Rapport de décembre : Génération d'images (groupe 6)

Lisa Giordani, Mouïn Ben Ammar, Yoldoz Tabei, Ilias Harkati
Tuteur du projet : Nacim Belkhir

01/12/21

1 Présentation du projet

Le deep learning est de plus en plus utilisé en entreprise, mais nécessite une très grande quantité de données. La génération d'images peut donc s'avérer une solution pour des problématiques de data augmentation ou encore de généralisation des modèles. C'est dans ce contexte que ce projet s'inscrit. Il portera sur la génération d'images en s'appuyant sur les deux stratégies principales existantes : les GAN (generative adversarial networks - réseaux antagonistes génératifs en Français) qui sont constitués d'un générateur et d'un discriminateur qui tente de deviner si une image est réelle ou a été créée par le générateur les VAE (variational autoencoders - autoencodeur variationnels en Français) qui sont des estimateurs de distribution non paramétriques

Une application de ce travail pourrait être la data augmentation pour servir des modèles de détection d'anomalies sur des chaînes de production de pièces aéronautiques (chez Safran) par exemple ou encore l'apprentissage de représentations.

De nombreux algorithmes ont été développés par des chercheurs et ingénieurs du domaine et sont disponibles en ligne et/ou dans des papiers scientifiques. Au cours de ce projet, nous devrons réimplémenter et tester certains des algorithmes existants que nous considérons comme pertinents au vu des ressources (GPU) et des compétences dont nous disposons. Pour ce faire, nous pourrons utiliser Google Collab, un petit cluster de calcul de l'ENSTA et un autre de Safran. Nous suivrons un cours pour apprendre à écrire des scripts permettant de lancer un code à distance sur les clusters. Nous écrirons des scripts, packagerons le code et nous serons vigilants au fait que le training puisse redémarrer seul à partir d'un checkpoint dans le cas où il s'arrêterait. M. Belkhir pourra alors récupérer le code que nous auront poussé sur GitHub et le lancer sur un cluster de Safran. Afin de pouvoir juger des performances des algorithmes implémentés, nous devrons aussi déterminer des indicateurs permettant de discriminer les algorithmes. Ces indicateurs pourront se baser sur la cohérence et la variété des images générées.

2 Objectifs du projet

Le projet se découpe en trois grandes parties :

1. Recherche et formation : constitution d'une bibliographie
2. Implémentation d'algorithmes
3. Présentation de nos recherches et de résultats via un blog et une soutenance

Les objectifs du projet sont les suivants :

— Réaliser un état de l'art sur le domaine des modèles génératifs d'images
— Ré-implémenter au moins un algorithme existant par nous-même

— Implémenter un algorithme de mesure de la qualité des images générées
— Tester l'algorithme de génération d'images implémenté sur une base de données (exemple : segmentation d'images de routes comme Cityscape si cela est possible, MNIST, SVHN, CIFAR)
— Mesurer ses performances en termes de cohérence et de diversité des images générées

Les rendus attendus sont :

— Fichiers de code
— Images générées
— Blog
— Soutenance oral avec une présentation PowerPoint

3 Recherches bibliographiques

3.1 GAN

GANs were introduced in 2014 by Ian Goodfelllow,and since then this topic opened up a new active area of research which kept on improving till this day and have shown magnificent results and promises to improve.

A GAN aims to solve the complex problem of "what distribution this data came from ?" in a game-theoretic approach, the two players are the Generator and the discriminator, and since an adversarial learning method is adopted, we don't need to care about approximating intractable density functions. In the adversarial training process, the discriminator aims to differentiate reals from fakes by learning $\mathbb{P}(Y|X)$, while the generator tries to fool the discriminator by generating synthetic data from a random noise vector, this is done by learning $\mathbb{P}(X, Y)$ to generate likely (x,y) samples, and by receiving feedback from the discriminator on how real those images look it learns to generate better images while nudging the discriminator to improve simultaneously. This is done by minMaxing the GAN "theoretical" loss function :

$E(G, D) = \frac{1}{2}(\mathrm{E}_{x\sim p_t}[1 - D(x)] + \mathrm{E}_{x\sim p_g}[D(x)])$

$$\underset{\mathbf{G}}{Max}(\underset{\mathbf{D}}{min}\, E(G, D)) \tag{1}$$

GANs have gone a long way since 2014 and are currently applicable in many fields other than simple image generation(reinforcement learning, image denoising,Text-to-Image Translation,Photograph Editing,Video Prediction...) which is mostly used for data augmentation purposes,and can be controlled much more efficiently using controllable and conditional generation, here's a brief showcase of their historic improvements :

— Vanilla GAN : low resolution human face generation
— DCGAN 2016 : image generation
— lsGAN 2016 : image generation
— text-2-image 2016 : generating an image from textual description
— WGAN 2017 : a different loss function
— proGAN 2017 : start from low resolution image and improves from there
— CycleGan 2017 : learns transformation between images of different styles.
— DiscoGAN 2017 : discovering cross-domain relations
— StyleGan 2018 : high resolution image generation

And here's an image showing the performance improvement over the years :

However, with great power comes great responsibility, Training a GAN is a delicate procedure and requires much attention on the various problems that could arise such as one of the two networks overpowering the other, mode collapse(generating one class of images), choosing an appropriate loss function,vanishing gradients(BCEloss), Quality assessment,checkerboard effect, failure to converge.

[1] Ho, J., Jain, A., and Abbeel, P. *Denoising diffusion probabilistic models*, 2020.
[2] Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., and Chan, W. Wavegrad : *Estimating gradients for waveform generation*, 2020b
[3] Alex Nichol, Prafulla Dhariwal *Improved Denoising Diffusion Probabilistic Models*, 2021
[4] Alex Nichol, Prafulla Dhariwal *Diffusion Models Beat GANs on Image Synthesis*, 2021

3.2 VAE

Les variational autoencoders (VAE) sont souvent associés aux autoencodeurs en raison de leur ressemblance architecturale, mais ces modèles ne s'appuyent pas sur les mêmes principes de résolution du problème de génération d'images.

L'**autoencodeur** est un réseau de neurones dont l'objectif est d'apprendre une fonction d'identité entre une image d'entrée originale et cette image reconstruite, de manière non supervisée. Ce modèle est composé de deux réseaux : un encodeur et un décodeur. L'encodeur compresse les données de l'image d'entrée afin de découvrir une représentation plus efficace et compressée dans ce qu'on appelle l'espace latent (processus de réduction de dimensions). Ensuite, le décodeur essaie de reconstruire l'image d'entrée à partir de sa compression. L'idée est née dans les années 1980 et a été théorisée en 2006 par l'article fondateur de Hinton Salakhutdinov [1].

Le modèle de l'autoencoder a été rapidement confronté au risque de sur-apprentissage notammennt lorsqu'il d'apprendre une fonction d'identité avec un nombre de paramètres du réseau supérieur au nombre d'images d'entrainement. Pour éviter l'overfitting et améliorer la robustesse, la méthode appelée **"denoising autoencoder"** (2008) [2] propose de corrompre partiellement l'entrée en ajoutant des bruits ou en masquant certaines valeurs du vecteur d'entrée de manière stochastique. Ensuite, le modèle est entraîné pour récupérer l'entrée originale (pas corrompue). Cette méthode fut inventée 4 ans avant la technique du dropout (Hinton, et al. 2012).

Une autre méthode permet également de réduire le risque de sur-apprentissage : elle a été mise en oeuvre sou le nom d'**autoencodeurs "sparse"**. Cette méthode applique une contrainte "sparse" sur l'activation des neurones cachés. Elle oblige le modèle à n'activer qu'un petit nombre de neurones cachées en même temps. Cette méthode améliore ainsi la robustesse du modèle. En 2013, Makhzani et Frey ont adapté cette méthode sous le nom d'entoencodeurs k-sparse [3]. Cette fois-ci uniquement les k activations les plus élevées dans la couche du goulot d'étranglement sont conservées à l'aide d'une fonction d'activation linéaire.

D'une manière similaire, l'**autoencodeur contractif** (2011) [4] encourage la représentation encodée à rester dans un espace de petite dimension. Un terme est ajouté dans la fonction de perte pour pénaliser les représentations encodées qui seraient est trop sensibles à l'entrée. Cette technique de pénalisation améliore la robustesse du modèle en réduisant le risque de sur-apprentissage.

L'**autoencodeur variationnel** (2014) [5] est proche de l'autoencodeur par son architecture mais different par son principe d'encodage. Au lieu de convertir l'entrée en un vecteur latent, les VAE convertissent l'entrée en une distribution latente. L'objectif est d'approximer la distribution réelle des images d'entrée, de la manière la plus précise possible. L'encodage consiste donc à déduire des images d'entrée, les paramètres (moyenne et écart-type) de la distribution gaussienne multivariée latente.
Ce principe d'approximation des VAE s'appuie sur des méthodes variationnelles bayeisiennes (VB). Ces dernières permettent de réécrire les problèmes d'inférence statistique en tant que problèmes d'optimisation. Cette dualité inférence-optimisation permet d'utiliser des algorithmes d'optimisation (ex : descente de gradient) pour résoudre des problèmes d'apprentissage automatique statistique.

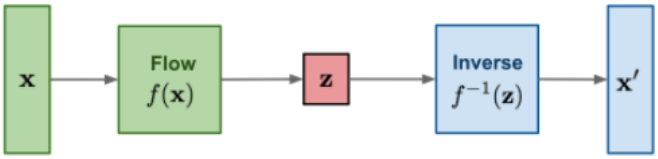
De nombreuses variations du modèle initial du VAE furent développées pour l'adapter à différents domaines et améliorer ses perfomances et :

- **CVAE** pour Conditional VAE (2015) [6] : ajoute des informations labelisées dans l'espace latent afin de forcer une représentation contrainte déterministe des données apprises.
- **Beta-VAE** (2017) [7] : introduit la notion de facteurs latents "disentangled". Une représentation dans l'espace latent est dite "disentangled" si chacune de ses variables n'est sensible qu'à un seul facteur génératif. Une représentation "disentangled" induit souvent une bonne interprétabilité et une généralisation facile à une variété de tâches.
- **VQ-VAE** pour Vector Quantised VAE (2017) [8] : utilise la quantification vectorielle (VQ) qui est une méthode permettant d'associer des vecteurs à K dimensions avec un ensemble fini de vecteurs "codes" (par une méthode similaire à l'algorithme KNN). L'encodeur apprend ainsi une variable latente discrète. Les perfomances sont meilleures dans les domaines liés au langage, à la parole et au raisonnement, car les représentations discrètes sont plus naturelles pour ces problèmes.
- **VQ-VAE-2** (2019) [9] : adapte le modèle du VQ-VAE de façon hiérarchique (deux niveaux) combiné à un modèle autorégressif d'auto-attention.
- **TD-VAE** pour Temporal Difference VAE (2019) [10] : s'appuie sur un modèle état-espace, des état de croyances ("belief state") et des prédictions par bonds ("jumpy prediction"). L'avantage de ce modèle est qu'il fonctionne avec des données séquentielles.

[1] *Reducing the Dimensionality of Data with Neural Networks*, G. E. Hinton, et al. (2006)
[2] *Extracting and Composing Robust Features with Denoising Autoencoders*, Pascal Vincent et al. (2008)
[3] *k-Sparse Autoencoders*, Makhzani et Frey (2013)
[4] *Contractive Auto-Encoders : Explicit Invariance During Feature Extraction*, Rifai, et al. (2011)
[5] *Auto-Encoding Variational Bayes*, Diederik P. Kingma, Max Welling (2014)
[6] *Learning Structured Output Representation using Deep Conditional Generative Models*, Kihyuk Sohn et al. (2015)
[7] *Beta-VAE : learning basic visual concepts with a constrained variational framework*, Irina Higgins et al. (2017)
[8] *Vector Quantised-Variational AutoEncoder*, van den Oord, et al. (2017)
[9] *Generating Diverse High-Fidelity Images with VQ-VAE-2*, Ali Razavi, et al. (2019)
[10] *Temporal Difference VAE* Gregor et al. (2019)

3.3 Flow-based Generative Models

L'architecture des " flow based models" ressemble à celle des auto-encodeurs originaux, mais elle subit quelques changements fondamentaux. Nous avons toujours une image qui est encodée à l'espace latent et décodée dans l'espace d'image.



Notre décodeur s'appelle un flux et implémente la fonction F et le décodeur s'appelle un flux inverse et implémente la fonction exactement inverse. Cela signifie que nous n'avons pas besoin d'entraîner le décodeur. Nous pouvons former un seul réseau, le flux et l'inférence que nous venons d'inverser pour générer de nouvelles images.

L'autre différence est que, parce que nous utilisons des fonctions inversibles, Z doit avoir la même dimension de X et non une dimension inférieure comme nous l'utilisons dans notre encodeur. Donc, schématiquement, il semble que Z ait la même dimension que X, mais normalement distribué.

Le flux prend X d'une distribution complexe et le mappe en Z qui est une distribution simple à l'aide d'une fonction inversible. On pourrait aussi remonter de Z à X. On peut utiliser cette correspondance entre X et Z pour écrire la probabilité de X que l'on ne sait pas calculer en fonction de la simple distribution de Z.

Nous obtenons selon la règle de la chaîne de variables :

$$p_x(x) = p_z(z)|det\frac{\partial f}{\partial x}|$$

Le premier terme est la probabilité de Z, qui est notre distribution simple. C'est là que nous décidons quelle serait la distribution de l'espace latent. Nous utilisons généralement la distribution normale.

Le deuxième terme a le rôle de s'assurer que cette expression est toujours une distribution de probabilité valide et qui est de somme égale à 1. Mathématiquement parlant, c'est un déterminant de la matrice jacobienne de la fonction de transfert.

Il y a deux conditions de cette fonction de transformation pour qu'elle soit utile dans le modèle génératif :

— Le déterminant du Jacobien doit être facile à calculer.

— Il doit être facilement inversible.

Dans le modèle génératif, la distribution de X est la distribution de nos images d'entraînement et cela peut être complexe. Nous pouvons trouver une fonction unique qui la mappera sur la distribution normale tout en conservant ces conditions, mais l'entraînement sera très lent. C'est pour cette raison que nous utilisons la normalisation des flux.

Un flux n'est pas une fonction unique, mais une composition de fonctions qui transforment progressivement la distribution complexe en une distribution simple.

L'avantage de cette composition est que la fonction inverse est une composition de chacune des fonctions inverses. Cela signifie qu'il suffit que chacune des fonctions intermédiaires vérifie ces conditions pour que tout le flux nous permette de générer des fonctions complexes à partir de fonctions simples.

Ainsi, au cours de l'entraînement, nous apprenons un réseau unique, qui est un flux, lui-même une composition de fonctions qui se transforme en différents blocs collés ensemble. Nous optimisons directement la probabilité des données réelles en utilisant cette expression :

$$max[log(p(x))] = max[log(p(x)) + \sum_{i=1}^{K} log|det\frac{\partial f_i}{\partial x}|]$$

Ensuite, nous pouvons utiliser le flux inverse pour reconstruire X à partir de Z ou pour échantillonner au hasard Z à partir de la distribution normale pour générer de nouvelles images.

Voici un aperçu de quelques améliorations faites :

— **Variational Inference with Normalizing Flows** (2015) [1] : Apprend des densités postérieures hautement non-gaussiennes en apprenant des transformations de densités simples en densités plus complexes par le biais d'un flux de normalisation, et utilise des réseaux d'inférence et une estimation efficace du gradient de Monte Carlo.

— **Sylvester Normalizing Flows for Variational Inference** (2018) [2] : Les flux normalisateurs de Sylvester peuvent être vus comme une généralisation des flux planaires. Ils éliminent le goulot d'étranglement bien connu des flux planaires, ce qui rend une transformation unique beaucoup plus flexible.

— **Self Normalizing Flow** (2020) [3] : Approxime le gradient du déterminant logarithmique jacobien à l'aide d'inverses appris, ce qui permet l'apprentissage d'architectures de flux normalisateurs intraitables.

— **Multi-Resolution Continuous Normalizing Flow** (2021) [4] : Les CNFs à résolution s sont utilisés pour générer des images x_s à partir de bruit à chaque résolution. Chaque CNF produit une image intermédiaire y_s, qui est ensuite combinée avec l'image immédiate $x_s + 1$ pour former x_s. Ce processus est répété jusqu'à ce que l'image de résolution la plus fine x_1 soit calculée.

[1] Danilo Jimenez Rezende, and Shakir Mohamed. *Variational Inference with Normalizing Flows*, 2015.
[2] Rianne van den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling. *Sylvester Normalizing Flows for Variational Inference*, 2018.
[3] T. Anderson Keller, Jorn W.T. Peters, Priyank Jaini, Emiel Hoogeboom, Patrick Forré, and Max Welling.*Self Normalizing Flow*, 2020.
[4] Vikram Voleti, Chris Finlay, Adam Oberman, and Christopher Pal. *Multi-Resolution Continuous Normalizing Flow*, 2021.

3.4 Diffusion Models

Les modèles de diffusion sont une autre classe, relativement récente, de modèle génératif ; le principe derrière ces modèles est d'apprendre à inverser un processus progressif de bruitage de sorte à obtenir, de façon itérative, une donnée débruité en sortie. Ainsi, à partir d'un simple bruit de générer de nouvelles données.

L'idée est donc de partir d'une élement d'un espace d'origine de données réelles, et de lui appliquer de façon itérative un bruit gaussien, un très grand nombre de fois successivement, de sorte à ce qu'à chaque étape t la donnée d'origine soit de plus en plus bruité. Cela un nombre T grand de fois. Avec T suffisamment grand pour obtenir un quasi-bruit gaussien isotrope.

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \quad q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

Et à partir d'une telle donnée bruitée d'entrainer un réseau à effectuer le processus inverse. Ou plutôt à éstimer les paramètres d'une loi - qu'on peut approximer comme normale du fait du caractère progressif du processus - et qui va correspondre à une étape dans l'autre sens.

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \quad p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

Les modèles de diffusion ont été introduit principalement par l'article de Sohl-Dickstein et al. (2015) sur les modèles de diffusions probabilistique.
Plus récemment Ho et al. (2020) dans leur article ont permit d'établir que les modèles génératifs basés sur le score de Song Ermon, (2019 ; 2020) étaient équivalent aux DDPM : les modèles de diffusions probabilistique à débruitement. Dans cet article sur les DDPM Ho et al. (2020) [1] ont montré que les modèles de diffusions permettaient d'obtenir de bons résultats sur les bases de données CIFAR-10 (Krizhevsky, 2009) et LSUN (Yu et al., 2015).
Quelques temps après Chen et al. (2020b) [2] ont montré que ces modèles étaient aussi efficace dans la génération audio.
Nichol Dhariwal(2021) [3] ont optimisé par diverses méthodes les DDPM qui étaient alors très gourmand en ressource et ont réduit le nombre d'étape nécessaire au débruitement ; ils ont également montré que DDPM pouvaient obtenir des résultats comparables aux autres modèles génératifs comme les GAN, et même des scores meilleurs avec des métriques comme le FID (Kynkäänniemi et al., 2019).
Puis dans un autre article Nichol Dhariwal(2021) [4] ont montré que les DNIM pouvaient être plus performant que les GAN dans la synthèse d'image.