ENSAE 3A
Oscar BOUVIER
Thomas JACQUOT

Projet final
Optimal Transport, 20/21

01-02-2022
oscar.bouvier@ensae.fr
thomas.jacquot@ensae.fr

Introduction

L'objectif final de ce projet est d'implémenter un *Generative Adversarial Network* (GAN) reposant sur des notions de Transport Optimal. Pour se faire, nous nous appuierons sur l'article suivant :

Salimans et al. "Improving GANs Using Optimal Transport"(2018)[10]

Ce devoir comporte deux parties. La première partie sera principalement consacrée à la théorie et en particulier à la synthese de l'article cité. La seconde sera, quant à elle, dédiée à l'implémentation que nous avons réalisée[1].

1 Première partie : théorie

1.1 GANs

Les GANs (*Réseaux antagonistes génératifs* en français) sont des modèles génératifs récents introduits par Goodfellow et al. en 2014 [5]. Ils permettent de générer des données grâce à la mise en concurrence d'un réseau générateur g et d'un réseau discriminateur d (également appelé *critique*).

La confrontation entre le générateur et le discriminateur peut être modélisée sous la forme d'un jeu à somme nulle (jeu où la somme des gains et des pertes de tous les joueurs vaut 0). En pratique, le générateur simule des données à partir d'un bruit $\mathbf{z}$. Les données simulées s'écrivent $\mathbf{y} = g(\mathbf{z}, \theta_{\mathbf{g}})$ où g représente un perceptron multicouche de paramètre $\theta_{\mathbf{g}}$. De son côté, le discriminateur cherche à distinguer les données réelles $\mathbf{x}$ des données simulées $\mathbf{y}$. Il se définit comme un perceptron multicouche de paramètre $\theta_{\mathbf{d}}$ de sorte que $d(\mathbf{x}, \theta_{\mathbf{d}})$ renvoie la probabilité que $\mathbf{x}$ soit issu de la distribution réelle p.

L'entrainement du modèle consite à trouver la paire (g, d) permettant d'atteindre un équilibre de Nash. A cet équilibre, le générateur minimise sa perte donnée par :

$$L_g = \sup_d \mathbb{E}_{\mathbf{x}\sim p}\big[\log\big(d(\mathbf{x})\big)\big] + \mathbb{E}_{\mathbf{y}\sim g}\big[\log\big(1 - d(\mathbf{y})\big)\big] \tag{1}$$

1.2 Transport optimal

La théorie du transport optimal s'interesse à l'étude du tranfert optimal de masse. D'un point de vue imagé, le problème sous jacent à cette théorie est le suivant : on cherche à transporter une quantité de terre d'un point A à un point B en faisant en sorte que le "coût" du transport soit minimal. Cette formulation du problème est due à Gaspard Monge et est décrite dans son *Mémoire sur la Théorie des Déblais et des Remblais*.

Dans un contexte de *Machine learning*, le transport optimal se déploie pleinement et permet le passage d'une distribution de probabilté à une autre de manière efficace. Cela necessite d'introduire des notions géomètriques permettant de calculer des distances entre deux distributions. A cet égard la distance de référence couramment utilisée est la distance de *Wasserstein* également appelée *Earth Mover's distance*.

1.3 Application du transport optimal aux modèles génératifs

Dans le contexte de construction de modèle génératif, le but est de construire des modèles capables de générer des données qui seraient de la même distribution que les données d'entraînement. Le but est donc de minimiser la distance entre la distribution de probabilité des données d'entraînement et des données simulées. La première formulation orientée Transport Optimal de la fonction de perte d'un modèle génératif apparaît donc intuitivement en utilisant la distance de *Wasserstein* décrite précédemment.

$$D_{\text{EMD}}(p, g) = \inf_{\gamma\in\Pi(p,g)} \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\gamma} c(\mathbf{x}, \mathbf{y}) \tag{2}$$

avec $\Pi(p, g)$ l'ensemble des distributions jointes $\gamma(\mathbf{x}, \mathbf{y})$ de marginales $p(\mathbf{x})$, $g(\mathbf{y})$ représentant respectivement les distributions de probabilité des données d'entraînement et des données générées, $c(\mathbf{x}, \mathbf{y})$ une fonction de

[1] `https://colab.research.google.com/drive/1_csUJ6TluuugaAFYwpGyh2cmzL_u0Q5f?usp=sharing`.

coût.

Cependant, ce problème d'optimisation est généralement insoluble. Une régularisation appelée la *Sinkhorn's Distance* est introduite par Cuturi [2] et restreint l'ensemble des distributions jointes de probabilités Π_β aux distributions d'entropie supérieure à une certaine constante β.

$$D_{\text{Sinkhorn}}(p, g) = \inf_{\gamma \in \Pi_\beta(p,g)} \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\gamma} c(\mathbf{x}, \mathbf{y}) \tag{3}$$

Cette distance est alors approximée sur des *mini-batchs* de données $\mathbf{X}, \mathbf{Y}$ grâce à l'algorithme *Sinkhorn AutoDiff* proposé par Genevay, Peyré, and Cuturi [4]. Cette méthode permet donc de calculer efficacement et de façon stable sur des GPU une approximation $\mathcal{W}_c(\mathbf{X}, \mathbf{Y})$ de la distance de Wasserstein évaluée sur des *mini-batchs* $\mathbf{X} \sim p$, $\mathbf{Y} \sim g$, décrite dans l'équation 2.

Parallèlement, Bellemare et al. [1] proposent d'introduire une autre distance appelée *Energy Distance* comme base de fonction de perte pour la construction de modèles génératifs. Elle permet intuitivement de rapprocher les distributions entre les échantillons d'entraînement $\mathbf{X}$ et les échantillons générés $\mathbf{Y}$ grâce à un terme attractif tout en éloignant les distributions de mêmes distributions grâce à des termes répulsifs.

$$D_{\text{ED}}(p, g) = \sqrt{2\mathbb{E}[d(\mathbf{X}, \mathbf{Y})] - \mathbb{E}[d(\mathbf{X}, \mathbf{X}')] - \mathbb{E}[d(\mathbf{Y}, \mathbf{Y}')]} \tag{4}$$

Dans sa première forme, la distance choisie est la distance euclidienne et est appliquée sur des échantillons individuels, mais cette formulation est facilement généralisable à n'importe quelle autre type de métriques et à des *mini-batchs*. Cette formulation étendue est appelée *Generalized Energy Distance*.

1.4 Algorithme `OT-GAN`

L'OT-GAN introduit par Salimans et al. cherche ainsi à construire un modèle génératif en combinant les approches par Transport Optimal et *Energy-based*, et en utilisant une structure de réseaux antagonistes similaire aux GANs.

La fonction de perte considérée combine ainsi la *Sinkhorn Distance* et la *Generalized Energy Distance* et s'appelle la *Mini-batch Energy Distance* :

$$D^2_{\text{MED}}(p, g) = 2\mathbb{E}[\mathcal{W}_c(\mathbf{X}, \mathbf{Y})] - \mathbb{E}[\mathcal{W}_c(\mathbf{X}, \mathbf{X}')] - \mathbb{E}[\mathcal{W}_c(\mathbf{Y}, \mathbf{Y}')] \tag{5}$$

où $\mathbf{X}, \mathbf{X}'$ sont des *mini-batchs* indépendamment tirés de la distribution d'entraînement, et $\mathbf{Y}, \mathbf{Y}'$ sont des *mini-batchs* indépendamment générés par le modèle génératif. À noter que, par rapport à la distance de *Sinkhorn* classique décrite dans l'équation 3, cette distance introduit des termes répulsifs permettant de rendre l'objectif statistiquement consistent.

L'architecture antagoniste inspirée par les GANs se retrouve dans l'apprentissage de la fonction de coût $c_\eta(\mathbf{x}, \mathbf{y})$ définie dans l'`OT-GAN` comme la *Cosine Distance* entre les représentations latentes de $\mathbf{x}$ et $\mathbf{y}$ générées par un réseau de neurones profond "critique" v_η. La fonction de coût est donc explicitement définie par :

$$c_\eta(\mathbf{x}, \mathbf{y}) = 1 - \frac{v_\eta(\mathbf{x}) v_\eta(\mathbf{y})}{\|v_\eta(\mathbf{x})\|_2 \|v_\eta(\mathbf{y})\|_2} \tag{6}$$

Finalement l'algorithme `OT-GAN` est donné ci-dessous. Notons que l'optimisation est réalisée par descente de gradient stochastique et que le générateur est mis à jour plus souvent que le discriminateur.

Algorithme 1 : `OT-GAN`

Entrées : n_{gen}, le nombre d'itérations du générateur par rapport au discriminateur. η_0, le paramètre initial du discriminateur. θ_0, le paramètre initial du discriminateur. α, le *learning rate* et N le nombre global d'itérations.

pour $t = 1, \cdots, N$ **faire**
- Tirer deux mini-batch indépendant des données réelles $\mathbf{X}, \mathbf{X}'$ et deux mini-batchs indépendants des données simulées $\mathbf{Y}, \mathbf{Y}'$
- Calculer $\mathcal{L} = \mathcal{W}_c(\mathbf{X}, \mathbf{Y}) + \mathcal{W}_c(\mathbf{X}, \mathbf{Y}') + \mathcal{W}_c(\mathbf{X}', \mathbf{Y}) + \mathcal{W}_c(\mathbf{X}', \mathbf{Y}') - 2\mathcal{W}_c(\mathbf{X}, \mathbf{X}') - 2\mathcal{W}_c(\mathbf{Y}, \mathbf{Y}')$
- **si** $t \equiv 0 \pmod{n_{gen} + 1}$ **alors**
  - $\eta \leftarrow \eta + \alpha \nabla_\eta \mathcal{L}$
- **sinon**
  - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
- **fin**

fin

2 Seconde partie : implémentation

Comme énoncé dans l'introduction, notre principal objectif est d'implémenter l'algorithme `OT-GAN` de l'article [10]. Cependant, afin d'avoir des modèles de comparaisons, nous avons également implémenté d'autres GANs (`BasicGAN` et `DCGAN`). Les références ([6], [3] et [8]) disponibles dans la bibliographie nous ont été utiles lors de l'implémentation.

2.1 Données utilisées

Afin d'implémenter les GANs, nous avons chosit d'utiliser le jeu de données MNIST (Figure 1). Ce-dernier est composé d'images de chiffres écrits à la main. Il s'agit d'un jeu de données très répandu dans la communauté du *machine learning* qui permet de faire des applications simples. Dans l'article étudié les auteurs ont utilisé d'autres jeux de données classiques tels que CIFAR-10, IMAGENET DOGS ou encore GAUSSIAN DATASET.

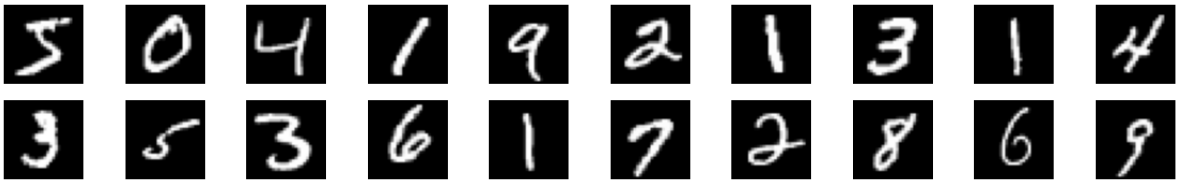

Figure 1: Echantillon de 20 images issues des 60 000 images du jeu d'entrainement MNIST [7]

2.2 Basic GAN and DCGAN

Dans un premier temps, nous avons donc construit un GAN très simple que l'on nommera dans la suite `BasicGAN`. La structure de ses réseaux antagonistes, donnée par les Tables 1 & 2, est relativement simple. Le générateur est constitué de neurones connectés auxquels on applique une fonction d'activation `Leaky ReLU`[2]. La dimension de sortie du générateur est adaptée pour atteindre les dimensions attendues des images du jeu de données MNIST. Le discriminateur reprend une architecture quasi-similaire sauf qu'il prend en entrée une image et renvoie la probabilité que l'image suive la distribution réelle. Cela est rendu possible par l'utilisation finale de la fonction d'activation sigmoide.

En entrainant `BasicGAN` sur 30 *epochs* en prenant comme *learning rate* 2×10^{-4} et une *batch size* de 100, le générateur arrive à produire les images données par la Figure 2. Ces dernières sont plutôt crédibles pour une première implémentation de GAN.

Figure 2: Echantillon de 20 images générées par `BasicGAN`

Ensuite, nous avons implémenté un *Deep convolutional* GAN. Il s'agit d'une autre catégorie de GAN qui a été introduite en 2015 par Radford, Metz, and Chintala [9]. L'architecture retenue suit celle développée dans l'article avec 4 couches de neurones convolutifs (voir les Tables 3 et 4 en Annexe) pour le générateur et le discriminateur. En entrainant `DCGAN` dans des condiitions exactement similaires à `BasicGAN`, on obtient les images de la Figure 3. Notons que la qualité des images produites par le générateur du `DCGAN` est bien plus satisfaisante que celle issues du générateur `BasicGAN`.

[2] L'avantage de `Leaky ReLU` par rapport à `ReLU` réside dans le fait que cette fonction dispose d'un gradient très faible mais non nul pour les entrées négatives.

Figure 3: Echantillon de 20 images générées par DCGAN

2.3 OTGAN

L'implémentation de OT-GAN suit l'algorithme 1. Pour l'architecture des réseaux générateurs et discriminateurs nous nous sommes inspirés de l'annexe B de l'article [10]. Ces architectures ont été utilisées par les auteurs sur le jeu de données CIFAR-10 et se sont avérées efficaces. Comme pour le DCGAN, il s'agit principalement de couches de neurones convolutifs suivis de fonction d'activation GLU pour le générateur et CReLU pour le discriminateur. Nous avons donc réutilisé ces architectures en les adaptant notamment aux dimensions de notre problème (voir les Tables 5 et 6 à la fin de ce rapport).

En exécutant OT-GAN dans les mêmes conditions que les autres mais avec seulement 10 *epochs* (à cause d'un temps exécution particulièrement long), nous obtenons les images générées suivantes :

Figure 4: Echantillon de 20 images générées par OT-GAN

Conclusion

Finalement, les meilleurs résultats sont obtenus par notre implémentation de DCGAN. L'OT-GAN a des résultats similaires au BasicGAN en terme de qualité des image générés mais permet globalement d'obtenir des images plus nettes. Les deux premiers algorithmes sont relativement rapides et s'exécutent en 5 min et 27 min respectivement. L'OT-GAN, quant à lui, a été lancé sur seulement 10 *epochs* et son exécution a durée 110 min. Il semble néanmoins probable qu'un entraînement plus long aurait permis d'obtenir de meilleurs résultats, similaires à ceux obtenus par le DCGAN. Notons cependant qu'à la fin de la première *epoch*, les images générées par le générateur de l'OT-GAN sont de bien meilleure qualité que pour toutes les autres méthodes (voir Figure 5). Cette observation permet de démontrer l'efficacité de l'approche par Transport Optimal dans la définition de l'objectif, plus concrète qu'une approche purement antagoniste, et qui permet au réseau d'obtenir rapidement de bonnes performances. Afin d'améliorer notre implémentation de l'OT-GAN, nous identifions plusieurs pistes :

- **Optimiser notre code**
- **Disposer d'un environnement GPU plus performant afin d'améliorer la rapidité d'exécution**
- **Améliorer le *fine-tuning* de nos hyper-paramètres et l'architecture du réseau.**

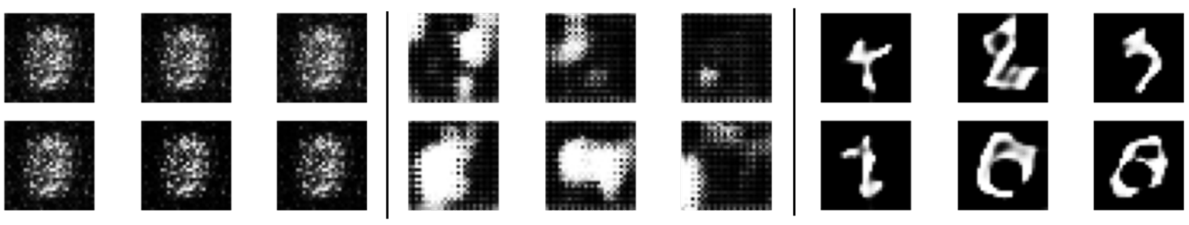

Figure 5: Echantillon d'images générées après 1 *epoch* par BasicGAN (à gauche), par DCGAN (au centre) et par OT-GAN (à droite)

Annexes

operation	activation	negative slope	output shape
z			100
linear	leaky ReLU	0.2	256
linear	leaky ReLU	0.2	512
linear	leaky ReLU	0.2	1024
linear	tanh		784
reshape			$1 \times 28 \times 28$

Table 1: Architecture du générateur de `BasicGAN` pour MNIST

operation	activation	negative slope	dropout	output shape
y				$1 \times 28 \times 28$
reshape				784
linear	leaky ReLU	0.2	0.3	1024
linear	leaky ReLU	0.2	0.3	512
linear	leaky ReLU	0.2	0.3	256
linear	sigmoid			1

Table 2: Architecture du discriminateur de `BasicGAN` pour MNIST

operation	kernel	stride	activation	batch normalisation	output shape
z					100
linear					12544
reshape					$256 \times 7 \times 7$
convolution	3×3	2	ReLU	128	$128 \times 14 \times 14$
convolution	3×3	1	ReLU	64	$64 \times 14 \times 14$
convolution	3×3	1	ReLU	32	$32 \times 14 \times 14$
convolution	3×3	2	tanh		$1 \times 28 \times 28$

Table 3: Architecture du générateur de `DCGAN` pour MNIST

operation	kernel	stride	activation	negative slope	batch normalisation	dropout	output shape
y							$1 \times 28 \times 28$
convolution	3×3	2	leaky ReLU	0.2	32	0.25	$32 \times 14 \times 14$
convolution	3×3	1	leaky ReLU	0.2	64	0.25	$64 \times 14 \times 14$
convolution	3×3	1	leaky ReLU	0.2	128	0.25	$128 \times 14 \times 14$
convolution	3×3	2	leaky ReLU	0.2	256	0.25	$256 \times 7 \times 7$
reshape							12544
linear			sigmoid				1

Table 4: Architecture du disciminateur de `DCGAN` pour MNIST

operation	kernel	stride	activation	output shape
z				100
linear			GLU	25088
reshape				$512 \times 7 \times 7$
2x NN upsample				$512 \times 14 \times 14$
convolution	5×5	1	GLU	$256 \times 14 \times 14$
2x NN upsample				$256 \times 28 \times 28$
convolution	5×5	1	GLU	$128 \times 28 \times 28$
convolution	5×5	1	tanh	$1 \times 28 \times 28$

Table 5: Architecture du générateur de `OTGAN` pour MNIST

operation	kernel	stride	activation	output shape
z				$1 \times 28 \times 28$
convolution	5×5	1	CReLU	$128 \times 28 \times 28$
convolution	5×5	2	CReLU	$256 \times 14 \times 14$
convolution	5×5	2	CReLU	$512 \times 7 \times 7$
reshape				25088
l2 normalize				1

Table 6: Architecture du disciminateur de OTGAN pour MNIST

Références

[1] Marc Bellemare et al. “The Cramer Distance as a Solution to Biased Wasserstein Gradients”. In: (May 2017).

[2] Marco Cuturi. “Sinkhorn Distances: Lightspeed Computation of Optimal Transport”. In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/af21d0c97db2e27e13572cbf59eb343d-Paper.pdf.

[3] *DCGAN On MNIST Dataset Using PyTorch*. https://www.kaggle.com/tjac718/dcgan-on-mnist-dataset-using-pytorch. Accessed: 2021-04-06.

[4] Aude Genevay, Gabriel Peyré, and Marco Cuturi. “Sinkhorn-AutoDiff: Tractable Wasserstein Learning of Generative Models”. In: (June 2017).

[5] Ian J. Goodfellow et al. “Generative Adversarial Networks”. In: (2014). arXiv: 1406.2661 [stat.ML].

[6] *Improving GANs Using Optimal Transport*. https://github.com/openai/ot-gan. Accessed: 2021-04-04.

[7] *MNIST handwritten digit database*. http://yann.lecun.com/exdb/mnist/. Accessed: 2021-04-14.

[8] *PyTorch Generative Adversarial Networks*. https://github.com/eriklindernoren/PyTorch-GAN. Accessed: 2021-04-09.

[9] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. In: (2015). arXiv: 1511.06434. URL: http://arxiv.org/abs/1511.06434.

[10] Tim Salimans et al. “Improving GANs Using Optimal Transport”. In: *CoRR* (2018). arXiv: 1803.05573. URL: http://arxiv.org/abs/1803.05573.