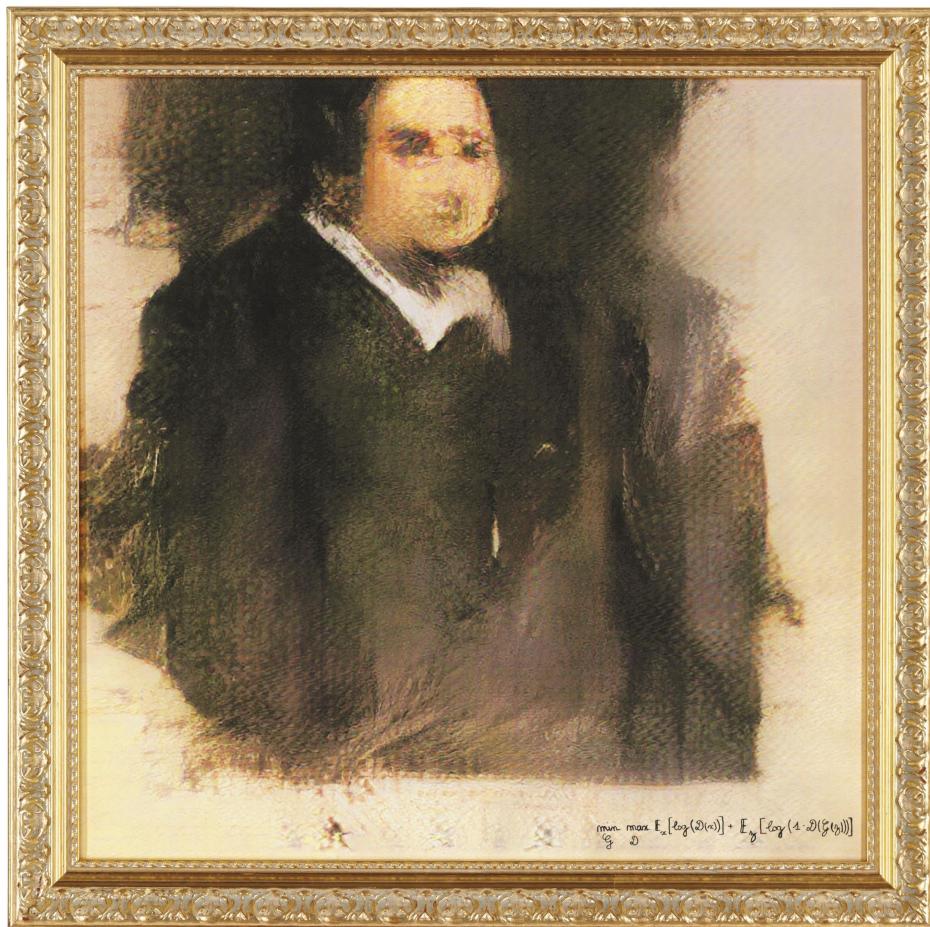

RAPPORT PROJET 2

Réseaux antagonistes génératifs - GANs

Auteurs : (ING4 - Gro2) Enseignant :
Sosthène Paultre de Lamotte M. BOIGE
Louis Donikian



Nous attestons que ce travail est original, qu'il est le fruit d'un travail commun au groupe et qu'il a été rédigé de manière autonome.

Paris, le 02/04/2021

Table des matières

Introduction	3
Présentation	3
Contexte	3
Fonctionnement	3
Description	3
Aspect mathématique	5
Cas d'utilisation	7
Fonctions et algorithmes utilisés	9
Partie pratique	10
Conclusion	15
Bibliographie	15

I. Introduction

1. Présentation

Les **réseaux antagonistes génératifs**, aussi appelés GANs (Generative Adversarial Networks) sont un type de réseaux de neurones artificiels très puissant utilisé en machine learning dans le cas de l'**apprentissage non supervisé**.

Les GANs reposent sur une **mise en compétition de deux réseaux** (ou modèles) au sein d'un même framework. Cette technique d'intelligence artificielle permet de créer des imitations quasi-parfaites d'images ou de tout autre type de données. Cet outil de manipulation et génération d'images peut aussi être implémenté dans des tâches de compréhension/gestion des risques et même dans des processus de guérison dans le domaine de la santé et la pharmacologie.

2. Contexte

Les GANs sont très récents et ont été introduits pour la première fois en 2014 par **Ian Goodfellow**. Il les a développés pour résoudre certains problèmes qu'il rencontrait avec des réseaux de neurones similaires comme la machine de Boltzmann et l'auto-encodeur.

Les GANs et ces deux réseaux ont en commun d'utiliser l'apprentissage non supervisé pour fonctionner. Ces deux derniers ont recours au processus de décision markovien (Chaînes de Markov) qui est très gourmande en termes de calcul algorithmique du à sa **complexité**.

Goodfellow a donc eu recours aux GANs pour gagner une **efficacité significative**.

II. Fonctionnement

1. Description

Comme nous avons pu le mentionner plus haut, un GAN est composé de deux réseaux appelés "**générateur**" et "**discriminateur**". Le premier est un type de **réseau neuronal convolutif** (CNN, Convolutional Neural Network), c'est-à-dire qui n'est pas cyclique et qui s'inspire du cortex visuel des animaux, dont le but est de créer de nouvelles instances d'un objet.

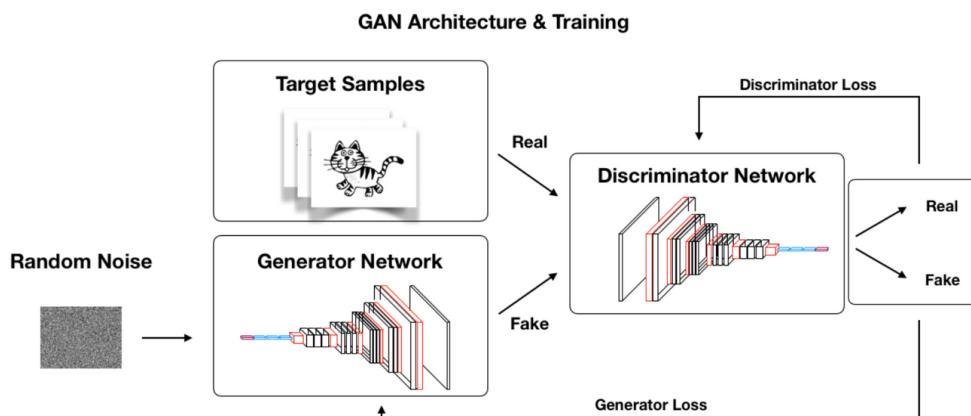
Quant au discriminateur, c'est un **réseau neuronal déconvolutif** qui vérifie l'authenticité de l'objet créé ou son appartenance à la dataset disponible. Les réseaux déconvolutifs sont des CNN qui fonctionnent de manière inversée. Cette application distincte de l'IA recherche et retrouve les signaux ou fonctionnalités perdus au cours de la tâche du CNN, ici le générateur, car jugés non importants à ce stade.

On parle donc dans notre cas d'**apprentissage non supervisé** car l'apprentissage par la machine se fait de manière totalement autonome. Des données sont alors communiquées à la machine sans lui fournir les exemples de résultats attendus en sortie, on dit alors que les données ne sont **pas étiquetées**. Cette solution est ainsi idéale sur le papier car elle ne nécessite pas de grands jeux de données étiquetés.

Si nous prenons l'**exemple** de l'art, une personne qui voudrait contrefaire un tableau va apprendre comment le peintre original a produit le tableau. Pendant ce temps, une seconde personne va récupérer ce que le faussaire apprend et le *critiquer*. Le faussaire est donc le générateur et le critique est le discriminateur.

Le générateur et le discriminateur vont donc apprendre l'un à partir de l'autre, dans un principe de "**compétition**" pour générer des échantillons réalistes à partir des données fournies. On parle donc de "jeu à somme nulle", le gain d'un modèle représente la perte d'un autre modèle.

Au fur et à mesure de l'apprentissage, les données générées vont finir par avoir statistiquement les mêmes données que celles de la **dataset**.

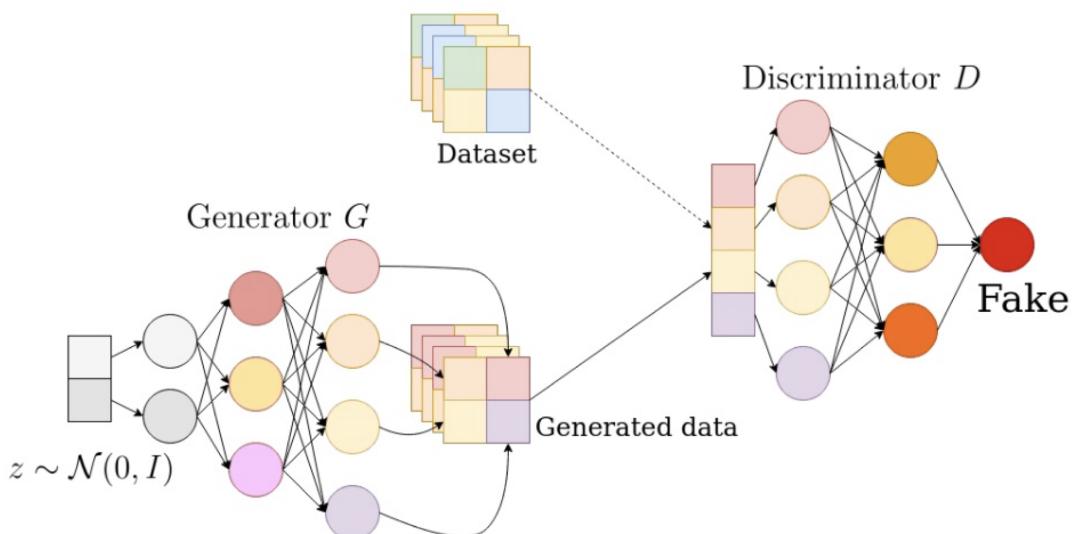


Pour faire fonctionner ce type d'architecture, il convient d'abord de déterminer ce que l'on veut que le GAN doive produire, c'est-à-dire les "**Target Samples**" dans notre image ci-dessus, aussi appelé l'output. Ensuite, il faut créer un ensemble de données basé sur les paramètres étiquetés plus tôt. C'est ce qu'on appelle ici le "**Random Noise**". Ces dernières seront, comme on le voit plus haut, intégrées à l'entrée du générateur jusqu'à ce qu'il produise des outputs de bonne qualité, c'est-à-dire le plus authentique possible.

Dans notre cas de *contrefaçon* d'image, le **générateur** va transmettre les images ainsi produites au **discriminateur** et celles-ci vont être comparées à de véritables images, qui sont donc notre dataset.

Le discriminateur a donc un rôle de filtrage des informations reçues et va établir si elles sont **Vrai** ou **Fausse** entre les données émises par le générateur et celles du dataset. On dit qu'il établit une valeur comprise entre 0 et 1, qui correspond à la **probabilité** d'avoir affaire à un faux (la valeur tend vers 0) ou un vrai (la valeur tend vers 1). Cette valeur sera ensuite transmise au générateur et ce procédé est répété jusqu'à ce que le niveau d'authenticité souhaité soit atteint.

2. Aspect mathématique



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

$$\max_D V(D) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

recognize real images better recognize generated images better

$$\min_G V(G) = \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Optimize G that can fool the discriminator the most.

Cet autre schéma représente aussi l'**architecture** d'un GAN en détaillant de manière succincte le réseau de neurones convolutif puis déconvolutif de respectivement le générateur et le discriminateur.

Pour expliquer mathématiquement, le discriminateur va faire un travail de **classification binaire** lorsqu'il reçoit la data (2 labels : real/fake). Il va donc faire appel à une **fonction coût**. La fonction coût associée à la classification binaire est appelée la **cross-entropy binaire** (*binary cross-entropy*). En Machine Learning, lorsque le nombre de classe J=2, cette fonction s'exprime ainsi :

$$-(y \log(p) + (1 - y) \log(1 - p))$$

Avec y : indicateur binaire (vrai/faux) et p : probabilité

Comme on peut le voir sur les 2 équations plus haut, elles ressemblent grandement à cette dernière. En effet, chaque partie de l'équation a un rôle précis comme :

- Reconnaître les **images réelles** (données réelles de la dataset)
- Reconnaître les **images générées** (données du générateur)
- Optimiser le générateur pour qu'il **trompe le discriminateur** et produise des données quasi-authentique

Ensuite, le générateur est exprimé selon $G(\mathbf{z})$, avec \mathbf{z} la donnée du même type que celle de la dataset et est créé à partir de $p(\mathbf{z})$

Finalement, deux types d'**optimisations** vont avoir lieu :

- Le discriminateur va **discriminer** entre les données *vraies* et *fausses* pour **minimiser** la cross-entropy binaire. **(1)**
- Le générateur va tromper le discriminateur en **maximisant** le *coût* du discriminateur. **(2)**

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```

for number of training iterations do
    for  $k$  steps do
        • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
        • Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
        • Update the discriminator by ascending its stochastic gradient:
```

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D(\mathbf{x}^{(i)}) + \log (1 - D(G(\mathbf{z}^{(i)}))) \right]. \quad \text{(1)}$$

```

end for
    • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
    • Update the generator by descending its stochastic gradient:
```

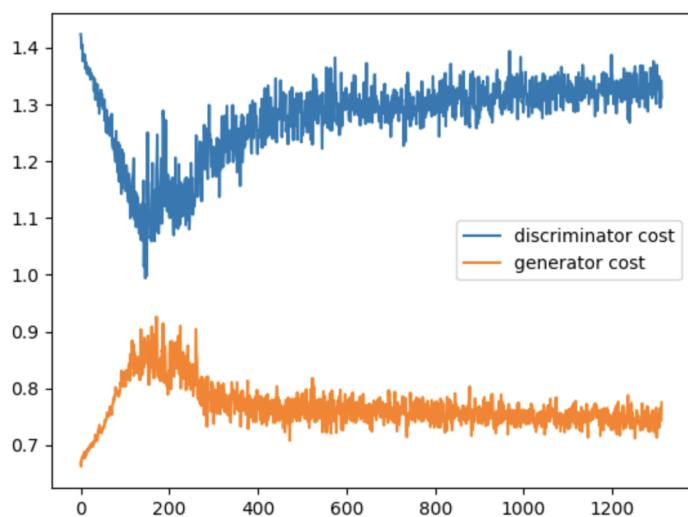
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(\mathbf{z}^{(i)}))). \quad \text{(2)}$$

```

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
```

On peut donc dire que la **fonction coût** du générateur est l'opposé de la fonction coût du discriminant, d'où l'appellation de 'jeu à somme nulle" car la somme des coûts des 2 réseaux neuronaux est toujours égale à zéro.

$\text{Cost}(G) = - \text{Cost}(D)$



Ainsi, nous avons 2 réseaux de neurones différents, 2 coûts différents et donc besoin de 2 optimiseurs différents.

On parle donc de jeu "**minimax**".

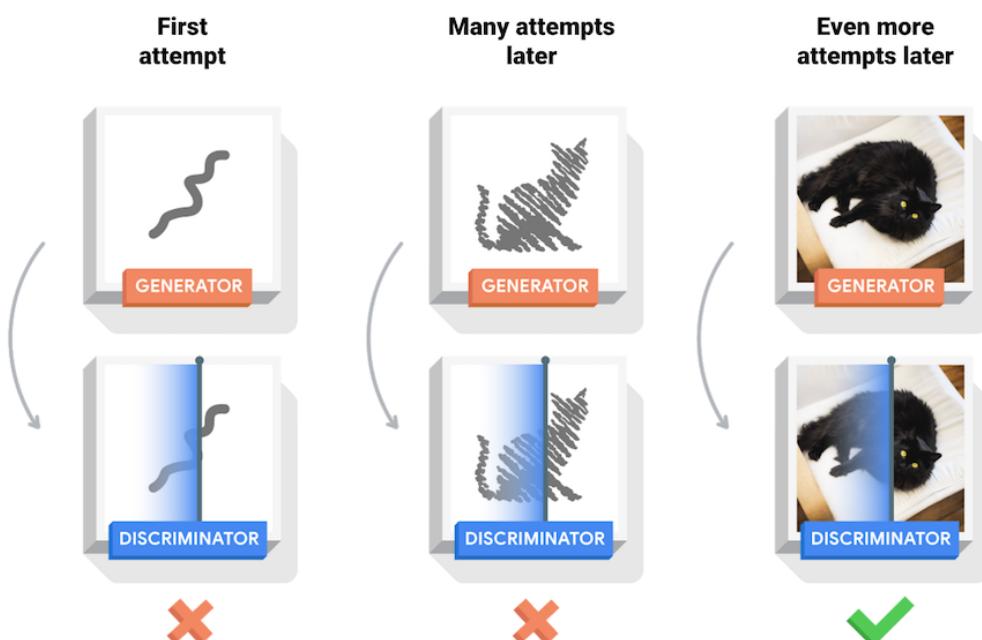
Ce graphe illustre bien ce principe de coût opposé entre le générateur et le discriminateur comme nous l'évoquions plus haut.

III. Cas d'utilisation

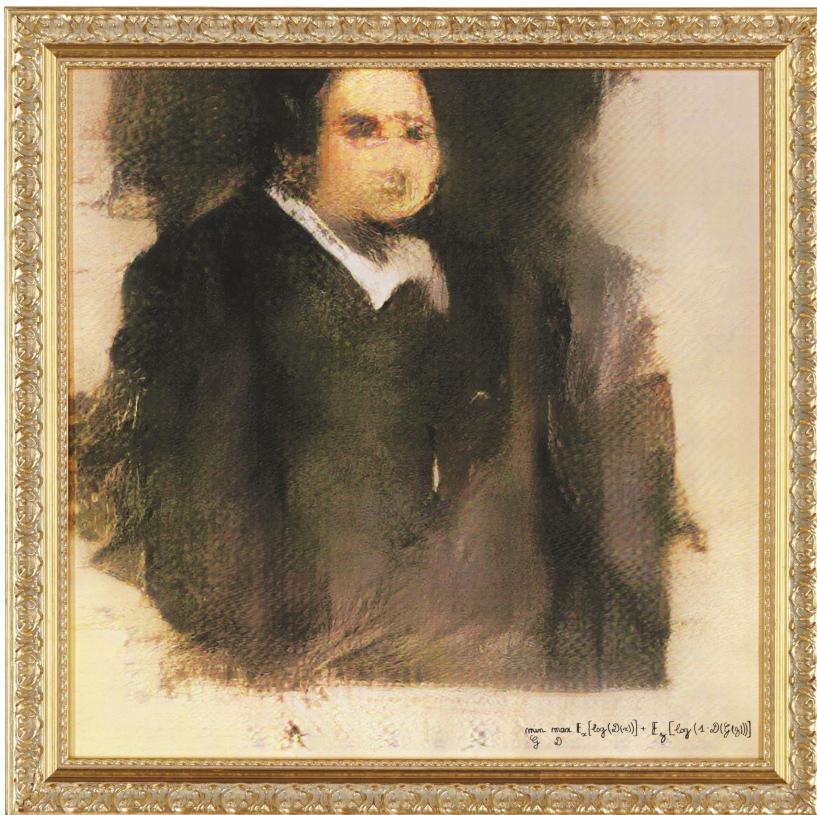
Les GANs sont aujourd'hui utilisés de plein de manières différentes et innovantes.

L'application la plus populaire est la **création d'images** avec le générateur ayant un rôle d'*artiste* et le discriminateur ayant un rôle de *critique d'art*. Le premier apprend à produire des images qui semblent authentiques et le second apprend à distinguer le vrai du faux.

Prenons le cas où le discriminateur va recevoir des centaines d'images labellisées "chat". Les deux modèles vont donc s'entraîner et **apprendre** jusqu'à ce qu'il y ait un équilibre où le discriminateur ne puisse plus distinguer les vraies images des fausses.



L'exemple le plus connu de cette application est la création par 3 étudiants en 2018 du tableau Edmond de Belamy référence à Ian Goodfellow (bel ami se traduit good fellow en anglais).



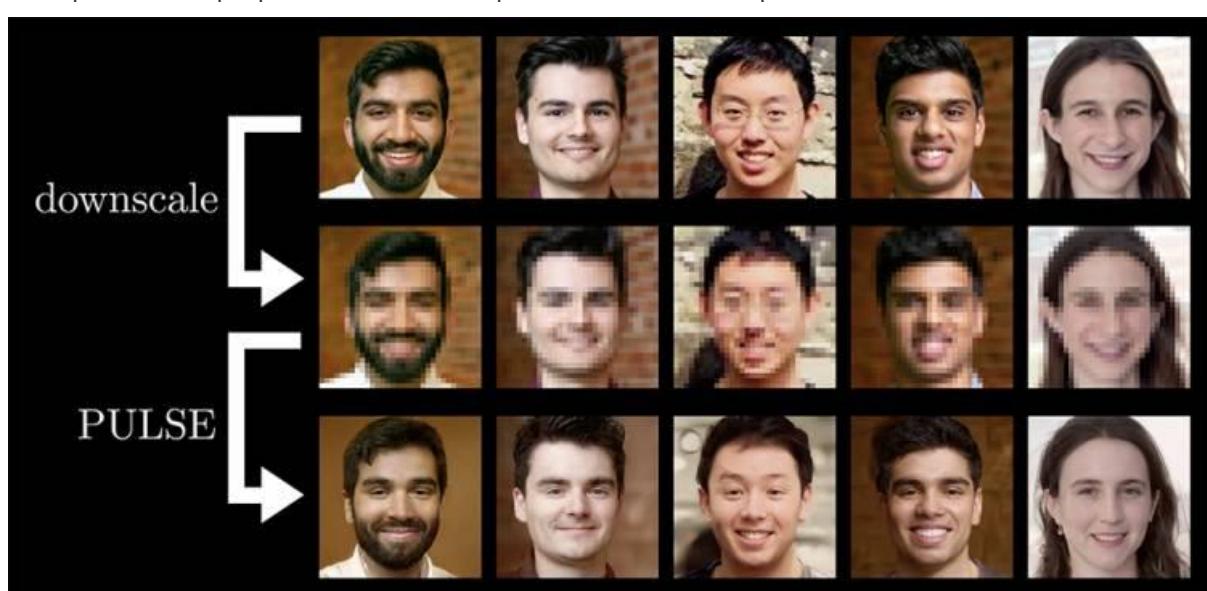
Ce tableau est donc la première œuvre produite grâce à l'intelligence artificielle qui a été proposée en vente aux enchères chez Christie's. Elle a même été vendue \$432.500 ! On peut même remarquer la signature de l'artiste

$\text{min}_G \max_D \mathbb{E}_x [\log(D(x))] + \mathbb{E}_z [\log(1 - D(G(z)))]$

Une référence certaine à l'algorithme des GANs.

D'autres applications sont possibles dans le domaine de l'image comme la **colorisation d'images noir et blanc** et l'**amélioration de la résolution d'une image**.

C'est effectivement ce qu'a pu réaliser des chercheurs de la Duke University en générant un portrait en haute résolution en partant d'une image très pixellisée. Ainsi, la technologie nommée **PULSE** (*Self-Supervised Photo Upsampling via Latent Space Exploration of Generative Models*) est une sorte d'outil de retouche très puissant qui permet de multiplier la résolution par 64.



La technologie n'est évidemment pas parfaite mais montre le potentiel monumental que représentent les GANs.

Une autre application parfois inquiétante est le **deepfake**. Le deepfake est une technique de synthèse d'image basée sur l'architecture GAN où le but est de superposer des fichiers vidéo et audio existants sur d'autres vidéos. C'est un principe de **deep learning** qui va permettre, au fur et à mesure de l'apprentissage du système, de fabriquer de fausses vidéos. Le danger réside dans le risque de partage de fausses informations où une vidéo faisant parler une personne sur des sujets sur lesquels elle ne s'est jamais exprimée. On peut ainsi citer la vidéo où Mark Zuckerberg se vante de contrôler les utilisateurs de Facebook, ou même celle où la présidente de la Chambre des représentants des États-Unis Nancy Pelosi semble ivre.

De manière générale, les GANs permettent de créer n'importe quel type de donnée **à partir de rien**. C'est l'apprentissage des deux modèles qui va permettre au fur et à mesure des itérations de s'autoréguler et produire des données quasi-authentique.

C'est donc pour cela que les types applications sont très nombreuses et parfois méconnues. C'est le cas de la **finance** quantitative pour la **prédiction de données boursières** à partir de dataset financier comme nous allons voir plus bas avec les TimeGAN.

IV. Fonctions et algorithmes utilisés

Les TimeGAN peuvent notamment être utilisés dans la finance ou dans l'énergie car les données d'entrée sont justement étroitement liées au temps, il faut donc inclure cet aspect pour réaliser les meilleures prédictions. Voici un détail des outils mathématiques et techniques qui nous ont permis de réaliser nos expérimentations.

MAE: Mean Absolute Error, c'est une métrique comme la **MSE** (Mean Squared Error), **SSE** (Sum of Squared Errors) ou la **BCE** (Binary Cross Entropy) qui permet d'évaluer la qualité d'un modèle d'apprentissage automatique.

Elle fait référence à la moyenne des **valeurs absolues de chaque erreur** de prédiction de l'ensemble de données de test. L'erreur de prédiction est la différence entre la valeur réelle et la valeur prédite.

$$mae = \frac{\sum_{i=1}^n abs(y_i - \lambda(x_i))}{n}$$

Les auto-encodeurs sont des réseaux de neurones artificiels utilisés pour l'apprentissage **non supervisé** de caractéristiques discriminantes.

L'objectif d'un auto-encodeur est d'apprendre une représentation d'un ensemble de données, et ainsi compresser de manière significative des données complexes sans perdre **d'informations clés**.

En d'autres termes, ils permettent de garder une data de qualité alors que l'on **réduit significativement la taille** de cette data.

Les **RNN** (réseaux de neurones récurrents), sont une classe de réseaux de neurones qui permettent aux prédictions antérieures d'être utilisées comme entrées, par le biais d'états cachés.

Pour une visualisation simplifiée de résultats :

- **T-SNE** : algorithme de réduction de dimensionnalité appelé t-distributed stochastic neighbor embedding est un algorithme d' apprentissage non supervisé. qui permet **d'analyser des données décrites dans des espaces à forte dimensionnalité**. Cet algorithme est très utilisé car il facilite la visualisation de données ayant beaucoup de descripteurs
- **PCA** : L'analyse en composantes principales permet de réduire le nombre de dimensions d'un jeu de données décrit par un grand nombre de variables
- **ROC**: Une courbe ROC (receiver operating characteristic) est un graphique représentant **les performances** d'un modèle de classification pour tous les seuils de classification. Cette courbe trace le taux de vrais positifs en fonction du taux de faux positifs
- **LSTM**: Les modèles de réseau LSTM sont un type de réseau neuronal récurrent (**RNN**) capable d'apprendre et de se souvenir sur de longues séquences de données d'entrée

V. Partie pratique

Pour cette partie nous nous sommes inspirés de cet [article](#) et de son [notebook](#) explicatif. Comme le modèle était pré-entraîné et que nous n'avons pas trouvé le fichier (.h5), nous avons essayé de récupérer un même dataset financier grâce à **Yahoo Finance**. Il permet de facilement récupérer les cours de sociétés selon une durée donnée.

Nous avons donc testé un Gan sur des problématiques financières et notamment pour la prédition de données de **finance quantitative**. Il s'agit dans ce cas de prédition de séries temporelles.

Une **série temporelle**, ou série chronologique, est une séquence de valeurs numériques indexées dans le temps, souvent avec un même pas de temps séparant deux observations successives

Le papier que nous avons utilisé pour implémenter cet algorithme de réseaux adversarial génératif en finance propose une approche par les TimeGan.

"A good generative model for time-series data should preserve temporal dynamics, in the sense that new sequences respect the original relationships between variables across time."

Nous avons donc utilisé les TimeGAN qui répondent à cette problématique, en effet TimeGAN incorpore un nouveau cadre pour générer des données de séries chronologiques synthétiques qui combine à la fois la formation supervisée et non supervisée afin de tenir compte des corrélations temporelles.

L'architecture TimeGAN combine un réseau antagoniste avec un auto-encodeur et comprend les quatre composants suivants :

- Autoencoder: intégration et récupération de réseaux
- Adversarial Network: réseaux générateurs et discriminateurs

Nous utiliserons pour nos agents des **réseaux de neurones récurrents (RNN)** car ils sont idéals pour modéliser les données de séquence telles que les séries chronologiques ou le langage naturel.

Generator & Discriminator

```
generator = make_rnn(n_layers=3,
                      hidden_units=hidden_dim,
                      output_units=hidden_dim,
                      name='Generator')
discriminator = make_rnn(n_layers=3,
                         hidden_units=hidden_dim,
                         output_units=1,
                         name='Discriminator')
supervisor = make_rnn(n_layers=2,
                      hidden_units=hidden_dim,
                      output_units=hidden_dim,
                      name='Supervisor')

def make_rnn(n_layers, hidden_units, output_units, name):
    return Sequential([GRU(units=hidden_units,
                           return_sequences=True,
                           name=f'GRU_{i + 1}') for i in range(n_layers)] +
                      [Dense(units=output_units,
                             activation='sigmoid',
                             name='OUT')], name=name)
```

Ce sont des réseaux de neurones établis sur **Tensorflow** avec l'aide de **Keras**, keras permet notamment de personnaliser les couches du réseau de neurones récurrent avec ici des couches GRU et Dense:

- Les couches GRU vous permettent de créer rapidement des modèles récurrents sans avoir à faire des choix de configuration difficiles.
- Une couche dense est simplement une couche où chaque unité ou neurone est connecté à chaque neurone de la couche suivante.

Les auteurs de TimeGAN proposent un nouveau cadre pour générer des données de séries chronologiques synthétiques qui combine à la fois la formation supervisée et non supervisée afin de tenir compte des corrélations temporelles.

```
Autoencoder Optimizer
autoencoder_optimizer = Adam()

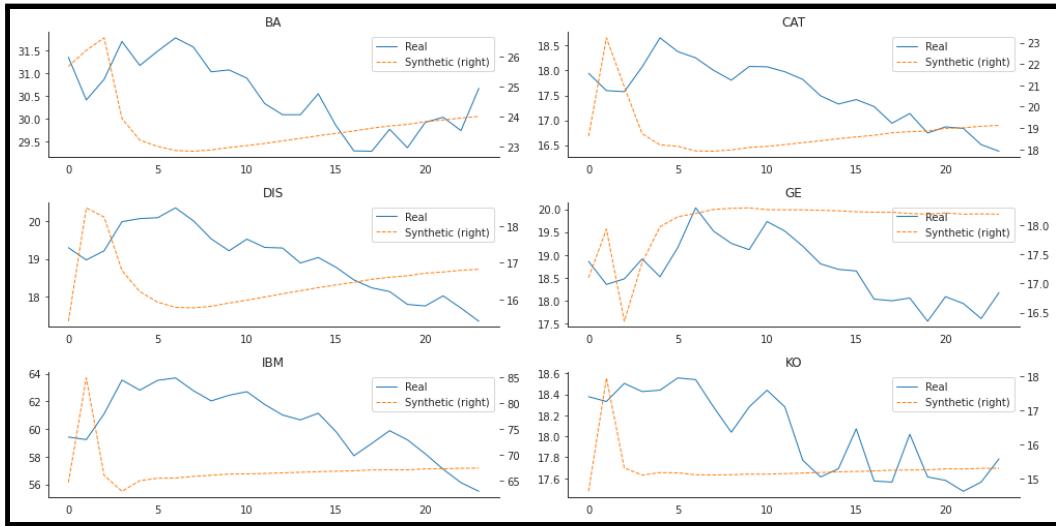
Autoencoder Training Step
@tf.function
def train_autoencoder_init(x):
    with tf.GradientTape() as tape:
        x_tilde = autoencoder(x)
        embedding_loss_t0 = mse(x, x_tilde)
        e_loss_0 = 10 * tf.sqrt(embedding_loss_t0)

    var_list = embedder.trainable_variables + recovery.trainable_variables
    gradients = tape.gradient(e_loss_0, var_list)
    autoencoder_optimizer.apply_gradients(zip(gradients, var_list))
    return tf.sqrt(embedding_loss_t0)

Autoencoder Training Loop
for step in tqdm(range(train_steps)):
    X_ = next(real_series_iter)
    step_e_loss_t0 = train_autoencoder_init(X_)
    with writer.as_default():
        tf.summary.scalar('Loss Autoencoder Init', step_e_loss_t0, step=step)
    100% [██████████] | 10000/10000 [05:20<00:00, 31.23it/s]
    im,
    im,
    im,
    output_units=n_seq,
    name='Recovery')
```

Pour cela, l'architecture des TimeGAN combine un **“adversarial network”** avec un **“autoencoder”** pour garder une donnée de qualité alors que l'on en réduit significativement la taille.

Nous obtenons les résultats suivants pour les 6 cours de bourses qui concernent les entreprises cotées suivantes : 'BA', 'CAT', 'DIS', 'GE', 'IBM', 'KO'.

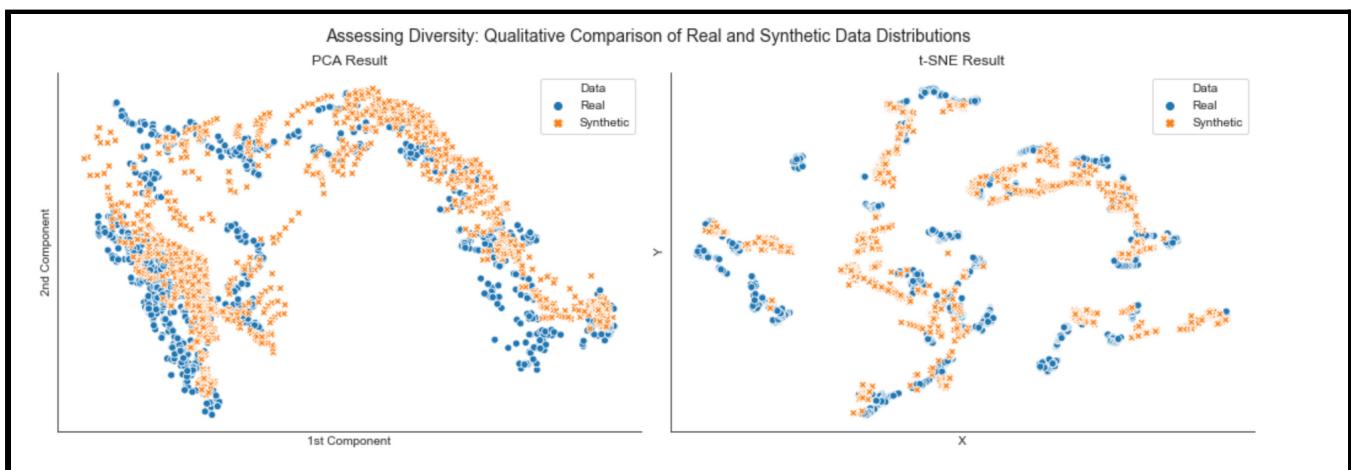


En jaune correspondent les données synthétiques générées par le GAN par rapport au cours réel.

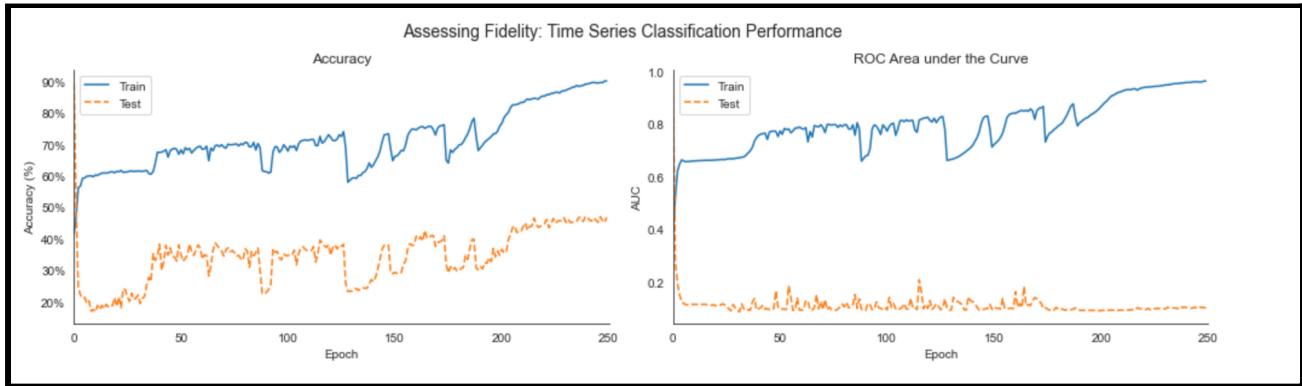
Il y a parfois de grands écarts à cause des caractères particulièrement aléatoires des cours des actions, mais il existe en réalité **3 manières** d'apprecier la qualité de la data générée:

1. **Diversité:** la distribution des données synthétiques doit correspondre à peu près aux données réelles.

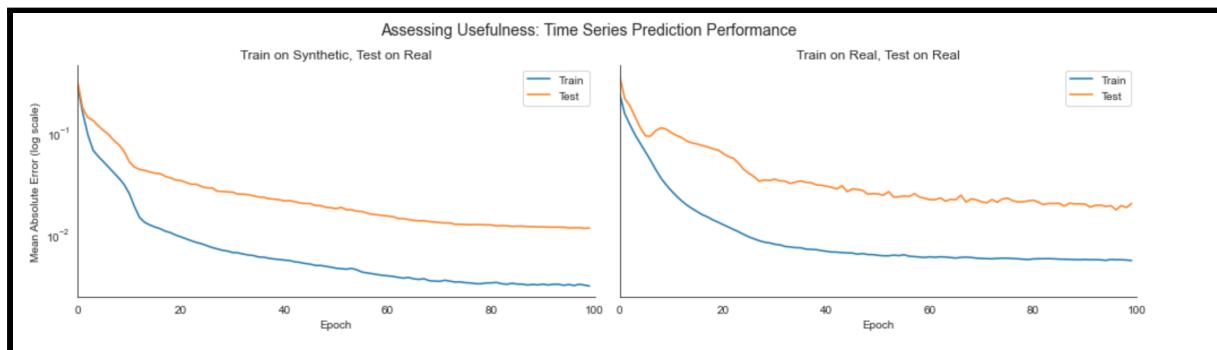
La diversité est évaluée à l'aide de deux techniques de réduction de dimensionnalité un **PCA** (L'analyse en composantes principales) un **t-SNE** (*t-distributed stochastic neighbor embedding*) pour évaluer dans quelle mesure la distribution des données synthétiques correspond aux données d'origine car les points bleu et les jaune ont l'air de se "chevaucher"



2. **Fidélité:** la série d'échantillons ne doit pas être distinguée des données réelles. On peut visualiser ça grâce aux courbes d'accuracy et ROC (receiver operating characteristic) obtenus à la suite d'un **LSTM** modèle de classification.



3. **Utilité:** Les données synthétiques devraient être tout aussi utiles que les données réelles pour résoudre des tâches prédictives.
 Pour l'évaluer, nous comparons les erreurs de "**test**" d'un modèle de prédiction de séries chronologiques formé sur des données réelles ou synthétiques avec les erreurs de "**train**". Pour cela on utilise la **MAE** (mean absolute error). Ce graphique est plus parlant car il indique bien une baisse de l'erreur et donc la preuve d'un apprentissage de plus en plus efficace au cours des **epochs**.



Les résultats obtenus avec cette expérience sont très prometteurs et passionnantes en ce qui concerne la génération de données séquentielles synthétiques.

VI. Conclusion

Pour conclure, la puissance des GANs n'est plus à démontrer et son développement exponentiel est assuré pour les prochaines décennies. Même si cette branche du machine learning peut poser des limites éthiques, il y a une réelle appétence du monde scientifique pour ce type d'algorithme. Nous allons donc voir cette technologie continuer à se développer que ce soit dans le milieu de l'art, de la cinématographie, de la médecine et même de la finance.

VII. Bibliographie

Generative Adversarial Nets for Synthetic Time Series Data :

<https://github.com/stefan-jansen/machine-learning-for-trading/tree/master/data>

Synthetic Financial Data with Generative Adversarial Networks (GANs):

<https://www.mlq.ai/synthetic-data-finance-gans/>

Some Thoughts on the Applications of Deep Generative Models in Finance:

<https://gmarti.gitlab.io/quant/2020/09/27/thoughts-gans-finance.html>

Time-series Generative Adversarial Networks:

<https://papers.nips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf>

GAN ou réseau antagoniste génératif : qu'est-ce que c'est ?:

<https://www.lebigdata.fr/gan-definition-tout-savoir>

What are generative adversarial networks (GANs) and how do they work?:

<https://hub.packtpub.com/what-are-generative-adversarial-networks-gans-and-how-do-they-work/>

Deep Convolutional Generative Adversarial Network:

<https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/generative/dcgan.ipynb>

Réseau Adversaire Génératif Convolutionnel Profond:

<https://www.tensorflow.org/tutorials/generative/dccgan>

Différence entre apprentissage supervisé et apprentissage non supervisé:

<https://www.actua.com/vulgarisation/difference-entre-apprentissage-supervise-apprentissage-non-supervise/>

ML Glossary - Loss Functions

https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

Examples for TimeGan

<https://github.com/jsyoon0823/TimeGAN>

Example for synthetic Time

<https://towardsdatascience.com/synthetic-time-series-data-a-gan-approach-869a984f2239>