“Amélioration” des GAN avec le GN-GAN
Gabriel ROCHON IASO
Pour le 03/01/25

Résumé :

Ce rapport porte sur l’impact de l'implémentation de GN-GAN et ambitionne de présenter les problèmes qu’il résout et ceux qu’il génère.

Introduction :

Le GN GAN est un ensemble de deux méthodes présentées par Ngoc-Trung Tran, Tuan-Anh Bui et Ngai-Man Cheung en 2018.
Il diffère des GAN classiques (via notamment l’ajout d’un neighbors embedding et d’un gradient matching).
Une nouveauté intéressante du GN-GAN est qu’il introduit un auto-encodeur dont le décodeur est le générateur.
Il a été créé notamment pour résoudre certains problèmes.

Les problèmes des GAN

Dans les GAN “classiques” le générateur peut créer des échantillons similaires. Le mode collapse se produit lorsque le générateur apprend à produire un sous-ensemble très limité des données possibles. Au lieu de capturer toute la diversité des données d'entraînement, il produit des échantillons similaires ou répétitifs. Par exemple, le générateur va produire un seul type d'échantillon qui trompe efficacement le discriminateur. Cette solution est sous-optimale car elle ne reflète pas toute la diversité des données.

L’apprentissage n’est pas toujours stable, en effet si l'un des deux modèles est trop puissant, l'autre peut avoir du mal à s'améliorer. Le GAN peut donc ne jamais atteindre un équilibre stable et produire des échantillons incohérents ou irréalistes.

Pour cela nous implémentons GN-GAN

Voici la méthode proposée par le papier de recherche :

Tout d’abord nous introduisons le neighbors embedding. C’est une méthode qui a pour but de conserver les structures locales des échantillons latents dans l’espace des données générées (un inverse t-SNE).
Le but est d'éviter de générer des données identiques provenant d'échantillons différents.
Le neighbors embedding permet aussi que les clusters qui sont relativement proches dans l’espace latent le soient également dans l’espace des données (il permet dans certains cas d’usage de choisir l’image que l’on veut générer. Si l'on sait que z produit un chat roux, alors z + epsilon va produire un chat roux différent).

Le neighbors embedding améliore donc la diversité et le recall.

Voici mathématiquement comment cela fonctionne.

$$\mathcal{V}_R(E,G) = \sum_i \sum_j p_{i,j} \log \frac{p_{i,j}}{q_{i,j}}$$

Ici E correspond à l’encodeur et G au générateur, ici $p_{i,j}$ vaut

$$p_{i,j} = \frac{p_{i|j} + p_{j|i}}{2n}$$

Et $p_{j|i}$ vaut

$$p_{j|i} = \frac{(1 + ||\mathrm{z}_j - \mathrm{z}_i||^2/2\sigma_\mathrm{z}^2)^{-1}}{\sum_{k \neq i}(1 + ||\mathrm{z}_k - \mathrm{z}_i||^2/2\sigma_\mathrm{z}^2)^{-1}}$$

Où σ_z est la variance de toutes les distances dans un mini batch d'échantillons latents.

Ici $q_{i,j}$ et $q_{j|i}$ valent :

$$q_{i,j} = \frac{q_{i|j} + q_{j|i}}{2n}$$

$$q_{j|i} = \frac{(1 + ||G(\mathrm{z}_j) - G(\mathrm{z}_i)||^2/2\sigma_{\mathrm{x}}^2)^{-1}}{\sum_{k \neq i}(1 + ||G(\mathrm{z}_k) - G(\mathrm{z}_i)||^2/2\sigma_{\mathrm{x}}^2)^{-1}}$$

Le neighbors embedding consiste à minimiser la KL divergence entre la distribution des probabilités dans l'espace latent et la distribution des probabilités dans l'espace des données générées. Il est utilisé pour améliorer l'auto-encodeur.

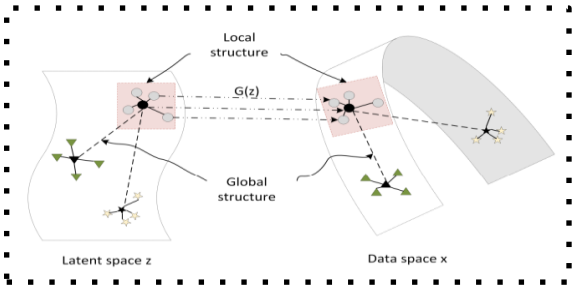

Figure 1 Voici une illustration du neighbors embedding.

Nous introduisons ensuite le Gradient matching.

Il a pour but d'aligner la distribution des échantillons générés et les échantillons réels en utilisant le score du discriminateur.

C'est donc un alignement en termes de valeurs des scores, mais aussi en termes de comportements locaux. Il est censé améliorer la diversité.

Voici comment il se définit mathématiquement:

$$\mathcal{V}_G(D,G) = ||\mathbb{E}_{\mathrm{x}} D(\mathrm{x}) - \mathbb{E}_z D(G(\mathrm{z}))|| + \lambda_{\mathrm{m}}^1 ||\mathbb{E}_{\mathrm{x}}(\nabla_{\mathrm{x}} D(\mathrm{x})) - \mathbb{E}_z(\nabla_{\mathrm{x}} D(G(\mathrm{z})))||^2 + \lambda_{\mathrm{m}}^2 ||\mathbb{E}_{\mathrm{x}}(\nabla_{\mathrm{x}} D(\mathrm{x})^T \mathrm{x}) - \mathbb{E}_z(\nabla_{\mathrm{x}} D(G(\mathrm{z}))^T G(\mathrm{z}))||^2 \quad (1)$$

Le premier terme force le générateur à produire des échantillons qui sont évalués de manière similaire aux données réelles par le discriminateur, donc à le berner.

Le second terme force le discriminateur à réagir de la même manière aux variations locales dans les données réelles et générées. Il fait en sorte que la direction moyenne des gradients soit similaire localement.

Le troisième terme contraint les produits scalaires entre les gradients et les échantillons.

Le gradient matching est utilisé pour améliorer le générateur. En pratique nous avons transformé $E_x[\nabla_x D(x)]$ en $\mathrm{Ex}\|\nabla \mathrm{x} D(\mathrm{x})\|$ et $E_x[\nabla_x D(x)^T x]$ en $\mathrm{Ex}\|\nabla \mathrm{x} D(\mathrm{x}) T\mathrm{x}\|$ pour plus de stabilité.

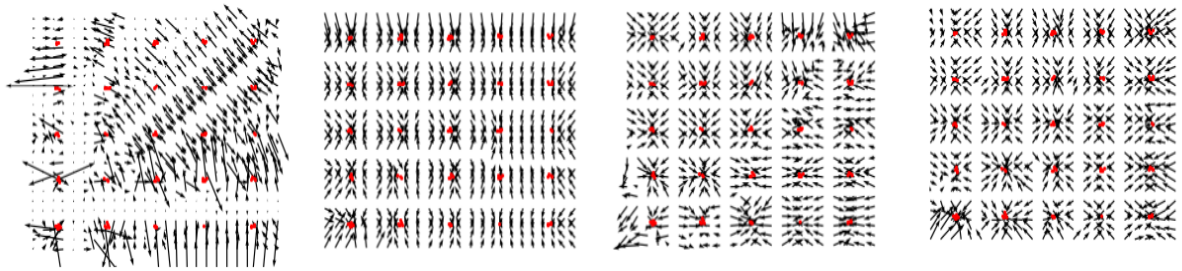

Figure 2 Cette figure montre les cartes de gradient du discriminateur pour différentes méthodes de GAN appliquées à un data-set synthétique 2D avec 25 modes gaussiens. Les figures de gauche à droite sont des exemples de cartes de gradient de GAN, WGAN-GP Dist-GAN et le GN-GAN

Voici l'algorithme général qui a été utilisé :

Algorithm 1 Our GN-GAN model

Initialize discriminator, encoder and generator D, E, G respectively. N_{iter} is the number of iterations.
repeat
  x ← Random mini-batch of m data points from dataset.
  z ← Random n samples from noise distribution P_z
  // Training the auto-encoder using x *and* z *by Eqn. 1*
  $E, G \leftarrow \min \mathcal{V}_{AE}(E, G)$
  // Training discriminator according to Eqn. 7 on x, z
  $D \leftarrow \max \mathcal{V}_D(D, G)$
  // Training the generator on x, z *according to Eqn. 13.*
  $G \leftarrow \min \mathcal{V}_G(D, G)$
until N_{iter}
return D, E, G

Nous implémentons donc un auto-encodeur. On le minimise comme un auto-encodeur classique mais avec l'ajout du neighbors embedding pour essayer de résoudre les problèmes présentés précédemment. Voici la formule mathématique
$VAE(E, G) = ||x - G(E(x))||^2 + \lambda_r VR(E, G)$

Nous avons aussi un discriminateur qu'on va maximiser comme un discriminateur classique mais avec des pénalités. Une des pénalités sera appliquée à la probabilité que D attribue un score élevé aux images venant du data set, ce qui a pour but que le discriminateur ne soit pas trop "fort" et donc d'augmenter la stabilité du modèle.
On va ensuite le forcer à prendre pour vraies les images passées par l'auto-encodeur.
On va enfin contrôler son gradient en le maintenant stable et significatif via une interpolation entre x et G(z). Voici la formule mathématique :

$$\mathcal{V}_D(D, G) = (1 - \alpha)\mathbb{E}_{\mathrm{x}} \log D(\mathrm{x}) + \alpha V_C + \mathbb{E}_{\mathrm{z}} \log(1 - D(G(\mathrm{z})) - \lambda_{\mathrm{p}} V_P$$

Et enfin nous avons un générateur qui a pour but d'aligner la distribution des données réelles et celles générées. On va l'entraîner en le minimisant comme un générateur "classique" avec l'ajout du gradient matching (la contrainte sur les gradients). La formule mathématique est (1) (cf page 2)

Résultat théorique :

Method	CIFAR	STL	CIFAR (R)
GAN-GP	37.7	-	-
WGAN-GP	40.2	55.1	-
SN-GAN	25.5	43.2	$21.7 \pm .21$
Dist-GAN	22.95	36.19	-
Ours	21.70	30.80	$16.47 \pm .28$

On peut comparer cette méthode à d'autres variations des GAN. On peut voir que sur les data-sets de très hautes dimensions nous obtenons un meilleur score FID comparé à ces autres méthodes.

Résultat obtenu :

Empiriquement il y a un gain de temps significatif comparé à un GAN classique pour un résultat moyen. En effet il met 4 fois moins de temps pour atteindre sa performance maximale. Cela est dû au fait qu'il faut moins d'epoch pour avoir la « meilleure » précision.

J'ai donc implémenté les méthodes susdites.

J'ai optimisé les nombreux hyperparamètres notamment les learning rates (tous égaux à 0.00015) et une hausse de la pénalité pour le discriminateur car il était trop "fort" ($\propto = 0.06$). J'ai rencontré une difficulté à optimiser ces paramètres car ils doivent évoluer ensemble. Chacun pris individuellement n'améliore pas significativement le modèle. Ce qui amène un problème computationnel très élevé.

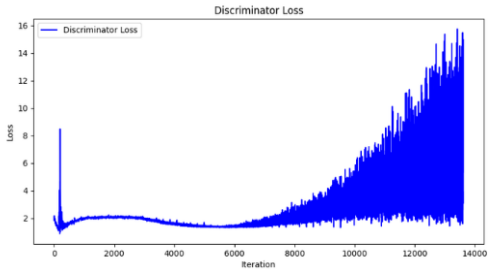

Figure 3 Affichage de la loss du discriminateur avant l'optimisation des paramètres

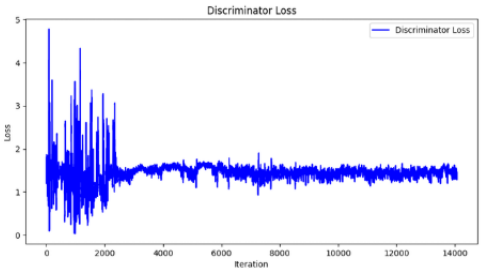

Figure 4 Affichage de la loss du discriminateur après l'optimisation des paramètres :

On peut voir entre la figure 3 et 4 une nette amélioration de la convergence de la perte du discriminateur et donc cela dénote l'amélioration du modèle après l'optimisation des hyperparamètres.

56.23	0.6	0.33

Figure 5 Voici le score obtenu après l'implémentation de GN-GAN sur le data-set Mnist. Le premier terme correspond au FID le second à la précision et le troisième au recall.

On peut voir dans la figure 5 une précision et un recall assez élevés malgré un FID assez faible (le résultat pour un GAN classique est de 27.35 de FID, 0.52 de précision et 0.21 de recall). En effet les images qui suivront ne seront pas très réalistes…

Figure 6 On peut voir le résultat après un GAN classique

Figure 7 On peut voir le résultat après un GN GAN.

On peut voir que la qualité visuelle des images se dégrade.

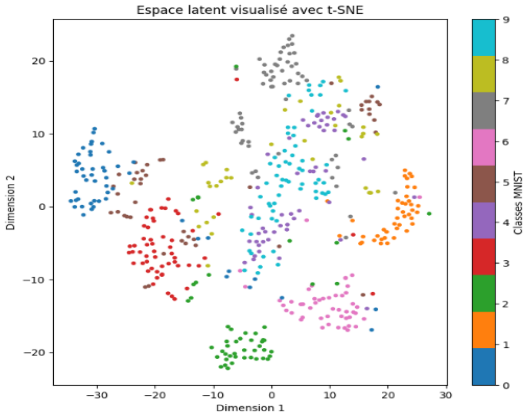

Figure 8 Ce graphique représente l'espace latent visualisé avec t-SNE pour les données du data-set MNIST. On peut voir de nombreux clusters

Limites :

Une des limites est que le modèle apprend très vite une solution sous optimale et n'arrive pas à en sortir.
J'ai eu beaucoup de difficulté à stabiliser le générateur et le discriminateur alors que ce n'est pas censé être le cas.
Le modèle apprend des nouveaux nombres à chaque fois il ne s'améliore donc pas entre chaque epoch ainsi le résultat au bout de 6 epochs est relativement le même qu'après une centaine. Cela peut se voir assez aisément dans la figure 9 présente en annexe.

Explication / piste :

On a donc vu que théoriquement cette méthode est censée améliorer la convergence du modèle. Mais en pratique sur le data-set MNIST elle n'améliore pas la fid voici mes hypothèses.

Cela peut être dû à une optimisation non efficace des nombreux hyperparamètres causé par notamment un manque de puissance de calcul.

Cela peut être causé par mon implémentation qui peut ne pas être efficiente et juste.

Cela peut aussi être dû au fait que le générateur utilisé sur Mnist n'est peut-être pas assez élaboré pour permettre un bon fonctionnement du neighbors embedding car il ne permet peut-être pas de capturer toute la complexité et tend à rendre moins efficace le neighbors embedding.

Et enfin cela peut être dû au fait que la méthode est plus performante sur des data-sets de haute dimension notamment CIFAR et STL car le neighbors embedding apporte une réelle plus-value. Mnist est peut-être de dimension "trop faible".

Conclusion :

Pour conclure le GN-GAN permet empiriquement d'obtenir une solution moyenne très rapidement au moins sur Mnist. Cela peut être utile si nous travaillons par exemple sur des modèles génératifs sans grosse puissance de calcul.
Cependant, il est important de noter que cette méthode présente encore des limites, notamment en termes de stabilité et de qualité des images générées. Des pistes d'amélioration incluent une optimisation plus efficace des hyperparamètres et une implémentation plus juste de la méthode.

Annexe :

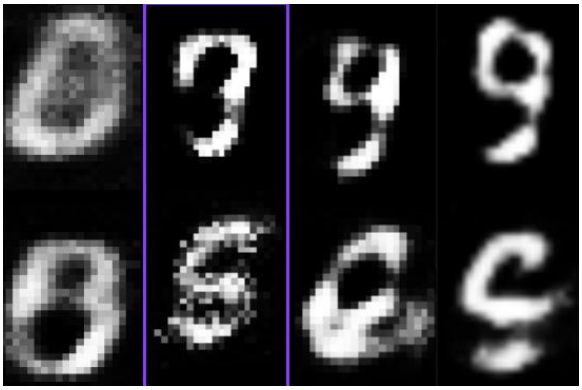

Figure 9 Voici les images provenant du même point de l'espace latent à différentes epoch (respectivement 3, 6, 9, 12).

Référence :

Ngoc-Trung Tran, Tuan-Anh Bui, Ngai-Man Cheung. "Improving GAN with Neighbors Embedding and Gradient Matching." arXiv:1811.01333.