Generative Adversarial Nets

Nicolas HUBERT, Alexandre TRENDEL

Mars 2019

La technologie Generative Adversarial Networks (GAN) présentée par Ian Goodfellow en 2014 vise à modéliser la distribution sous-jacente des données observées. En d'autres termes, au lieu de s'attacher à prédire un label, les GANs font partie des modèles génératifs : ces derniers tentent de modéliser la distribution des données réelles conditionnellement au label attribué à une observation donnée, et de générer des échantillons selon la distribution estimée.

La technologie GAN repose sur un apprentissage supervisé de deux réseaux de neurones s'entraînant mutuellement :

- le générateur crée des *designs* (i.e des données pour lesquelles les valeurs des *features* sont toutes renseignées) ;
- le discriminateur, lui, reçoit ces données du générateur, mais également d'un jeu de données réel préexistant. Son rôle est d'inférer la provenance des designs qu'il observe : sont-ils issus du générateur, ou de la base de données initiale ?

Ce n'est qu'après avoir fait une passe sur l'ensemble d'un *mini-batch* que le générateur et le discriminateur reçoivent un *feedback* sur leurs performances respectives. Il convient de noter que l'opposition entre générateur et discriminateur n'implique pas que l'amélioration des performances de l'un se fasse au détriment du second ; en effet, le générateur et le discriminateur n'apprennent pas simultanément en pratique, et il est possible de les entraîner à des vitesses différentes (k corrections pour l'un, une seule pour l'autre).

A contrario, il s'agit-là d'une relation gagnant-gagnant où le générateur produit des données de plus en plus similaires à ceux issus du jeu de données initial, en même temps que les capacités d'identification du discriminateur s'accroissent.

1 Fonctionnement des GANs

1.1 Le processus d'apprentissage

Le GAN originel contient un modèle génératif G et un modèle discriminatif D. Ces deux modèles possèdent en général une structure de type perceptron multicouches. Un vecteur de bruit z, dont les composantes sont typiquement générées par une distribution gaussienne ou uniforme, est placé en entrée de G. On dit également que z représente les variables latentes. Par exemple, pour une problématique de génération d'images, ces variables latentes seraient la couleur ou encore la forme.

Avec la technologie GAN, on ne contrôle pas la signification de z. On laisse au contraire le processus d'apprentissage l'inférer. D est entraîné à distinguer les échantillons réels des faux échantillons générés par G, pendant que ce dernier est entraîné à produire des faux échantillons devant sembler aussi réels que possible.

Le processus d'apprentissage d'un GAN peut se structurer de la façon suivante :

(i) Le générateur G est alimenté par un vecteur de bruit z. Il produit alors une fausse donnée x_F, associée au label $y = 0$;

(ii) Le discriminateur D est alimenté par la fausse paire $(x_F, y = 0)$ et la vraie paire $(x_R, y = 1)$, issue du jeu de données existant. Le discriminateur étant un réseau de classification binaire, il calcule les pertes respectives pour x_F et x_R, et les combine dans sa fonction de perte agrégée;

(iii) Le générateur G calcule également sa propre perte;

(iv) Les deux pertes calculées sont renvoyées à leur réseau respectif. Les paramètres sont ajustés par rapport à la perte, avec par exemple une descente de gradient;

(v) Ce procédé est répété un certain nombre d'itérations ou jusqu'à une condition d'arrêt.

1.2 D'un problème d'optimisation à la théorie des jeux

Le discriminateur renvoie une valeur $D(x)$ indiquant la probabilité que x soit une observation réelle. Notre objectif est de maximiser la probabilité d'identifier la nature des données : réelles ou générées, i.e maximiser la

vraisemblance des données. Pour définir la perte, on utilise la cross-entropy, que l'on cherche à minimiser, ce qui est équivalent à maximiser la log-vraissemblance. À noter cependant que le classifieur est ici entraîné sur deux *mini-batches* : l'un venant du jeu de données réel noté $x \sim p_{\text{data}(x)}$, l'autre venant du générateur noté $G(z), z \sim p_z(z)$, avec des étiquettes opposées. Sa fonction objectif est alors :

$$J^{(D)} = \max_D V(D) = \underbrace{\mathbb{E}_x[\log D(x)]}_{\text{reconnaître les vraies données}} + \underbrace{\mathbb{E}_z[\log(1 - D(G(z)))]}_{\text{reconnaître les contrefaçons}}.$$

Pour le générateur, la fonction objectif cherche à générer des données avec la plus forte valeur possible pour $D(x)$, traduisant l'incapacité du discriminateur à distinguer les données factices des réelles :

$$J^{(G)} = \min_G V(G) = \mathbb{E}_z[\log(1 - D(G(z)))].$$

Le problème d'optimisation peut être vu sous un nouvel angle : celui de la théorie des jeux. On définit le GAN comme un jeu de minimax dans lequel G veut minimiser V tandis que D cherche à le maximiser :

$$\min_G \max_D V(D, G) = \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))].$$

Dans ce cadre, il peut être intéressant de trouver un point d'équilibre dit de Nash pour lequel les deux "joueurs" (ici, le générateur et le discriminateur) n'ont plus intérêt à modifier leur stratégie – i.e leurs fonctions objectif ont atteintes simultanément un optimum a priori local. La recherche de cet équilibre est cependant complexe.

On peut se demander si au contraire il n'est pas préférable de chercher à instaurer un rapport de force inégal entre D et G. L'idée avancée par Goodfellow est que les GANs permettent implicitement d'estimer le ratio suivant :

$$\frac{p_{data}(x)}{p_g(x)}.$$

Ce ratio étant estimé correctement lorsque D est optimal, on peut tout à fait accepter une domination de D sur G.

Dans la mesure où ce ratio constitue une information que le discriminateur partage avec le générateur, on peut également concevoir les GANs comme un jeu gagnant-gagnant où le discriminateur aiderait ainsi le générateur à confectionner des données de plus en plus vraisemblables.

L'algorithme GAN, décomposé selon les itérations sur G et D, est laissé en annexes. L'algorithme initial se heurte toutefois au *vanishing gradient problem* pour le générateur : le discriminateur gagne souvent très tôt dans les itérations, car il est aisé de distinguer les données réelles de celles générées. De fait, $-\log(1 - D(G(z))$ tend vers 0 et le gradient pour le générateur s'affaiblit, ce qui rend la descente de gradient plus laborieuse. Au lieu de limiter la domination du discriminateur, il est proposé une fonction de coût alternative aux gradients plus importants dans cette situation :

$$-\log(1 - D(G(z))) \text{ devient } \log(D(G(z))).$$

2 Résultats théoriques

2.1 Stratégie du discriminateur et optimum global

Compte tenu de la fonction objectif pour le discriminateur, on minimise la fonction perte du discriminateur (les paramètres de G étant fixés) en :

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

On peut montrer que l'optimum pour le jeu minimax décrit plus haut est unique et est atteint pour $D^*(x) = \frac{1}{2}$. Cela correspond au cas idéal où les samples produits par le générateur sont en fait issu de la distribution p_{data}, soit encore $p_{\text{data}} = p_g$.

2.2 Garanties algorithmiques

L'algorithme ne garantit pas que l'apprentissage conduise à une convergence vers cette optimum ; en pratique, il apparaît que le générateur oscille entre plusieurs distributions proches de la distribution réelle, mais sans nécessairement faire un choix définitif. Très souvent, en effet, les corrections apportées à G ou à D ont un impact défavorable sur l'autre.

2.3 Problème de la concentration de modes

De plus, l'algorithme utilisé ne garantit pas non plus que l'on travaille sur un objectif minimax ; comme les paramètres de G et D sont mis à jour simultanément, il n'y a pas de raison que la descente de gradient conduise à une solution minimax plutôt qu'à une solution "maximin" :

$$\max_D \min_G V(D, G) = \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))].$$

Dans ce second cas, à D fixé, G a tout intérêt à produire des données qui ont déjà fait leur preuve et ont réussi à tromper le discriminateur. L'algorithme peut alors converger mais vers une distribution insatisfaisante, qui ne génère que des exemples trop peu variés.

3 Applications

3.1 Comme modèle génératif

Les GANs trouvent de nombreuses applications comme modèles génératifs. Ils sont préférés à d'autres alternatives du fait de leur tendance à ne pas construire des distributions "moyennées" mais à faire au contraire une sélection.

Dans le domaine de la génération d'images par exemple, cela se traduit concrètement par des images nettes plutôt que par une image floue qui cherche à approcher au mieux un grand nombre d'images d'un jeu de données réel. Cela peut s'expliquer par le problème évoqué plus haut, en 2.2.

Les GANs peuvent alors être utilisés pour générer des images "réalistes" (d'un point de vue subjectif – il n'y a pas vraiment de critère formel pour juger de cela) inédites. Si l'on arrive de plus à donner un sens à la distribution de z, on peut même alors créer de nouvelles données qui mêlent plusieurs caractéristiques identifiées. Un exemple est de générer l'image d'une femme portant des lunettes alors que le jeu de données original ne comportait que des hommes à lunettes et des femmes n'en portant jamais.

3.2 Un modèle génératif rapide

Beaucoup de modèles génératifs s'attachent à décrire le lien entre les valeurs de certaines features dans les observations à générer. Cela leur donne un pouvoir explicatif, mais les rend beaucoup plus complexes à évaluer et à entraîner (et nécessitent souvent de faire des approximations).

Au contraire, les GANs permettent de générer des observations complètes de manière très rapide. De plus, même si ce sont essentiellement des perceptrons qui sont utilisés ici, le principe des GANs se soucie peu des modèles utilisés pour le généreur et le discriminateur.

4 Pistes de recherche contemporaines

Les sujets de recherche les plus suivis sont notamment :

- Le sous-titrage d'images "inversé". Il s'agit de générer des images reflétant le plus fidèlement possible la teneur d'un texte. C'est un des problèmes les plus ardus, en partie parce que les GANs sont alors reliés à d'autres architectures d'apprentissage qu'il faut coordonner ;

- Le *Natural Language Processing* et plus généralement toute problématique liée à la génération de texte, car les GANs ne sont définis que pour des valeurs continues. Pour rappel, le gradient calculé en sortie du disciminateur par rapport aux données synthétiques nous dit à quel point il est nécessaire de modifier les caractéristiques de ces données pour qu'elles ressemblent autant que possible aux données réelles. Or, s'il est possible de procéder avec des valeurs continues, il est impossible d'opérer un ajustement si les données en entrée sont discrètes (ex : la valeur d'un pixel peut être corrigée de 1.075 à 1.1 à la prochaine itération, mais quel sens donner à "chat + 0.25" ? Il faudrait en effet générer un nouveau mot, tel que "chien"). I.Goodfellow cite le *REINFORCE algorithm* comme une piste à creuser pour résoudre ce problème ;

- D'autres sujets brûlants s'attachent davantage à l'optimisation du processus d'apprentissage qu'à des applications concrètes des GANs. Par exemple, le papier *Improved techniques for training GANs (2016)* tente de résoudre le problème de non-convergence durant l'apprentissage des GANs. On rappelle effectivement que les fonctions de coût du générateur et discriminateur ne peuvent être optimisées simultanément. Résoudre ce problème de non-convergence revient à améliorer les performances en apprentissage semi-supervisé et en génération d'échantillons.

5 Annexes

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**

Minimise D

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

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

end for

Maximise G

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Figure 1: L'idée selon laquelle D doit être optimal afin d'estimer le ratio des densités nous conduit à choisir $k > 1$. En pratique, on fixe pourtant souvent $k = 1$. Autrement dit, on entraîne D et G l'un après l'autre, en fixant à chaque itération les paramètres de celui n'étant pas entraîné.