

Le dernier ensemble de tâches ne concerne plus la classification. L'objectif est ici de créer des images similaires à celles vues pendant un entraînement et donc correspondant à la même distribution. Jusqu'ici, entraîner un réseau de neurones signifiait pour nous annihiler l'écart entre une *cible* et l'estimation courante du réseau. Or, en contexte de génération, la cible est la distribution des données ; elle n'est pas connue à l'avance, juste représentée par des échantillons. Comment savoir alors si l'apprentissage a abouti ? Une réponse à ce problème est le GAN, ou Generative Adversarial Network.

Principe général des GAN

Le composant principal d'un GAN est le générateur G , un réseau de neurones capable de transformer une entrée z en image plausible \tilde{x} par rapport à des données de train $x^* \in Data$. Au terme de l'apprentissage, n'importe quel z tiré d'une distribution $P(z)$ doit pouvoir, une fois transformé par G , suivre celle qu'on cible : $\tilde{x} = G(z) \sim P(Data)$.

La difficulté à estimer $P(Data)$ rend l'apprentissage impossible à surveiller directement avec une loss. Il faut donc compléter le setting avec un réseau discriminateur D , censé distinguer les images originales des faux générés en leur attribuant un score de réalisme $s \in [0,1]$. Une image d'entrée x est censée recevoir un score $s = 0$ si $x = G(z)$ (*fake*), et un score $s = 1$ si $x \sim P(Data)$ (*ground truth*). Mais si G apprend $P(Data)$, D se perd et tous les s valent 1... et c'est exactement ce qu'on veut.

Les ambitions des deux unités sont donc opposées, d'où la notion d'*adversaire*. On utilise parfois l'image du critique et du faussaire pour décrire D et G : le faussaire sera au sommet de son art quand il aura su tromper la surveillance du critique, lequel fera tout pour l'en empêcher.

Mathématiquement, étant donnée une entrée de départ z , l'objectif d'apprentissage du GAN est :

$$\min_G \max_D \mathbb{E}_{x^* \in Data} [\log D(x^*)] + \mathbb{E}_{z \sim P(z)} [\log (1 - D(G(z)))] \quad (0)$$

Cela dit, en pratique, G et D sont optimisés en alternance – il faut donc calculer des loss séparées :

$$\max_G \mathbb{E}_{z \sim P(z)} [\log D(G(z))] \quad (1) \qquad \max_D \mathbb{E}_{x^* \in Data} [\log D(x^*)] + \mathbb{E}_{z \sim P(z)} [\log (1 - D(G(z)))] \quad (2)$$

1 Interprétez les équations (1) et (2). Que se passerait-il si on utilisait seulement l'une des deux ?

(1) représente l'objectif du générateur G , qui veut être capable de tromper le discriminateur : littéralement, l'espérance de sa réponse $D(G(z))$ pour la classification d'une donnée générée $G(z)$ doit être maximale, donc proche de 1 – alors que D est censé accorder les plus hauts scores aux données réelles.

(2) représente l'objectif du discriminateur D . Il dépend des scores qu'il attribue aux données réelles $D(x^*)$ et de l'opposé de ceux des données générées $D(G(z))$. Pour maximiser leur somme, il doit donc maximiser les uns et minimiser les autres. On peut lire 'donnée générée' et 'donnée originale' comme des labels à deviner (à valeurs respectives 0 et 1), et dire que D veut juste trouver la bonne classification.

Utiliser seulement l'une des deux équations, c'est annuler le jeu par l'absence d'un des deux adversaires. L'unité qui reste ne fera pas d'efforts. Si l'objectif de D (2) reste et G n'apprend pas, la génération restera très sommaire et la discrimination sera facile. Si l'objectif de G (1) reste et D n'apprend pas, la génération sera aussi très mauvaise car la discrimination ne sera pas là pour pousser G à se perfectionner.

2 Idéalement, en quoi le générateur G transforme-t-il la distribution $P(z)$?

G doit produire des images réalistes similaires aux données d'entraînement. Idéalement, on voudrait qu'il transforme la distribution des entrées $P(z)$ en $P(Data)$, qu'il aura bien estimée à partir des exemples vus. Par ailleurs, il faudrait forcer G à générer des échantillons non seulement plausibles selon $P(Data)$, mais aussi représentatifs de sa diversité. S'il se cantonne à une petite partie, on parle de *mode collapse*.

3 Notez que l'équation (1) n'est pas dérivée de l'équation (0). Cela est justifié par les auteurs pour éviter la saturation des gradients. Quelle aurait dû être la vraie équation ici ?

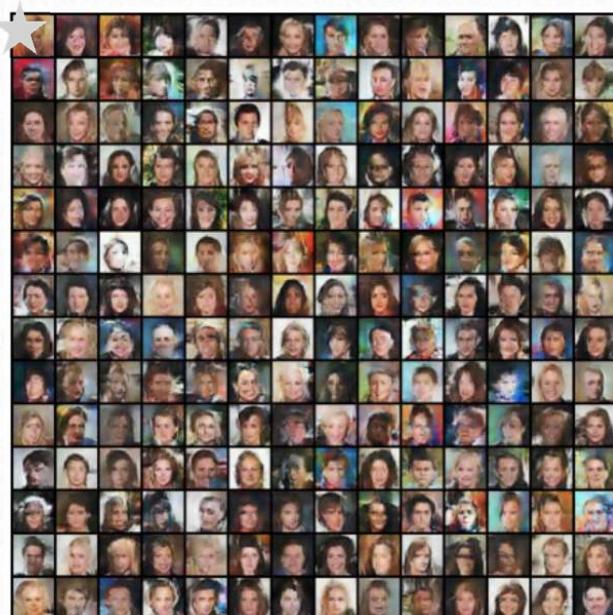
Dans l'équation (0), l'objectif de D (2) apparaît comme argument d'une minimisation. S'il fallait en extraire directement l'objectif de G (1), ce serait donc aussi une minimisation : $\min_G \mathbb{E}_{z \sim P(z)} [\log(1 - D(G(z)))]$.

Architectures des réseaux

Un GAN classique est fondé sur des convolutions (Deep Convolutional GAN, DCGAN) : son G injecte progressivement de l'information spatiale dans un vecteur de bruit z donné en entrée. Notre DCGAN est appris avec la base de visages CelebA et a pour tâche de générer ceux de nouvelles personnes.

4 Commentez l'apprentissage du GAN avec les hyperparamètres proposés par défaut.

Après 8000 itérations (soit dix epochs), le GAN basique produit l'ensemble de fakes suivant. Il n'est **pas optimal** en termes de réalisme et de définition, mais représente bien des visages. Leur diversité est satisfaisante, les angles de vue, traits, coiffures, sourires, palettes générales diffèrent entre tous.



Ensemble 1 – Génération finale obtenue à 8000 itérations avec les paramètres de base. A droite, un des visages générés, et un échantillon à 4000 itérations. Courbes de loss ci-dessous.

L'historique d'affichage montre que les outputs antérieurs étaient convaincants. La structure des visages apparaissait à 800 itérations, était crédible à 2100, et a peu changé après 4000 ; juste des variations des coiffures et couleurs, alors que la précision s'améliorait lentement.

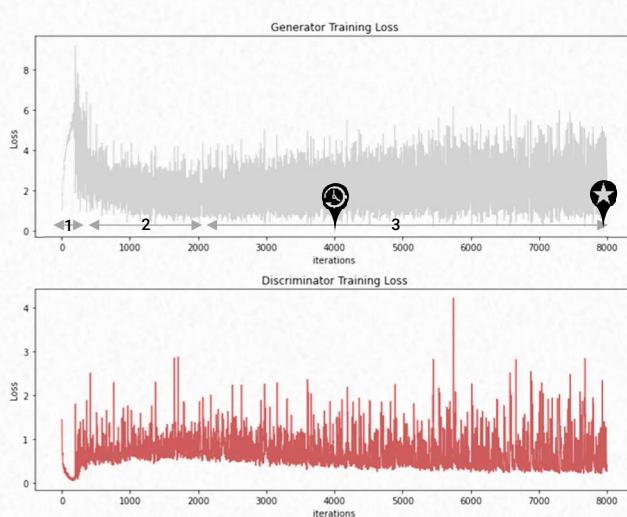


Sur le visage isolé ici, tous les éléments attendus sont présents ; mais il n'est pas tout à fait symétrique, les couleurs ne concordent pas, et la texture générale est inégale. Les cheveux sont ratés. D'autres sont pires.

Côté loss, pas de convergence. Les oscillations sont dues au fait que les générations au temps t peuvent être plus ou moins crédibles pour D , quoiqu'issues de la même distribution estimée (on s'en convainc avec la génération ci-dessus).

Etant donné le contexte adversaire, les loss sont interdépendantes et doivent être lues de manière relative, leur valeur exacte importe peu.

On isole des tendances globales, [1], [2] et [3]. Au départ [1], la loss de D baisse et celle de G augmente, car D apprend vite à séparer le vrai du faux. Puis G s'améliore [2] et sa loss baisse, ce qui déconcerne D . Par la suite [3], il n'y a pas d'équilibre, avec un D légèrement plus fort – capable de faire encore baisser sa loss – et un G en difficulté croissante qui montre des oscillations de plus en plus amples.



Ce résultat sert de référence pour les analyses suivantes. Les notations [1] [2] [3] des phases de loss vont resservir. On souhaite améliorer la performance, qui pâtit peut-être des constantes choisies (10 epochs, learning rate 0.0002, momentum 0.5, taille de l'input z 100) en imitant l'article fondateur.

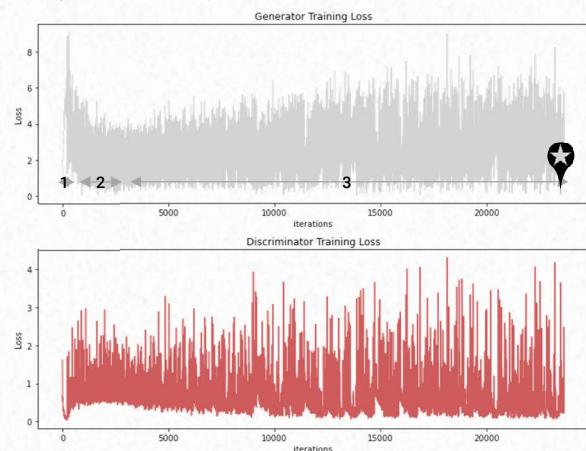
5 Commentez les expériences complémentaires réalisées. Comment influent-elle sur l'apprentissage (vitesse, stabilité, évolution de la loss, qualité, diversité...) ?

SET D'EXPERIENCES A → TRICKS D'APPRENTISSAGE

- (1) On a naïvement pensé que 10 epochs ne suffisaient peut-être pas à bien estimer la distribution des données. On rallonge donc cet apprentissage à 30 epochs pour obtenir le résultat suivant :



Ensemble 2 – Génération obtenue à 30 epochs (26000 itérations). Focus sur des visages générés. Courbes de loss ci-dessous.



Malheureusement, la tendance [3] observée plus tôt sur les loss ne s'inverse pas. D s'améliore encore (même s'il hésite et se trompe régulièrement) et G perd, sa loss augmente. Visuellement, le résultat n'est pas plus parlant, avec toujours des aberrations et des visages qui ne sont pas beaucoup plus détaillés.

- (2) On pense alors à augmenter le learning rate des deux unités (0.001) pour faire mieux en autant d'epochs. Cette fois, les couleurs sont ternes, mais les visages plus réalistes (plus *lisses*) : le visage isolé ci-dessous contrefait une photo floue (une stratégie intelligente de la part de G). La qualité et l'harmonie des couleurs et textures est meilleure qu'avant même si la symétrie des traits reste imparfaite.

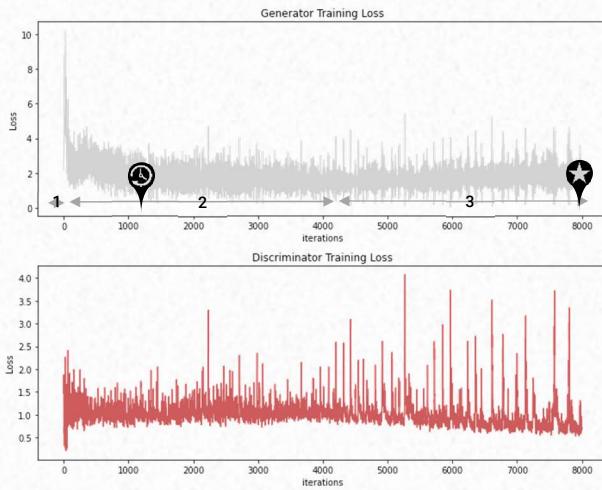


La diversité est subtilement impactée, hasard ou mini-mode-collapse dû à l'accélération de G .

Les images étaient déjà lisibles à 1000 itérations grâce à un contraste initial très fort. Elles se sont ternies à mesure que le réalisme s'améliorait. La colorimétrie a donc comme la forme influé sur les décisions du discriminateur.



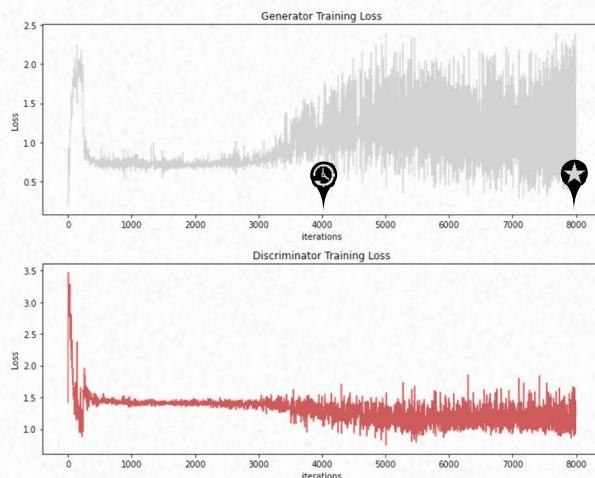
Ensemble 3 – Résultats obtenus avec un pas d'apprentissage plus grand. A gauche, visualisation à 8000 itérations. A droite, un des visages générés, et un échantillon de l'historique à 1200 itérations. Courbes de loss suivante.



A comparer avec la page 2, les loss sont plus stables. Celle de G est aussi plus basse qu'avant, et celle de D plus élevée. Il a apparemment du mal à distinguer les classes. La phase [3] où il domine commence plus tard, à 4000 its. au lieu de 2000 ; le générateur a donc mieux atteint son objectif de tromperie, et cela rejoint l'observation du résultat qui est plus réaliste.

A noter que la phase de perfectionnement de D [1] est quasi invisible du fait de son extrême rapidité. C'est l'impact du paramétrage. Cela peut expliquer les images très contrastées que G a dû créer pour s'en sortir.

(2bis) Cela suffit-il d'augmenter seulement le learning rate de G ? La réponse est non : à 4000 its., les visages générés n'en sont pas. G rend une mauvaise texture, et c'est une illustration de ce qui se passe quand un faussaire doit tromper un critique médiocre, plus lent que lui ; il ne fait pas d'efforts (cf. Q3).

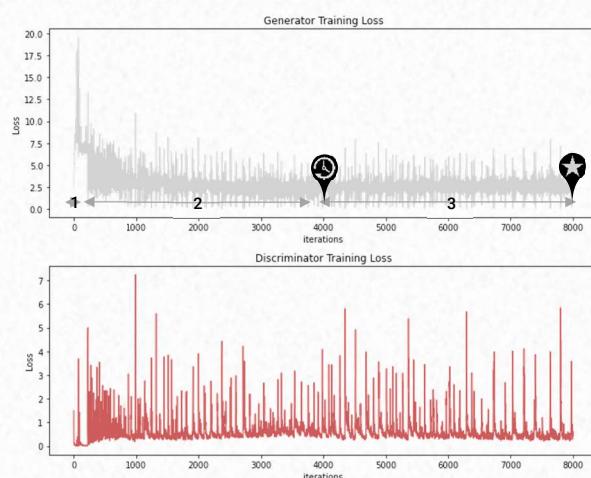


Ensemble 4 – Résultats et loss avec un pas d'apprentissage plus grand pour G seulement. Ci-dessous, visualisation à 8000 itérations, focus sur un visage généré, et historique à 4000 itérations.



Les loss explosent. Même si leurs valeurs finales ressemblent à celles qu'on avait plus haut, l'important est encore une fois leur dynamique, leur évolution : là, on s'éloigne de la convergence.

(2ter) En inversant, c'est mieux. Augmenter le learning rate de D seul force G à se surpasser.



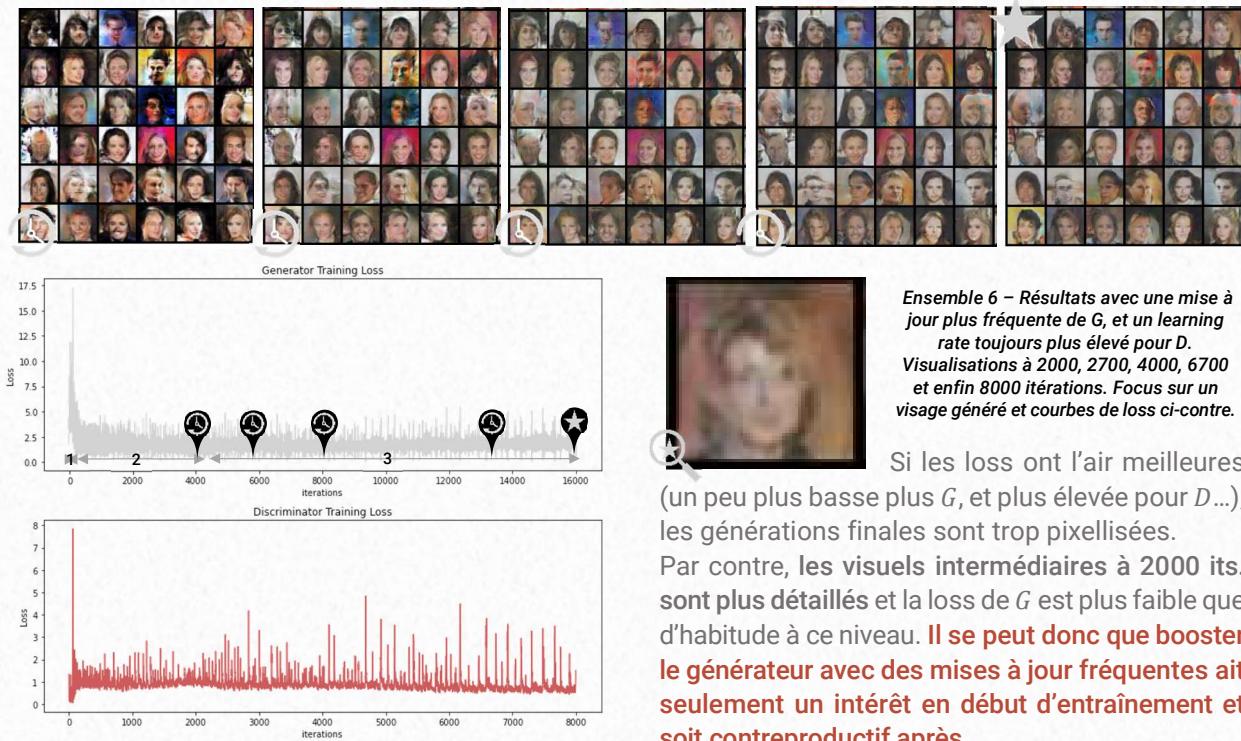
Ensemble 5 – Résultats et loss avec un pas d'apprentissage plus grand pour D seulement. Ci-dessous, visualisation à 8000 itérations, focus sur un visage généré, et historique à 4000 itérations.



La forme des loss ressemble à celle obtenue quand les deux unités étaient boostées ; c'est donc bien la « vivacité » du discriminateur qui stabilise l'apprentissage. Mais la performance de D est trop bonne, ce qui n'est pas ce qu'on souhaite. G gagnait à s'améliorer aussi vite que lui.

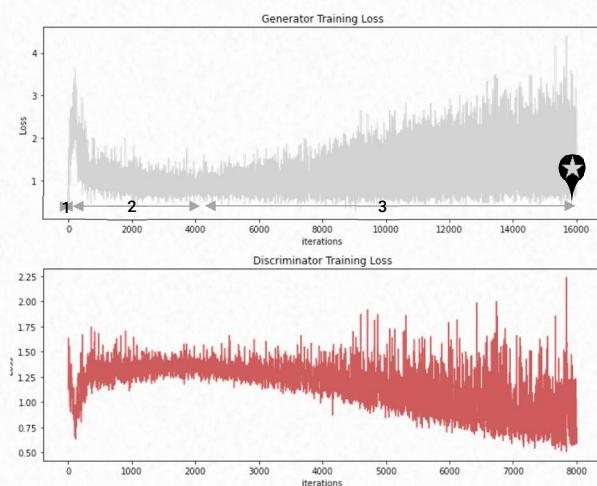
Comparaison entre (2), (2bis) et (2ter) : (2) et (2ter) fonctionnent, meilleur résultat obtenu lorsque D et G sont à l'équilibre en termes de vitesse – avec lr = 0.001, un peu plus que l'hyperparamètre de base.

(3) On s'intéresse aussi au cas où **G** est mis à jour deux fois plus souvent que **D**. C'est une manière de régler l'équilibre dont nous venons de parler. On essaie d'abord de conserver les learning rates différents du test précédent : des mises à jour fréquentes peuvent-elles corriger le désavantage de **G** ?



Si les loss ont l'air meilleures (un peu plus basse pour **G**, et plus élevée pour **D**...), les générations finales sont trop pixellisées.

Par contre, les visuels intermédiaires à 2000 its. sont plus détaillés et la loss de **G** est plus faible que d'habitude à ce niveau. **Il se peut donc que booster le générateur avec des mises à jour fréquentes ait seulement un intérêt en début d'entraînement et soit contreproductif après.**



(3bis) C'est confirmé par une version à learning rates égaux où l'évolution des loss commence bien, puis le discriminateur gagne après 4000 itérations.

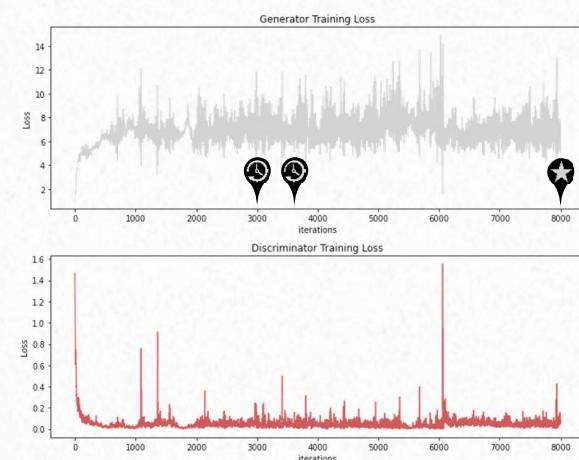


Ensemble 7 – Idem mais à learning rates égaux; visuels à 8000 itérations.

(4) On revient au learning rate initial et à un nombre de mises à jour égal. On peut maintenant penser à augmenter les momentums, avec un résultat décevant : les visages mettent plus de 1000 its à apparaître et il n'y a aucune amélioration sémantique avant la fin.



Ensemble 8 – Résultats avec un momentum classique (0.9) pour **D et **G**. Visualisations à 3000, 3500 et 8000 itérations. Mise en évidence d'oscillations dans l'espace des couleurs de 100 en 100 itérations. Courbes de loss ci-contre.**



Le momentum ajuste la direction du pas de gradient en la pondérant par celle des précédents ; or à cause de cela, la loss de G explose. C'est que G avance à l'aveugle dans son estimation de $P(\text{Data})$. S'il a trop confiance en ses erreurs précédentes, il se perdra dans la mauvaise direction.
 La loss de D par contre est maintenant quasi-nulle en moyenne, le fort momentum l'arrange : il lui permet de trouver très vite la frontière entre réel et fake. (Note : c'est seulement parce que la génération est mauvaise qu'il peut s'en contenter, et quand G est plus « intelligent », c'est un problème.)

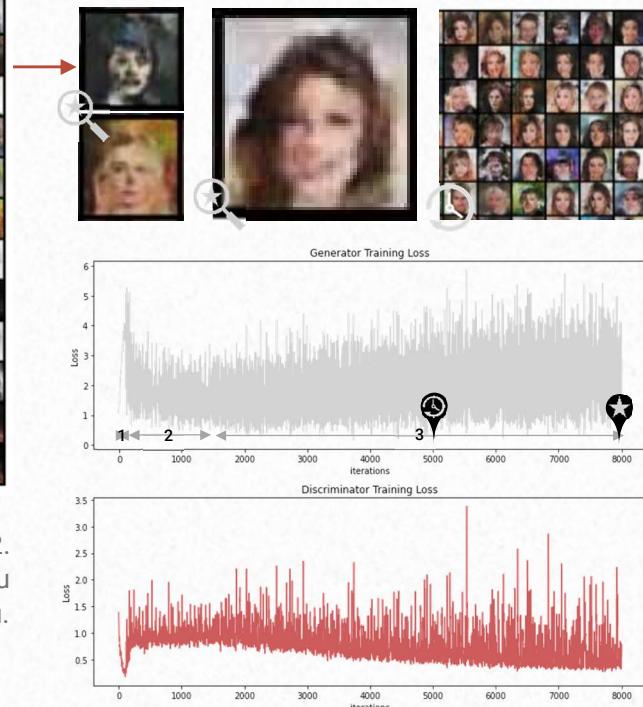
Des paramètres de momentum faibles (comme 0.5) pour D comme G sont donc optimaux pour les GAN.

SET D'EXPERIENCES B → MODIFICATION DES INPUTS

(5) Le DCGAN part d'un vecteur z qui initialise des « composantes » de l'image générée. Comme on se plaint du manque de détail des générations, cela importe-t-il qu'il soit plus petit, plus grand ? Ici, z vaut 10 au lieu de 100 : par rapport à la page 2, cela n'a pas d'impact conséquent sur la génération, ni sur la diversité, ni sur le temps d'apprentissage ; les résultats intermédiaires sont juste un peu meilleurs.

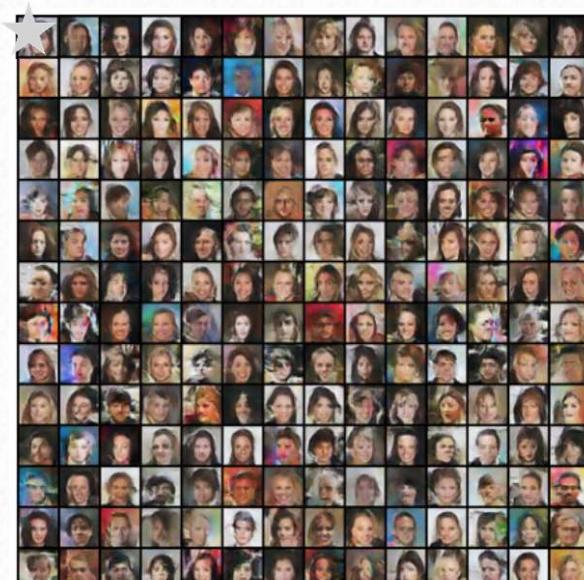


Ensemble 9 – Retour aux hyperparamètres de base, taille de l'input de G à 10 au lieu de 100. Visualisation à 8000 itérations, affichage de visages, échantillon à 5000 itérations. Courbes de loss ci-dessous.

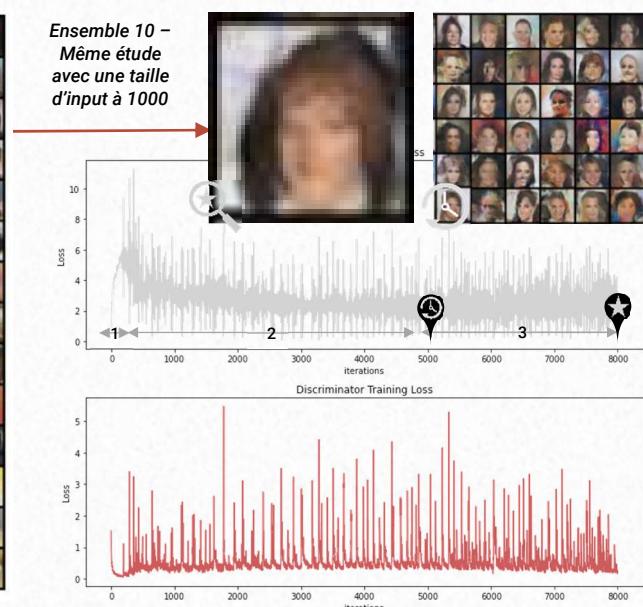


Il y a peu de différence avec les loss de la page 2. L'apprentissage de D au départ est juste un peu plus rapide. On dirait que la taille de z importe peu.

(5bis) Pour un z plus grand, de taille 1000 :



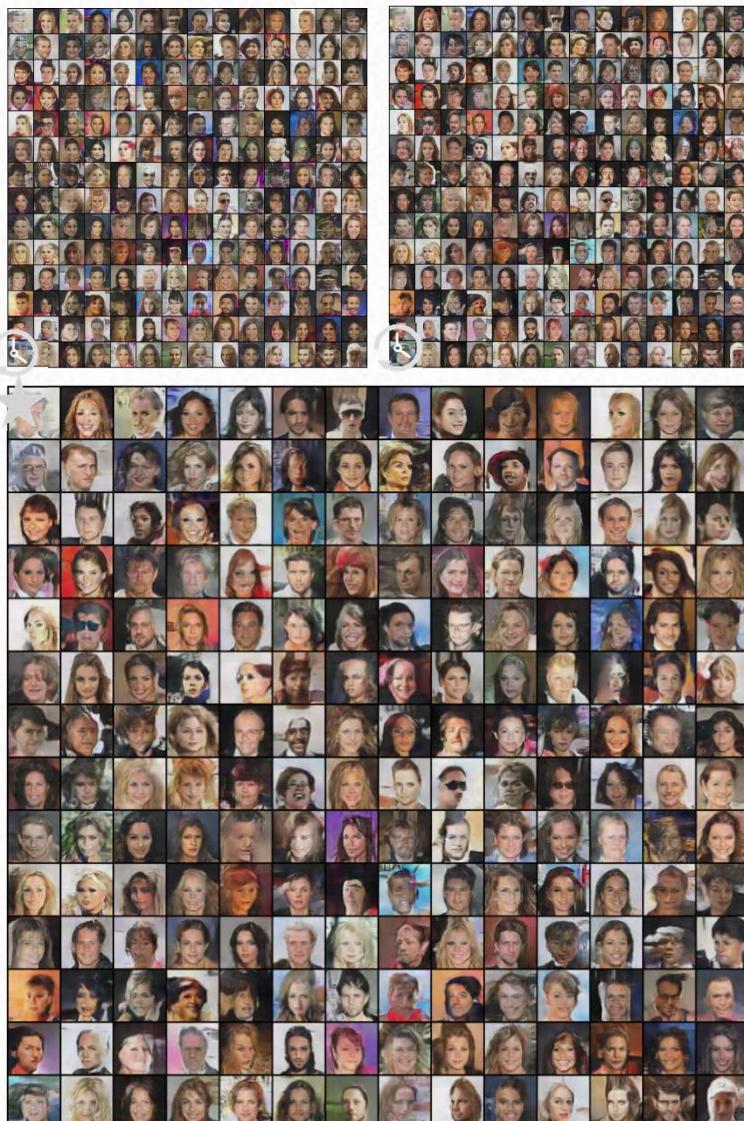
Ensemble 10 – Même étude avec une taille d'input à 1000



Les générations ressemblent encore à celles de la page 2. Le rendu à 5000 et à 8000 itérations diffère (plus lisse, même trop), mais l'effet n'est pas du tout proportionnel à la différence entre les valeurs de nz . De fait, la génération aura la même dimension dans tous les cas, l'impact de la taille de z est dissolu après la première couche de convolution du générateur. Cela explique la différence minime en visualisation.

Les loss par contre n'ont pas le même aspect. On peut penser que le tirage des entrées z est plus uniforme si elles sont de taille 1000 (loi des grands nombres) ; les $G(z)$ le seront donc aussi, et D aura vite une bonne idée de ce que fait G en les observant. D'où sa force, et la stabilité associée qu'on a déjà décrite plus haut.
Mais comme la génération n'y gagne pas, on décide de maintenir un nz intermédiaire.

(6) En gardant les hyperparamètres de départ, on peut aussi modifier les images d'entraînement. On essaie la base CelebA64 qui contient des images de plus haute résolution (les bons GAN de l'état-de-l'art utilisent du 512x512 au moins). Là, tout fonctionne : au même nombre d'epochs, les visuels sont meilleurs, le GAN porte plus d'attention aux détails et cela s'améliore encore avec des itérations supplémentaires. Il y a des visages d'hommes, de femmes, d'apparences et d'âges différents, avec ou sans barbe, lunettes... et de qualité relativement satisfaisante ; mais au prix d'un entraînement beaucoup plus long.



On a systématiquement perdu la loss malgré plusieurs runs à cause des contraintes d'utilisation de Colab.

Ensemble 11 – Générations avec CelebA64, historique à 8000, 16000 puis final à 20000 itérations. Focus sur des visages générés.



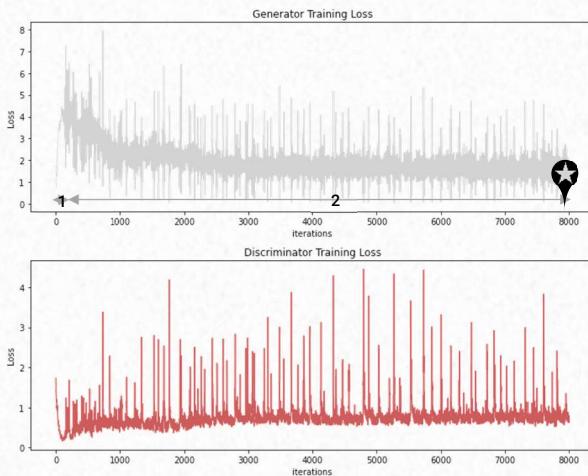
Tous les visages ne sont toujours pas optimaux, mais certains semblent vraiment réalistes (contrairement à ce qu'on avait avant).

C'est avec ce modèle qu'on a essayé une interpolation entre deux inputs, pour comprendre la façon dont les attributs sont codés : on se rend compte qu'on passe d'un visage à un autre en faisant varier plusieurs attributs à la fois. Il n'y a pas de séparation dans l'encodage.

↓ Réaction du GAN à l'interpolation entre deux z



(7) Dernier essai déconnecté des autres, en modifiant la base : on passe à CIFAR qui ne présente pas le même genre de classes. Même si l'échelle est acceptable pour le transfert du modèle (50000 vs. 200000 images), on a là des données très hétéroclites par rapport à CelebA qui ne présente que des portraits.



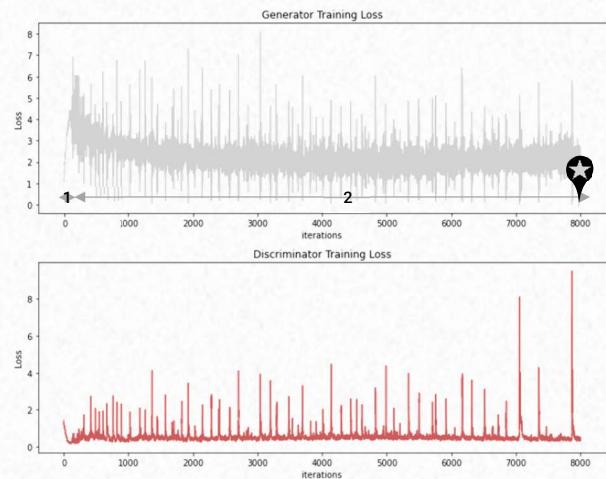
Ensemble 12 – Essai sur CIFAR avec les hyperparamètres de départ

Pour preuve, la génération avec tous les éléments de CIFAR n'a pas de sens. La loss de D est élevée : il y croit, mais les images générées ne floueraient pas un humain. Aucun objet ou animal n'est très reconnaissable parmi chiens, chats, avions, chevaux, bateaux, grenouilles, oiseaux...



(7bis) On a donc choisi une seule classe (*bird*). En restreignant ainsi la base, on sait que le même nombre d'itérations donnera un très grand nombre d'epochs et on s'attend à quelque chose de parfait. Mais les images de train sont disparates en termes de style et de cadrage et le résultat s'en ressent : au final, les faux oiseaux peuvent tromper de loin (couleurs...), mais pas en détail. Côté loss, le graphique ci-contre ressemble au précédent, quoique D se trompe parfois plus lourdement.

Ce cas et le précédent prouvent que compter sur la loss pour statuer sur la qualité de génération n'est pas une bonne stratégie.



Ensemble 13 – Restriction à la classe *bird*, courbes de loss et génération à 8000 itérations ci-contre ; comparaison avec les images de train ci-dessous.





CONCLUSIONS SUR LES DEUX SETS D'EXPERIENCES

Impression générale : ces expériences ont pris du temps (et trop d'espace ici). Elles sont récompensées par l'esthétique unique de certaines générations : une de nos préférées ci-dessus !

Des phénomènes concurrents doivent être pris en compte et il n'est pas facile d'interpréter les résultats.

Loss : elles ne doivent pas être lues par valeurs, on retient que leur situation relative *informe* sur l'état de l'apprentissage (domination de G ou de D) et elles ne reflètent pas toujours la qualité sémantique de la génération.

Paramétrage : les hyperparamètres ont des effets divers,

certains fixes : avec l'optimiseur Adam, les momentums pour D et G doivent absolument être faibles pour pouvoir se diriger vers l'optimum (pp. 5-6).

certains peu impactants : la taille des vecteurs d'input (pp. 6-7) n'a pas une très grande importance.

certains essentiels : trouver l'équilibre D/G (learning rates et mises à jour par itération, pp. 3-5) requiert de faire des essais. **Certaines sources** conseillent d'entraîner D plus souvent que G car une bonne critique force G à s'améliorer. Dans nos expériences, on en avait eu l'intuition quand le learning rate de D était légèrement augmenté.

Dataset d'entraînement : sans conditionnement (voir section suivante), avoir une génération correcte demande des images d'entraînement homogènes (p. 8). Pour améliorer le détail, le mieux est encore d'augmenter leur résolution (p. 7) ; c'est la seule chose qui ait eu un réel impact sur la qualité des sorties.

Principe des GAN conditionnels

Jusqu'ici, nous n'avions aucun moyen de contrôler l'aspect de la génération. Les GAN conditionnels (cGAN) le permettent en prenant une autre entrée en plus du z d'initialisation : un élément y qui décrit les attributs désirés, comme le nom d'une classe connue, ou une image d'un autre domaine... On a alors $\tilde{x} = cG(z, y)$ avec $z \sim P(z)$. Idéalement, à terme, $\tilde{x} \sim P(Data | Data has y)$.

6 Formellement dans le cas cGAN, quel est le problème qu'on cherche à optimiser ? Réécrire les équations (1) et (2) (voir page 1) avec un discriminateur et génératrice conditionnels.

Il suffit d'injecter les y dans les équations : connaissant y , les objectifs de cG et cD deviennent alors

$$\max_{cG} \mathbb{E}_{z \sim P(z)} [\log(cD(cG(z, y), y))] \quad \text{et} \quad \max_{cD} \mathbb{E}_{x^* \in Data} [\log(cD(x^*, y))] + \mathbb{E}_{z \sim P(z)} [\log(1 - cD(cG(z, y), y))]$$

7 A quelles variables la génération ci-dessous pourrait-elle être conditionnée, sachant que ce modèle permet de modifier des images existantes ?

Modifier l'âge (ou le genre apparent) sur un portrait existant requiert de le fournir comme entrée z , puis de faire varier l'attribut y . y peut simplement contenir une valeur flottante indiquant l'âge (où 0 = jeune et 1 = vieux). On précise ici qu'on utilise un **Fader Network** pour avoir les interpolations : on retrouve l'image originale au milieu et les transitions sont possibles dans les deux sens (on augmente ci-dessous la « féminité » ou la « virilité » d'un visage initialement facile à genrer).



8 A quelles variables le génératrice de cette vidéo pourrait-il être conditionné ?

→ Ce génératrice transforme l'hiver en été sur une vidéo existante. Il lui faut donc des frames de la vidéo originale en z (avec un mécanisme de récurrence puisque c'est une séquence temporelle).



→ Pour le conditionnement, il lui faut un y qui précise la saison. Il se peut qu'il soit plus complexe qu'un entier 0 (*winter*) ou 1 (*summer*), que ce soit donc plutôt un vecteur représentant des sous-attributs : *snow* (entre 0 et 1), *clouds*, *leaves*...

→ Pour la cohérence de la représentation séquentielle (comme c'est une vidéo), il se peut qu'un autre y soit donné en complément à chaque pas de temps, à savoir la génération du frame précédent.

9 A quelles variables le génératrice du début de cette vidéo pourrait-il être conditionné ?

→ En entrée z , il est dit que le génératrice prend une carte de *masques* qui correspondent à des positions d'items dans la scène à créer. Il doit les remplir avec des textures appropriées.

→ Les noms de classes associées à chaque masque (*voiture*, *arbre*) peuvent alors être empilées dans un vecteur et servir de y pour choisir celle qui convient à chacun.



→ Comme le tout est très réaliste, on peut faire l'hypothèse qu'il y a aussi une condition globale sur la vraisemblance de la scène (luminosité, colorimétrie), soit un autre y qui peut être donné à des couches plus profondes du génératrice.

→ On donnerait enfin un troisième y , le frame précédent comme ci-dessus, pour la cohérence de la vidéo.

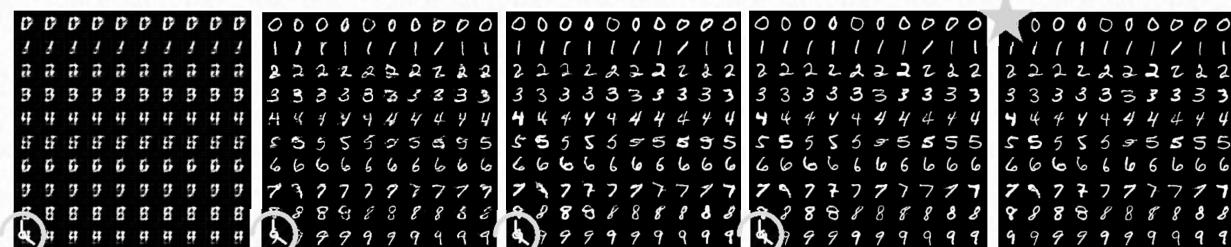
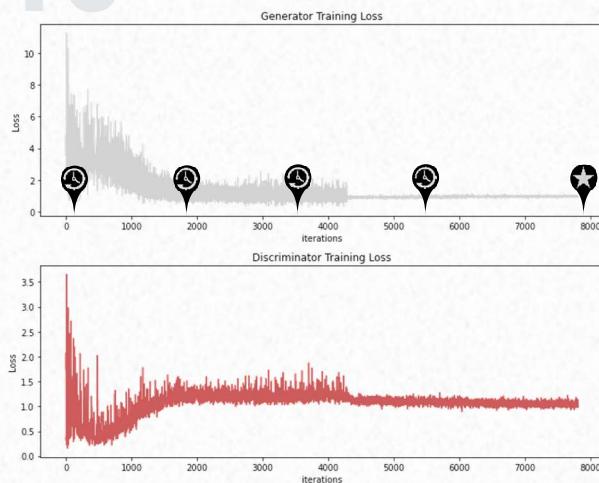
Là, on aura donc trois conditions pour chaque pas de temps de la génération, un vecteur de noms de classes, une constante numérique (ou un tuple) pour l'aspect général, et une image.

Essais avec un cDCGAN

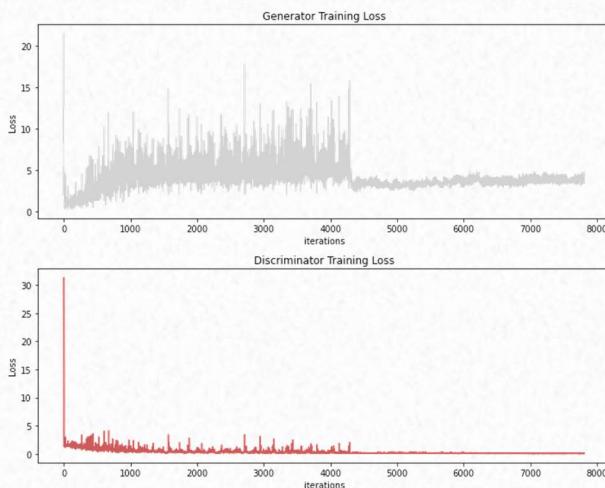
Le cDCGAN (pour Conditional DCGAN) est entraîné sur MNIST. L'attribut y est fourni au générateur et au discriminateur sous forme de vecteur one-hot, il indique la classe du chiffre désiré.

10

Commentez vos expériences avec le cDCGAN.



Ensemble 14 – Génération cDCGAN sur MNIST. Visualisation à 100, 1800, 3500, 5500, puis 8000 itérations. Courbes de loss ci-dessous.



8000 its. correspondent à 20 epochs sur MNIST. En cDCGAN, cela suffit pour une convergence totale des deux loss. Les générations ne varient soudain plus après 4200 itérations, soit un peu plus de dix epochs. Nous avons certes essayé le DCGAN classique sur une base difficile (CelebA), mais il n'a jamais montré une telle stabilité, et ce quelle que soit la durée totale d'entraînement.

100 itérations ont suffi à obtenir des chiffres déjà reconnaissables et correspondant immédiatement à la contrainte y , dans le style de MNIST. Les suivantes raffinent et diversifient la génération.

Par ailleurs, cette expérience peut être répétée avec le même résultat. Le cDCGAN est donc assez stable (à comparer avec cGAN plus loin).

Les hyperparamètres de base suffisaient, mais : **Essai en augmentant le momentum** (non montré ici) – on arrive à la même conclusion que pour DCGAN, la génération est incohérente.

Essai avec un learning rate différent (graphe des loss ci-contre) – l'augmenter est une mauvaise idée, car même si les loss se stabilisent, il n'y a pas convergence dans le bon sens. La loss de G augmente sur la fin et celle de D se rapproche de 0. Les visuels n'étaient pas meilleurs.

11

Pourrait-on enlever y des entrées du discriminateur (c'est-à-dire avoir $cD(x)$ et non $cD(x, y)$) ?

Pour MNIST, y sert à préciser le chiffre à générer. En plus de qualifier la vraisemblance de l'image, le discriminateur doit savoir si le générateur a respecté la consigne. Sans cette entrée, D jugerait seulement du réalisme des générations et rien ne permettrait d'évaluer l'« obéissance » de G .

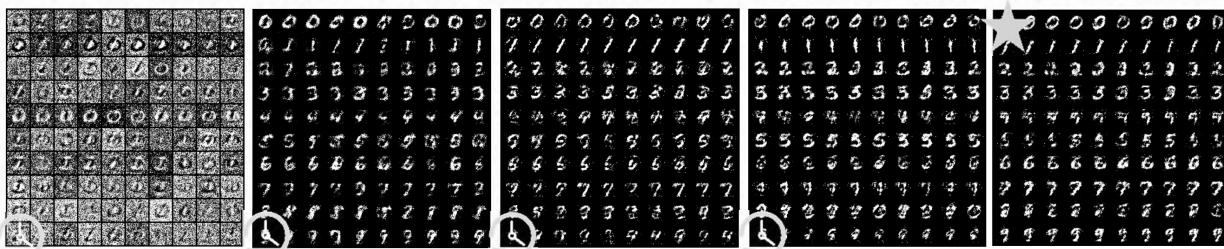
Essais avec un cGAN

Le principe des conditions est conservé, mais les convolutions sont remplacées par des couches FC.

12

Commentez vos expériences avec le cGAN, reportez la meilleure génération de chiffres conditionnés.

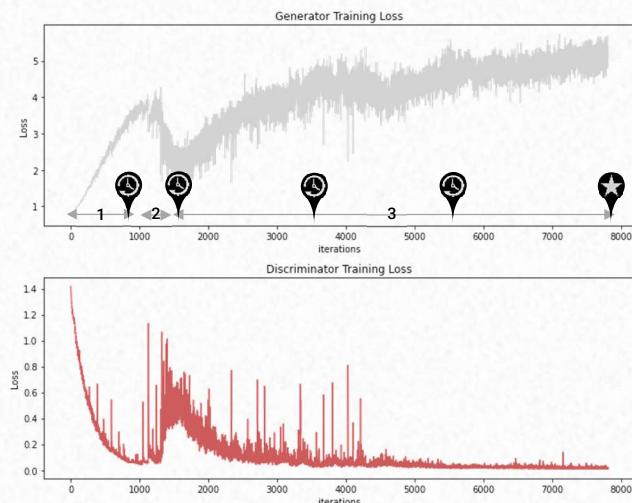
L'absence de convolutions réduit les paramètres (!) et le cGAN est rapide (introduction de cet article). Les seuls hyperparamètres qui aient fonctionné sont ceux par défaut. Là encore, changer le learning rate, le momentum, l'équilibre d'apprentissage n'est pas concluant : on l'a vérifié avec plusieurs runs.



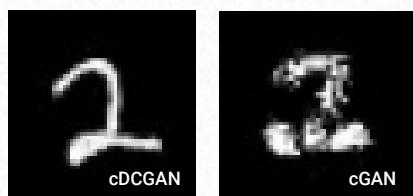
Ensemble 15 – Génération cGAN sur MNIST. Visualisation à 800, 1500, 3500, 5500, puis 8000 itérations. Courbes de loss ci-dessous.

Côté loss, la première phase d'apprentissage (perfectionnement de D) prend beaucoup plus de temps qu'en cDCGAN. Cela se voit sur les visualisations ci-dessus : la forme des chiffres n'apparaît pas avant 1000 itérations. Après cela, la loss de G n'arrive jamais à baisser suffisamment. Elle explose et D gagne après quelques centaines d'itérations. Augmenter le learning rate ne fait qu'accélérer le processus.

Il est important de souligner la **stochasticité des résultats**, relancer le cGAN avec le même paramétrage donne quelque chose de différent. Seule cette stochasticité permet d'obtenir de temps en temps un résultat convaincant.



Dans tous les cas, le rendu a un aspect « nuage de points » avec une grande dispersion ; les chiffres sont composés de points regroupés, juste de plus en plus resserrés selon l'itération. Il n'y a pas de tracé de forme.



De fait, le cGAN génère ses chiffres sous forme d'un vecteur de valeurs indépendantes. Sans convolutions, les pixels de l'image finale n'ont aucun impact direct sur leurs voisins. Cela explique l'apparition parfois de « bruit » transitoire, de points blancs dans l'arrière-plan qui n'appartiennent pas à la forme : ce sont juste des pixels mal évalués isolément.

13

Il est plus difficile de générer des chiffres conditionnés avec un cGAN qu'avec un cDCGAN, pourquoi ?

Supprimer les couches convolutionnelles, c'est perdre la représentation spatiale qui est cruciale dans un cDCGAN. Quand vient l'évaluation, D constate juste l'inadéquation entre les pixels et ne se fait pas piéger. G , lui, aura du mal à s'adapter à cette réponse et à comprendre quels pixels le trahissent. Il ne pourra pas choisir intelligemment ceux qui doivent être mis à jour, et n'est pas forcé de maintenir une cohérence. Or, la mise à jour des poids se fait par région dans un DCGAN, il y a moins de quoi tergiverser.

Conclusion

La conclusion de la Q5 résume ce que nous savons désormais des GAN. La suite montre qu'on peut les conditionner pour diriger la génération vers un mode précis, avec un grand potentiel créatif. Il apparaît dans les dernières expériences que la convolution est essentielle : une fois de plus, on a prouvé que considérer une image région par région permettait d'en avoir la représentation la plus cohérente. Incorporer cette vision aux GAN est un investissement capital pour le réalisme des visuels.