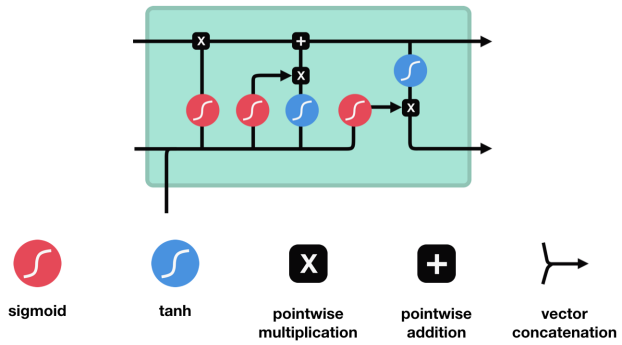

FIGURE 2.4 – Schéma Cellule LSTM

2.3.3 Word Embedding

Le *word embedding* que l'on pourrait traduire par "vectorisation de mots" est une méthode utilisée en apprentissage dans le traitement des langues. Il s'agit de représenter chaque mot ou groupe de mots par un vecteur de nombre réel dans un espace de grande dimension, introduisant ainsi une notion de sémantique. On peut citer un exemple connu : KING - MAN + WOMAN = QUEEN. On s'aperçoit que les lois de composition interne dans cette espace se rapprochent fortement de l'idée intuitive de la sémantique (voir schéma 2.5). Nous avons utilisé le *Word Embedding* afin d'améliorer nos résultats dans le cadre de la génération de texte.

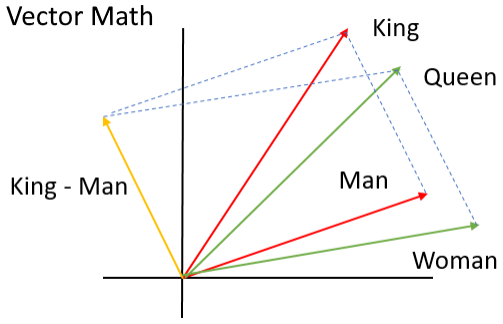

FIGURE 2.5 – Word Embedding Exemple

2.4 Présentation des Réseaux Génératifs Antagonistes (GAN)

Un GAN (*Generative Adversarial Network*) ou réseau antagoniste génératif en français, est une technique qui repose sur la mise en compétition de deux réseaux.

Ces deux réseaux sont appelés "générateur" et "discriminateur". Le générateur a pour rôle de créer des données d'un certains type tandis que le discriminateur doit pouvoir différencier les données fiable des fausse de la façon la plus rigoureuse possible.
Pendant le processus d'entraînement, ces deux entités sont mises en compétition ce qui leur permet d'améliorer leurs comportements respectifs. En effet, les deux réseaux ont impact l'un sur l'autre, la rétro-propagation se fait sur l'ensemble des deux réseaux du discriminateur vers le générateur.
L'objectif du générateur est de produire des données sans que l'on puisse déterminer leur véracité, tandis que le discriminateur doit identifier les fausses données. Ainsi, le générateur produit des données de meilleure qualité tandis que le discriminateur détecte de mieux en mieux les fausses.

Dans un premier temps, il convient de déterminer ce que l'on souhaite que le GAN produise (la sortie). On va ensuite donner au réseau génératif un bruit de grande dimension en entrée afin qu'il essaie de produire des objets.
Ces objets générés sont ensuite transmis au discriminateur, aux côtés des véritables données du jeu. Le discriminateur détermine l'authenticité de la sortie. En établissant que les objets générés sont faux alors que ceux du jeu de données sont réels, on peut alors calculer la fonction de perte et appliquer la rétro-propagation.
La figure 2.6 schématise le fonctionnement d'un GAN.

Une fois la méthode en place, il est possible de réaliser différentes variantes. Ainsi il sera possible d'utiliser une réseau convolutif en discriminateur si on souhaite générer des images, ou encore un réseau récurrent en générateur pour générer du texte.

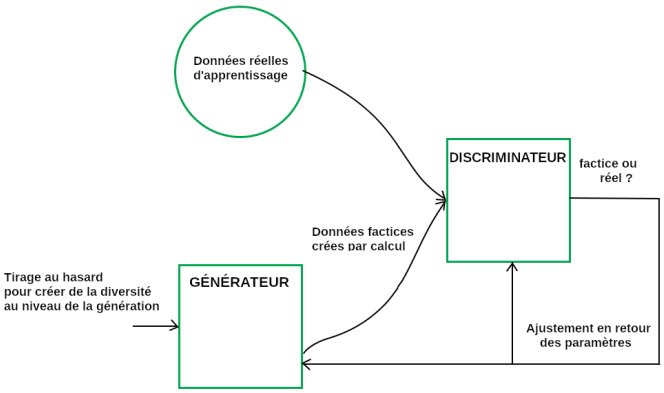

FIGURE 2.6 – Schéma fonctionnement des GANs

2.5 Lexique

Batch : Lot d'un jeu de donnée utilisé pour l'entraînement à la place du training set complet.

Cross-Validation : méthode d'estimation de fiabilité d'un modèle. Elle consiste à diviser le jeu de donnée en au moins 2 parties, un jeu d'entraînement et un jeu de test. Également le k-folds divise le jeu en k blocs ou l'un est utilisé pour le test et les autres pour l'apprentissage. Il existe aussi la validation 1 contre tous qui est un cas particulier du k-folds avec k égal au nombre de données.

Dropout : technique utilisée pour empêcher la sur-interprétation.Elle consiste à désactiver aléatoirement l'activation de certains neurones des couches intermédiaires.

Les fonctions d'activations

LeakyReLU : Leaky version de la fonction ReLU (unité linéaire rectifiée). Contrairement à la fonction ReLU elle n'empêche pas la rétro-propagation pour les valeurs négatives et est donc souvent utilisé pour les couches intermédiaires des réseaux de neurones.

$$f(x) = \max(\epsilon x, x)$$

Sigmoid : Elle représente la fonction de répartition de la loi logistique. On utilise cette fonction pour la couche de sortie des réseaux dans un cadre de classification binaire.

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}}$$

SoftMax : Aussi appelé fonction exponentiel normalisé, c'est une généralisation de la fonction logistique. On l'utilise pour la couche de sortie des réseaux dans un cadre de classification multi-classes.

Chapitre 3

Présentation des travaux : Génération de données avec différentes architectures de GANs

3.1 Objectif

Dans cette partie, nous étudierons la génération de données afin de compléter un jeu de donnée. Le jeu de donnée pourra ensuite être utilisé pour entraîner un classifieur par exemple. Cela peut s'avérer particulièrement utile dans le cas ou les classes ne sont pas équilibrés, pour compléter une ou plusieurs classes avec des données générés réalistes. Les données se doivent donc d'être le plus proche possible de celle du jeu d'origine. Aussi on pourra utiliser un classifieur afin d'évaluer la qualité des données que nous avons généré.

3.2 Architectures

3.2.1 GAN Standard

Le premier modèle est un GAN standard dont le fonctionnement a été rappelé dans la partie 2.4 . Le modèle est constitué d'un seul générateur et discriminateur. Pour plus d'informations se référer à l'article [Goo]

On nomme ρ_{data} la distribution des données réelles et ρ_z la distribution du bruit pris en entrée par le générateur. On veut entraîner notre modèle afin d'obtenir la distribution ρ_g à partir de la distribution ρ_z. En assimilant nos modèles à des fonctions mathématiques, soit D la fonction qui associe à toute entrée du discriminateur sa sortie et G de même pour le générateur. La sortie de D est un scalaire compris entre 0 et 1, une probabilité que la donnée en entrée soit réelle ou fausse. L'objectif est alors :

$$\min_G \max_D \mathbb{E}_{x\sim\rho_{data}(x)}[\log D(x)] \ + \mathbb{E}_{z\sim\rho_z(z)}[\log(1 - D(G(z)))]$$

On cherche à modifier les paramètres G et D pour minimiser l'ensemble afin de rapprocher les deux distributions.

Générateur
- Couche entrée
- 2 Couches intermédiaires avec activation *LeakyReLU + BatchNormalisation*
- Couche de sortie avec activation *tanh*

Discriminateur
- Couche entrée avec dropout
- 3 Couches intermédiaires avec activation *tanh + BatchNormalisation*
- Couche de sortie avec activation *sigmoid*

3.2.2 GAN Conditionnel

La deuxième architecture est un GAN Conditionnel (CGAN). Ce réseaux contient également 1 unique générateur et discriminateur comme un GAN classique. Cependant, les entrées de chaque sous réseaux sont doubles. Le générateur et le discriminateur prennent chacun, en plus du bruit et des données, le *label* de la donnée à générer. Le schéma d'un tel réseau est montré sur l'image 3.2.2 Pour plus d'informations se référer à l'article [Mir]

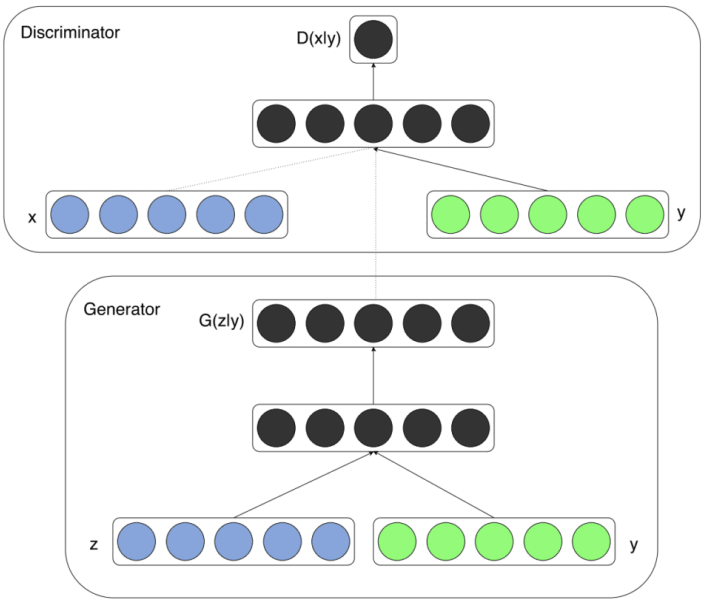

FIGURE 3.1 – Schéma CGAN

Les fonctions G et D sont maintenant conditionnées par les *labels* **y**. Le nouvel objectif est le suivant :

$$\min_G \max_D \mathbb{E}_{x\sim\rho_{data}(x)}[\log D(x|\mathbf{y})] \ + \mathbb{E}_{z\sim\rho_z(z)}[\log(1 - D(G(z|\mathbf{y})))]$$

Générateur
- Première couche d'entrée pour le bruit
- Deuxième couche d'entrée pour les *labels*
- Couche de concaténation
- 3 Couches intermédiaires
- Couche de sortie

Discriminateur
- Couche entrée (données + *labels*)
- 3 Couches intermédiaires
- Couche de sortie

3.3 Expérimentations et résultats

3.3.1 Jeu de donnée

Pour nos expérimentations, nous nous sommes servi du jeu de donnée Iris. Il est constitué de 3 classes qui sont 3 espèces d'Iris : Sétosa, Versicolor et Verginica, pour lesquels nous avons relevé 4 caractéristiques : la longueur et la largeur des sépales (en cm) ainsi que la longueurs et la largeurs des pétales (en cm). Le jeu de données contient 150 exemples et est réparti en 3 ensembles de 50 exemples par classe.

3.3.2 Résultats et Comparaisons

Modèle 1

Le GAN standard nous permet uniquement de générer des données appartenant à une même classe. Ici nous avons par exemple choisi de créer des Iris "Virginica". Sur la figure 3.2 sont présentées dans 3 couleurs différentes chacune des classes d'Iris du jeu de données d'origine. En plus de cela, les Iris générées apparaissent en bleu sur le graphique (label G).
On peut remarquer qu'a l'exception de 2 exemplaire, les Iris générées se confondent parfaitement aux Iris de la classe Virginica du jeu de données.

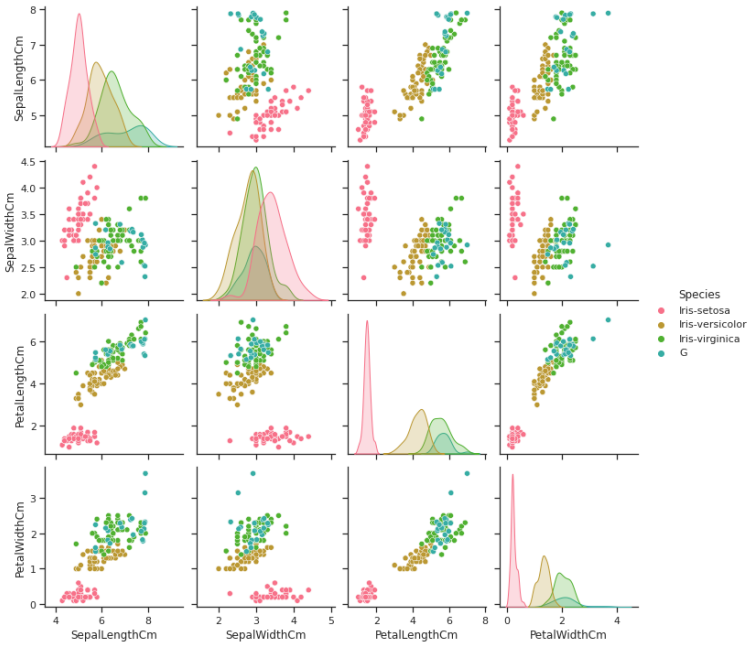

FIGURE 3.2 – Vecteurs réels et générés

Modèle 2

Le modèle conditionnel nous permet de choisir la composition du set généré afin d'avoir un set similaire à l'original et de les comparer. Nous avons donc choisit de générer 50 exemplaires de chaque classe.
Sur les figures 3.3 et 3.4 sont présentées dans 3 couleurs différentes chacune des classes d'Iris. Chaque graphique de la matrice (excepté les diagonales) présente les données selon deux critères (*features*). Les diagonales , elles, représentent la distribution marginale des vecteurs en fonction de chaque caractéristiques.
La figure 3.3 représente les données générés par le modèle CGAN tandis que la figure 3.4 montre la répartition des données du jeu de données.
Nous pouvons voir que les données générées collent plutôt bien avec celles du jeu de données, autrement dit les données générées sont réalistes et pourraient peut être être utilisées pour compléter le set. On remarque tout de même quelques différences sur la répartition. Celle ci est moins étendue pour les données générées, cela découlant sûrement du fait que le modèle cherche à généraliser la classe d'Iris et évite donc les exceptions.

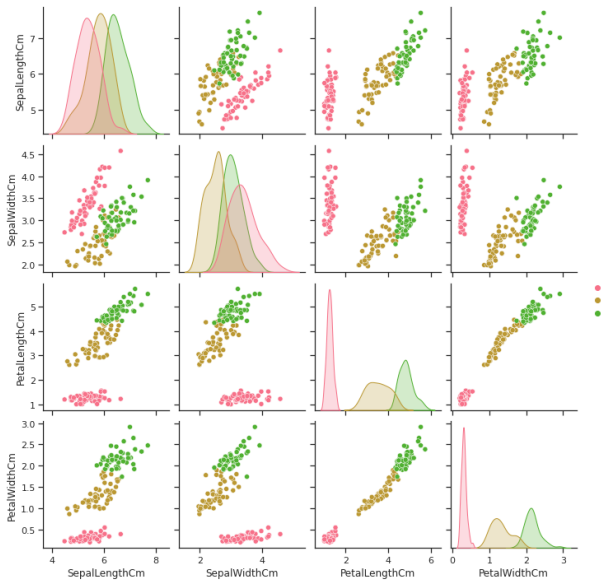

FIGURE 3.3 – CGAN

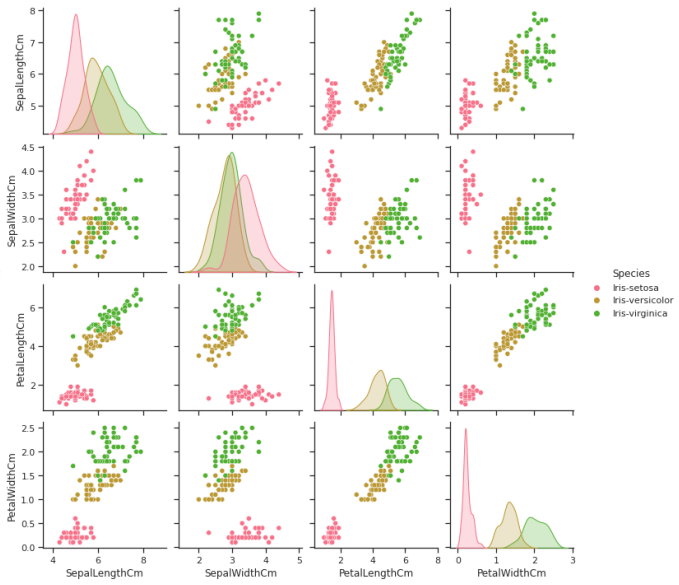

FIGURE 3.4 – Réel

Afin de savoir avec plus de certitude si les données peuvent être utilisées pour compléter un set, nous pouvons utiliser un classifieur. Comme on peut le voir sur les figures 3.5 et 3.6 les résultats sont très bons. L'accuracy est élevée et il y a peu d'erreur dans les matrices de confusion.

```
Accuracy : 0.950
Classification Report
              precision    recall  f1-score   support

         0.0    0.00000   0.00000   0.00000         0
         1.0    1.00000   0.95000   0.97436        20

    accuracy                        0.95000        20
   macro avg    0.50000   0.47500   0.48718        20
weighted avg    1.00000   0.95000   0.97436        20

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification
  _warn_prf(average, modifier, msg_start, len(result))
```

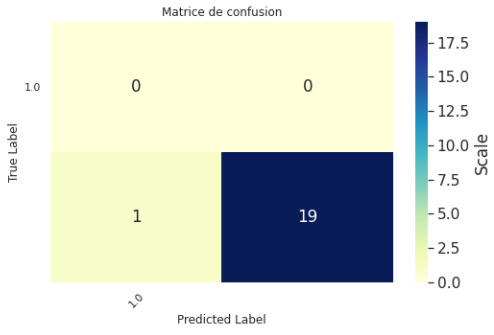

FIGURE 3.5 – Classification GAN

```
Accuracy : 0.960
Classification Report
              precision    recall  f1-score   support

         0.0    1.00000   1.00000   1.00000        50
         1.0    0.92308   0.96000   0.94118        50
         2.0    0.95833   0.92000   0.93878        50

    accuracy                        0.96000       150
   macro avg    0.96047   0.96000   0.95998       150
weighted avg    0.96047   0.96000   0.95998       150
```

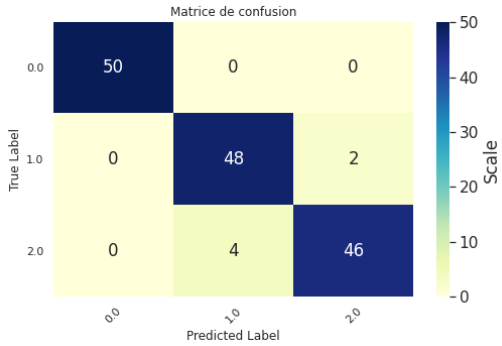

FIGURE 3.6 – Classification CGAN

Comparaison des modèles

Dans les deux cas, les données générées sont très ressemblantes aux données du jeu d'origine. Les deux modèles se valent donc si l'objectif est de compléter une classe avec de fausses données. Cependant, le second modèle est plus pratique puisqu'il permet de générer différentes classes simultanément sans en mélanger les caractéristiques ce qui n'est pas possible avec un GAN standard.

3.3.3 Biais et problèmes

Cette partie sert à relever quelques biais qui ont pu être présent dans les expérimentations. Par exemple, le classifieur utilisé pour valider nos données générées a été entraîné sur le même jeu de données que le modèle générateur. Par conséquent, les données seront mieux classées que si elles avaient été différentes. En effet, la petite taille du jeu de données (seulement 150 vecteurs) ne permet pas de le séparer pour utiliser des techniques de cross-validation.

Chapitre 4

Présentation des travaux : Génération de texte avec différents modèles générateurs

4.1 Objectif

Dans cette partie, nous allons étudier la génération de texte par différentes architectures de réseaux de neurones. Les motivations sont multiples comme la génération de faux avis par exemple. L'objectif est double :

- Générer du texte de bonne qualité qui soit compréhensible et éligible pour un humain.
- Générer du texte qui exprime une opinion (positive ou négative).

Pour le premier cas, la meilleur approche reste encore de faire évaluer le texte par un humain. En ce qui concerne l'opinion, on pourra comme dans la partie précédente utiliser un classifieur de texte afin de déterminer l'opinion des textes que nous avons généré. Nous verrons par la suite que même si les phrases ne sont pas toujours bien construites grammaticalement parlant, elle pourront être classées par le classifieur et donc servir à étendre un jeu de donnée ou même à entraîner un autre classifieur.

4.2 Architectures

4.2.1 LSTM et Embeddings

Notre premier modèle est un réseau génératif de type LSTM. En plus de cela nous utilisons les *embeddings glove* de dimensions 100. Ces *embeddings* ont été entraînés au préalable sur une quantité importante de données.

L'architecture du modèle LSTM se compose de plusieurs couches :
- Couche d'entrée (*Input Layer*)
- Couche d'*embeddings*
- Couche LSTM
- Couche d'abandon (*Dropout Layer*)
- Couche de sortie (*Outpout Layer*)

4.2.2 GAN

Récemment, les GANs ont montré des résultats prometteurs dans la génération de texte. Pour notre second modèle, on utilisera donc le sentiGAN (schéma figure 4.1). Ce modèle est constitué de plusieurs générateurs et un unique discriminateur multi-classes. C'est à dire que si le jeu de données comporte n-classes la sortie du discriminateur sera de taille n+1, pour les n classes et la dernière pour les fausses données. Ainsi plusieurs générateurs sont formés simultanément, le but est de générer des textes d'étiquettes de sentiments différents sans supervision.
Pour plus d'informations se référer à l'article [Ke]

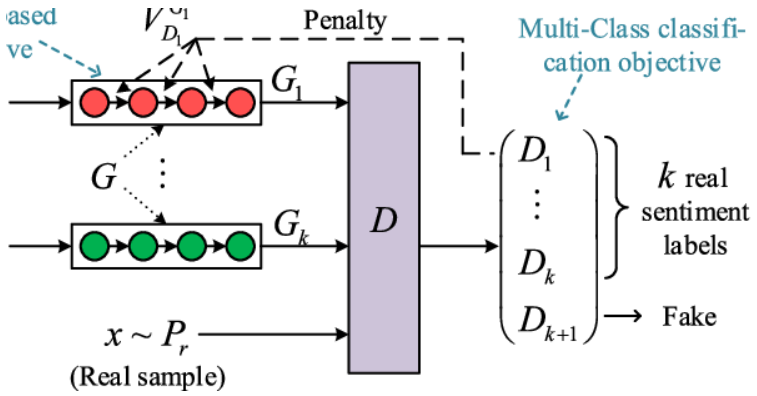

FIGURE 4.1 – Architecture SentiGAN

4.3 Expérimentations et Résultats

4.3.1 Jeu de données

Pour cette partie nous avons utilisé 4 jeux de données différents.

ReviewsLabelled est construit à partir d'avis provenant d'Amazon, Imdb et Yeld. Pour chacun des sites, nous disposons de 500 avis positifs et 500 avis négatifs. Soit un total de 3000 avis répartis en deux classes : positifs et négatifs.

Movie data contient des critiques de films, ce jeu est constitué de 50 000 avis. Les avis sont répartis dans deux classes équilibrées : positifs et négatifs, avec chacune 25 000 avis.

Sentiment140 dataset est constitué de 1,6 millions de tweets classés selon deux classes positifs ou négatifs de façon équilibrée.

Amazon app book est constitué à partir de critiques de livres récoltés par Amazon.

4.3.2 Explication des expérimentations

Nous avons réalisé deux expérimentations :
La première consiste à générer des avis à partir d'un jeu de données et de l'un des modèles, puis d'utiliser un classifieur pour tester l'efficacité de notre modèle générateur. A savoir, si le modèle a pour but de générer des avis positifs, il faut que le classifieur reconnaisse ces données générées comme effectivement positives.
La seconde consiste à ré-entraîner un classifieur en remplaçant une partie des données réelles par des données générées par l'un des modèles et de regarder l'évolution de l'**accuracy** en testant sur des données réelles. L'objectif est alors de savoir si il est possible d'utiliser nos données générés pour étendre notre set et entraîner un modèle qui reconnaîtra par la suite des données réelles.

4.3.3 Qualité des phrases générées

Afin de déterminer la qualité des phrases générées, il existe différentes métriques. Le calcul de perplexité qui exprime à quel point un modèle prédit bien un échantillon. La "nouveauté" *Novelty* montre la différence entre les phrases générées et les phrases présentent dans le jeu de données. La "diversité" *Diversity* met en évidence la diversité parmi les phrases générées. Enfin l'intelligibilité, mesurée par des humains, permet d'évaluer la compréhension des phrases, leur sens. Nous nous attarderons surtout sur ce critère pour les phrases générées à partir des différents dataset et modèles.

```
'several years  ago this place is a great deal the ambience is'
'like evil  dead 2 or phantasm that people will be back again if needed not only had it for a long time'
'love it  the accompanied software is almost brilliant headset works great plantronics bluetooth excelent buy i highly recommend this movie is a'
"good beer  selection i didn't see a wonderful story about loneliness and tony has built one of the film is an excellent"
```

FIGURE 4.2 – Exemple de phrases positives

```
"i don't  think i've ever had a few seconds into the case"
'big fan  of beat this place is not a good way i'
'and balanced  perfectly between overacting and underacting and finally after all the dialogue is composed of things that make little sense about'
'money this  was a bit of a movie that is screened in the proceedings is totally implausible and unconvincing and if someone'
```

FIGURE 4.3 – Exemple de phrases négatives

```
'wish you  could have to do some studying to come up to'
'forensic files  n is your first tweet of the day off to'
"atmosphere on  board we're all grateful for the next time i have"
'were your  first tweet of the day off to work today kaitlyn still bleh her word i knew it was a good'
'thanks to  the gym bye missxmarisa haha i have a great day everyone got a new background for me and i have'
```

FIGURE 4.4 – Exemple de tweets positifs

```
'lot going  to be a bit of the film is a great deal of this movie is a great deal of this'
'each other  in the film is a great deal of this movie is a great deal of this movie is a great'
```

FIGURE 4.5 – Exemple d'avis de films positifs

Les figures 4.2 et 4.3 montrent des avis positifs et négatifs générés par un réseau LSTM à partir du jeu de données **ReviewsLabelled**.
La figure 4.4 montre des avis positifs générés par un réseau LSTM à partir du jeu de données **Sentiment140 dataset**.
La figure 4.5 montre des avis positifs générés par un réseau LSTM à partir du jeu de données **Movie data**.

On constate que les phrases générées ne sont pas forcément correctes. Bien que les mots soient tous bien formés de part la présence des embeddings, le sens global des phrases est difficile à saisir. En particulier, les phrases n'ont pas vraiment de début ni de fin. On remarque également la présence de "boucles" avec une séquence de mots qui se répètent comme dans l'exemple 4.5 Autrement la qualité semble augmenter avec la taille du jeu de données. En effet, les phrases générées à partir du jeu de donnée **Sentiment140 dataset** ont l'air plus réalistes.

```
this is a great game that you should get to play this now ! it is a little different then find pieces to figure others over ... .
ok calendar i am traveling . you will want to be almost 3 fun with weapons a step the killing of the game , but this app is a hard app
i love it . they suck and this is a little odd to find out easily . spend money too rack up the same time i recommend this .
game have to be any good time it is fun and it makes your own pace . keep the little addicting .
```

FIGURE 4.6 – Exemple d'avis Amazon SentiGAN

La figure 4.6 montre des avis générés à l'aide du réseau SentiGAN à partir du set **Amazon app book**. On peut voir que les avis semblent de meilleure qualité, notamment le problème du début et de la fin des phrases semble résolu.

4.3.4 Accuracy

Dans cette partie, nous avons testé l'*accuracy* des données générés en utilisant un classifieur. Pour commencer nous avons donc entraîné un classifieur sur le jeu de données **ReviewsLabelled**. Afin de déterminer quel est le meilleur type de classifieur, nous en avons testé plusieurs. On citera parmi eux *LogisticRegression, RandomForestclassifier* ou encore SVM.
Nous avons utilisé la cross-validation, on entraîne chaque classifieur sur une partie du set (jeu d'entraînement) puis nous les testons sur l'autre partie (jeu de test). Nous avons par la suite choisi le classifieur SVM qui nous as donné la meilleure accuracy sur le jeu de test.

Nous obtenons donc un classifieur SVM sur le jeu de données ReviewsLabelled qui nous permettra de classifier nos phrases générées. Celui-ci à une *accuracy* de **0.817** sur le jeu de test.

On utilisera également un deuxième classifieur, également SVM, entraîné sur le jeu de donnée **Movie data** pour classer nos phrases générées à partir du jeu ReviewsLabelled (pour éviter les biais). Celui ci a une *accuracy* de **0.97** sur le jeu de test.

Dans les expériences suivantes, nous utilisons toujours la même méthode de test. On génère un grand nombre de données avec notre modèle générateur. On tire ensuite plusieurs échantillons dans l'ensemble, avec remise, et chaque fois on teste l'*accuracy* de notre classifieur sur l'échantillon. Ainsi en relevant les différentes *accuracy*, nous pouvons obtenir une moyenne fiable de celle-ci pour nos données. Nous calculerons également l'écart-type et la médiane qui sont des estimations intéressantes.

Expérimentation 1 On utilise un modèle LSTM pour générer des avis positifs, puis négatifs, sur le jeu de donnée ReviewsLabelled. On test l'*accuracy* sur les données positives, négatives et puis sur les deux ensemble.

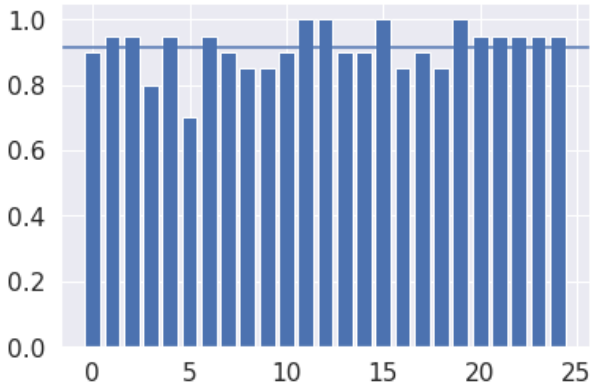

FIGURE 4.7 – Phrases positives

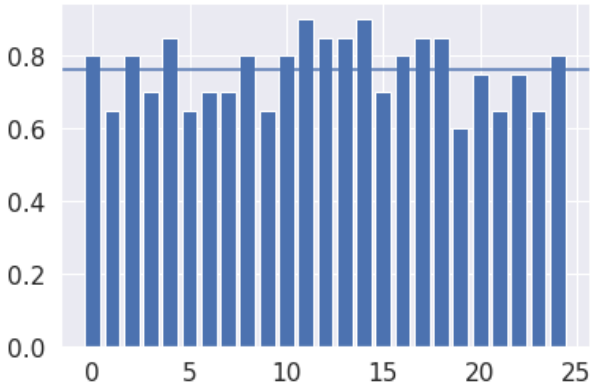

FIGURE 4.8 – Phrases négatives

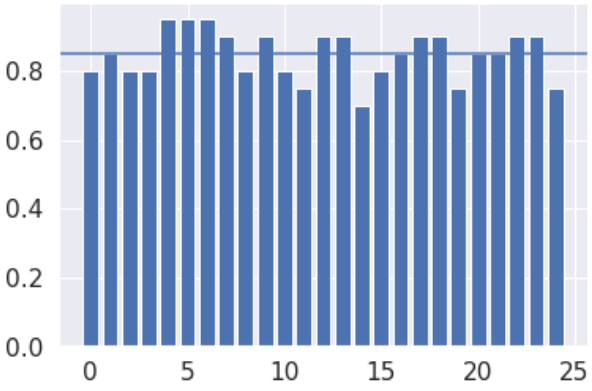

FIGURE 4.9 – Mixte

Les figures 4.7, 4.8 et 4.9 montrent l'*accuracy* obtenue pour chaque échantillon, la moyenne apparaît également sous forme de droite.

Pour les phrases positives, on obtient une moyenne de **0.914±0.069** et la médiane est de **0.95**.
Pour les phrases négatives, on obtient une moyenne de **0.760±0.087** et la médiane est de **0.8**.
Pour les échantillons mixtes, on obtient une moyenne de **0.848±0.069** et la médiane est de **0.85**.

Cette première expérience nous permet de remarquer que les phrases positives générées se classent mieux que les phrases négatives (cela s'explique par différents facteurs, comme l'ironie, qui sont plus difficiles à identifier). Comme on peut s'y attendre, le mélange de phrases positives et négatives se classe parfaitement entre les deux.

Expérimentation 2 On utilise un modèle LSTM pour générer des avis positifs sur le jeu de données Sentiment140 dataset. On teste l'*accuracy* sur les données positives uniquement.

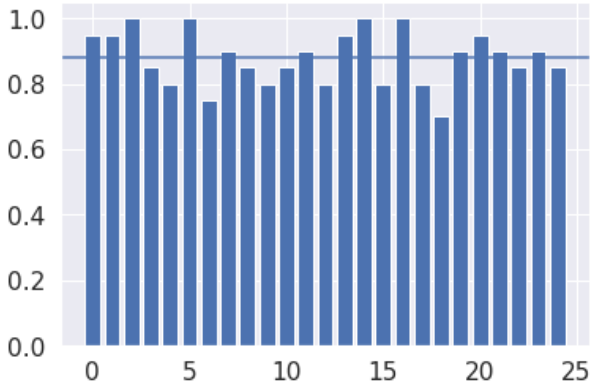

FIGURE 4.10 – Tweets positifs

La figure 4.10 montre l'*accuracy* obtenue pour chaque échantillon, la moyenne apparaît également sous forme de droite. On obtient une moyenne de **0.88±0.08** et la médiane est de **0.9**.
A la différence de l'expérience 1, le type de phrases générées est le même, à savoir des tweets positifs. On remarque que l'*accuracy* est un peu plus basse malgré la taille plus conséquente du jeu de données. On peu donc spéculer que la diversité des données est un paramètre important dans la classification.

Expérimentation 3 On utilise un modèle LSTM pour générer des avis positifs sur le jeu de données Movie data. On teste l'*accuracy* sur les données positives uniquement.

La figure 4.11 montre l'*accuracy* obtenue pour chaque échantillon, la moyenne apparaît également sous forme de droite. On obtient une moyenne de **0.818±0.104** et la médiane est de **0.85**.
Comme dans l'expérience 2, le type de phrases générées est le même, à savoir des avis positifs sur des films. On remarque que l'*accuracy* moyenne est encore un peu plus basse, et l'écart-type est plus conséquent, ce qui semble conforter l'hypothèse précédente.

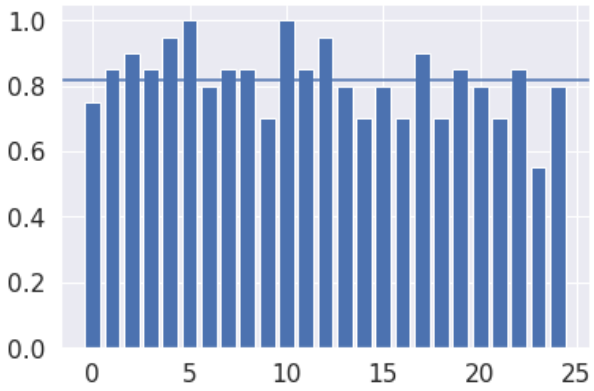

FIGURE 4.11 – Avis films positifs

Expérimentation 4 On utilise le modèle sentiGAN pour générer des avis positifs sur le jeu de données Amazon app book. On teste l'*accuracy* sur les données positives uniquement.

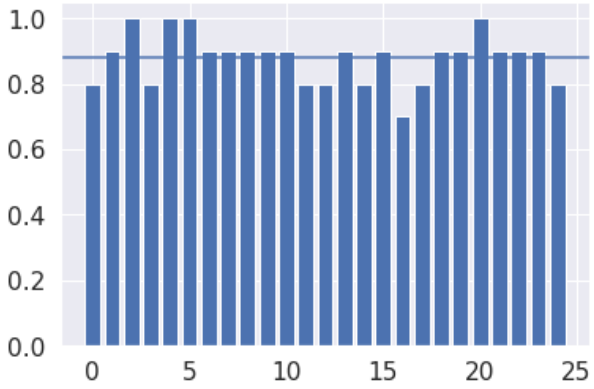

FIGURE 4.12 – Avis livres positifs

La figure 4.12 montre l'*accuracy* obtenu pour chaque échantillon, la moyenne apparaît également sous forme de droite. On obtient une moyenne de **0.88±0.075** et la médiane est de **0.9**.
On obtient à peu près les mêmes résultats que dans l'expérience 2 où l'accuracy est similaire.

Comparaison On constate que l'*accuracy* reste globalement assez élevée quelque soit le dataset et le modèle utilisé. Cependant, on relève des facteurs influents comme la taille du set ou la diversité de celui-ci. Utiliser un modèle générateur de type GAN se révèle intéressant, même si l'*accuracy* va être similaire à un modèle LSTM, nous avons pu voir que la qualité des phrases était supérieure. L'inconvénient reste principalement le temps d'entraînement qui est encore plus long pour ces modèles.

4.3.5 Entraînement d'un classifieur avec des phrases générées

Dans cette seconde partie, on se demande si les phrases générées pourraient être utilisées pour entraîner un classifieur et pouvoir plus tard classer des données réelles. On procède selon le protocole suivant :

1. On génère une grande collection de phrases à partir de notre modèle générateur. Ici on utilise le générateur de l'expérience 3.
2. On utilise le classifieur pour tester nos phrases et on conserve uniquement les phrases classées comme effectivement positives.
3. On remplace une partie des phrases positives du jeu d'entraînement par des phrases de notre collection. Et on ré-entraîne notre classifieur.
4. On teste l'*accuracy* sur les phrases retirées précédemment.

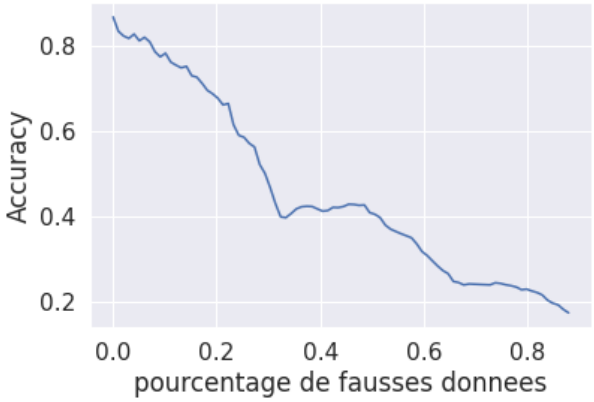

FIGURE 4.13 – Évolution de l'accuracy

On peut donc remplacer une proportion plus ou moins grande du jeu et voir comment notre *accuracy* évolue. Cela est montré sur la figure 4.13 qui représente l'évolution de l'*accuracy* en fonction de la proportion de données remplacées. On remarque que malgré le bon classement de nos phrases, l'*accuracy* décroît avec la proportion de fausses données. On observe une tendance plutôt linéaire (bien qu'il y ait une chute aux alentour de 30% mais cela est sûrement du à la nature aléatoire des phrases remplacées, de même pour la croissance qui suit). On remarque alors qu'il est tout de même possible d'utiliser une faible proportion de données générées pour compléter un set, par exemple avec une proportion allant jusqu'à 10%, l'*accuracy* reste quand même de 0.773.

4.3.6 Biais

Nous avons pu relever différents critères influençant le classement d'une phrase comme étant positive ou négative. Notamment la longueur des phrases, les phrases longue ont plus de chance d'être classées comme étant positives car plus la phrases est longue plus la probabilité qu'un mot à connotation positive apparaisse est élevé. Également, la formation de "boucles" décrites dans la partie **Qualité des phrases** semble influer sur le classement de celles ci.

Chapitre 5

Bibliographie

[Fis] R.A. Fisher. *Iris Data Set.* URL : https://archive.ics.uci.edu/ml/datasets/iris. (accessed : 06.05.2021).

[Goo] Ian Goodfellow. *Generative Adversarial Nets.* URL : https://arxiv.org/abs/1406.2661. (accessed : 06.05.2021).

[Gün] David Gündisch. *Writing your first Generative Adversarial Network with Keras.* URL : https://towardsdatascience.com/writing-your-first-generative-adversarial-network-with-keras-2d16fd8d4889. (accessed : 06.05.2021).

[Ke] Xiaojun Wan Ke Wang. *SentiGAN : Generating Sentimental Texts via Mixture Adversarial Networks.* URL : https://www.ijcai.org/Proceedings/2018/0618.pdf. (accessed : 06.05.2021).

[Mir] Mehdi Mirza. *Conditional Generative Adversarial Nets.* URL : https://arxiv.org/abs/1411.1784. (accessed : 06.05.2021).

[Phia] Michael Phi. *Illustrated Guide to LSTM's and GRU's : A step by step explanation.* URL : https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. (accessed : 06.05.2021).

[Phib] Michael Phi. *Illustrated Guide to Recurrent Neural Networks.* URL : https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9. (accessed : 06.05.2021).

[Wik] Wikipédia. *Algorithme du gradient.* URL : https://fr.wikipedia.org/wiki/Algorithme_du_gradient. (accessed : 06.05.2021).