Reproducing the results and analysis

Progressive Growing of GANs was first introduced in [1] by NVIDIA researchers. Using this method, one can generate high-resolution images by starting with low-resolution generators and discriminators and progressively growing higher resolution models. In this document we focus on reproducing the results presented in [1]. To reproduce results, we first relied on the sole paper and then discovered that a closer look at Karras et al. code [2] was also necessary to get the same results.

Loss

Introduction

NVIDIA's method relies on WGANGP [3], and though their implementation is slightly different than others, it is equivalent theoretically.

As a reminder, the WGANGP loss is computed as follow:

$$L_D = \mathbb{E}(D(\hat{x}))_{\hat{x}\in\mathbb{P}_g} - \mathbb{E}(D(x))_{x\in\mathbb{P}_r} + \lambda\mathbb{E}((||\nabla D((1-\alpha)x + \alpha\hat{\hat{x}})|| - 1)^2)_{\alpha\in\mathcal{U}[0,1]}$$

$$L_G = \mathbb{E}(D(\hat{x}))_{\hat{x}\in\mathbb{P}_g}$$

Where α is a value randomly drawn in $[0,1]$ and $\mathbb{P}_r$ and $\mathbb{P}_g$ represent respectively the real and generated data distributions.

A penalty on D is added in order to prevent D's output to diverge:

$$L_\epsilon = \epsilon.\mathbb{E}(D(x)^2)_{x\in real}$$

With $\epsilon = 0.001$.

The parameters of G and D are updated to minimize respectively L_G and $L_D + L_\epsilon$.

Loss profile

In practice, training on the lowest resolutions shows a smooth, easily replicable loss profile, but the model gets more and more unstable as the resolution grows. Figure 1 plots all three losses on celebaHQ.

Layers initialization

In order to ensure that each weight evolves within the same range, NVIDIA describes what it calls the "equalized learning rate". If the weights evolve in various ranges, the correction factors will vary the same way, implying that some weights will have much greater amplitude of change than other.

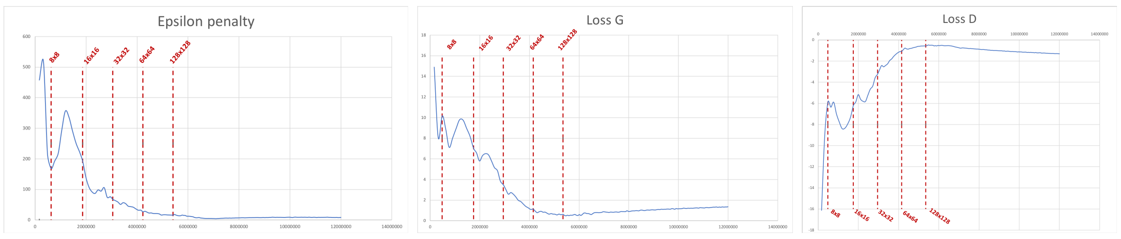

Figure 1: Losses obtained with NVIDIA's code. Dashed vertical lines indicate an increase in resolution. From left to right: $E(D(x)^2)_{x\in real}$, L_G and L_D

If we initialize all weights with a standard normal distribution $\mathcal{N}(0,1)$ and, at runtime compute:

```
x = layer(x) / c
```

Where c is He's [6] constant, instead of initializing all weights with $\mathcal{N}(0,c)$; then all weights should stay within the same range and thus, evolve at the same speed. **This change actually seems to make a big difference in the GAN's output**. Indeed, the model becomes much more stable and can be trained without any artificial clipping or normalization of the gradient.

Probably for the same reason, the He's constant is adjusted for some layers of the generator and the discriminator.

Minibatch size

The default preset in Karras's code uses a minibatch size much larger than what is described in the paper

Resolution	Minibatch size in the paper	Default sizes in the code (1 GPU)
4x4	16	128
8x8	16	128
16x16	16	128
32x32	16	64
64x64	16	32
128x128	16	16
256x256	14	8
512x512	6	4

One of the most common source of failure with GANs is all gradients pointing to the same direction, leading the discriminator or the generator to escalate into a vicious circle. Big minibatches could be a way to avoid that, since the more inputs the system have, the less likely it is to output gradients all pointing in the same direction. However, we noticed that if we forced large minibatches on resolutions higher than 64x64 the model became unstable.

Alpha blending

When a new scale is initialized, Progressive GAN performs a smooth blending between the former low resolution output and the new one. For more details on how this blending work, we invite the reader to refer directly to [1] as this is well explained in the original paper. This stage is important for the model convergence and cannot be skipped. Besides, although this is not mentioned in the original paper, one can notice in Kerras's code that input images from the real dataset also undergo the low resolution blending during the transition phase. In other words, if α is the value of the blending factor at current stage of the transition, then any input real image I will be transformed as follows:

$$I = (1 - \alpha)I + \mathcal{L}(I)$$

Where $\mathcal{L}$ is the low-pass filter corresponding to a down-sampling operation followed by an up-sampling one. This smooth transition on the real images seems to have a big impact on the model's stability.

A few additional details about the transition phase are necessary to achieve good performance. Indeed, the paper mentions that the transition phase lasts for about 600k images per scale but doesn't detail how big each step of the blending ratio should be. By looking at the code, we notice that this ratio is updated linearly by 1 / 600 every 1000 images. Changes too harsh of the blending ratio often result in the model diverging.

CelebA and CelebA-HQ Dataset

CelebA dataset contains approximately 200k pictures of celebrities of various resolutions. When talking about celeba we usually consider the "align" version where all images are centered around the character's face and are kept to a 178×218 resolution. In NVIDIA's work are considered pictures of the "align" dataset tightly cropped around the subject face to fit a 128×128 resolution. This cropping seems to be a common practice with celebA. Besides, it reduces the diversity of the input dataset and makes the GAN training easier.

Celeba-HQ contains 30k images of size 1024×1024 centered around the subject's face.

Overflow barrier

Karras added an "overflow wall" in his code: when the gradient overflows, the weight are not updated. This barrier is barely used with default settings (big minibatch), but it could be interesting to see if overflowing gradients happen more often with a small minibatch.

Stability

With all above "tricks" set the model is stable and reach smoothly the 1024x1024 resolution for any dataset without mode collapse. However, missing some of these "tricks" and especially the layer initialization makes it much less reliable.

Running average

As underlined in the paper, as far as visualization is concerned, we consider the results of the smoothed generator. At the n-th iteration, G^n_{smooth} is defined as follow:

$$G^n_{smooth} = 0.999 * G^{n-1}_{smooth} + 0.001 * G^n$$

Network evaluation

Sliced Wasserstein Distance

The paper uses to the Sliced Wasserstein distance (SWD) [4] to evaluate their results. They actually implements a different method, simpler and faster than the gradient descent described in [4].

```
**Data:** Input point cloud $X_i$, target point cloud $Y_i$ in $\mathbb{R}^n$
**Result:** Approximation $\sigma*$ of the optimal permutation minimizing
        $\sum_i ||X_{\sigma*(i)} - Y_i||$
initialization;
while Not converging do
    Choose $K$ random directions $\theta_k \in \mathbb{R}^n$;
    Initialize the gradient $\nabla X_i : l = 0$;
    for each $\theta_\mathbf{k}$ do
        Compute the projections $(x_i^k, y_i^k)$ of $X_i$ and $Y_i$ on $\theta_k$;
        Sort both distribution independently, and store the corresponding
          permutations $(\sigma_X, \sigma_Y)$;
        Compute $\sigma = \sigma_X \circ \sigma_Y^{-1}$;
        Update $\nabla X_i := \nabla X_i + \theta_\mathbf{k}(x^k_{\sigma(i)} - y_i^k)$;
    end
    Update $X := X + \mu \nabla X_i$;
end
```

Algorithm 1: SWD method as described in [4]

This algorithm usually converges to a point cloud $X*$ with a unique permutation $\theta*$. This permutation can then be used for the computation of the approximate Wasserstein distance.

Karras' version of the SWD distance is expressed as follows:

```
Data: Input point cloud $X_i$, target point cloud $Y_i$ in $\mathbb{R}^n$
Result: Approximation of the SWD distance
Initialize the distance $D = 0$;
; for $r$ in $N_{repeats}$ do
    Choose $K$ random directions $\theta_k \in \mathbb{R}^n$;
    for each $\theta_\mathbf{k}$ do
        Compute the projections $(x_i^k, y_i^k)$ of $X_i$ and $Y_i$ on $\theta_k$;
        Sort both distribution independently, and store the corresponding
         permutations $(\sigma_X, \sigma_Y)$;
        Compute $\sigma = \sigma_X \circ \sigma_Y^{-1}$;
        Update $D := D + \sum_i |x_{\sigma_X(i)} - y_{\sigma_Y(i)}|$;
    end
end
$D := \frac{1}{KN_{repeats}} D$;
```

Algorithm 2: SWD method used in Karras's code

This metric measures the similarity of the generated distribution with the training distribution: it's a fidelity metric. A low SWD distance means that randomly sampled patches from the training dataset on the one hand, and from fake generations on the other hand have a similar distribution.

Furthermore, in order to have a proper reference value for this metrics, Karras also uses it to compare two set of patches randomly extracted from the original dataset. We obtain the typical "noise" or "variation" of the dataset. Theses values are referenced in Table 1.

Celeba	128x128	64x64	32x32	16x16	Average
Internal Noise	1.5656	1.5661	1.4281	3.9843	2.1360
SWD score with NVIDIA's code	2.9429	2.5752	2.4550	7.3625	3.8339

Table 1: SWD score on the celeba dataset $\times 10^3$

Multi-scale structural similarity

Another metric is employed by NVIDIA to assess the quality of their method Wang & Co's MS-SSIM measure [5]. The MS-SSIM similarity score between two images X and Y is a real in $[0; 1]$ number asserting the similarity of:

- the global luminance value of each image represented by their mean values μ_X and μ_Y
- the global contrast value of each image represented by their variance σ_X and σ_Y
- the pixelwise covariance between X and Y σ_{XY}

A MS-SSIM score equal to one indicates that X and Y are the same images, on the contrary a score close to 0 implies that X and Y are extremely dissimilar. This value is computed for several scales of the input.

In the case of GANs, this score is used to estimate the variability of the output: for each couple of images in the dataset, their MS-SSIM score is computed; the final score of the dataset is the averages of the scores computed on pairs.

References

[1] T. Karras, T. Aila, S. Laine & J. Lehtimen 2017. Progressive Growing of GANs for Improved Quality, Stability, and Variation

[2] T. Karras : https://github.com/tkarras/progressive_growing_of_gans

[3] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. Courville, Improved Training of Wasserstein GANs

[4] R. Julien, G. Peyre, J. Delon, B. Marc, Wasserstein Barycenter and its Application to Texture Mixing

[5] Z. Wang, E. Simoncelli, A. Bovik, Multi-scale structural similarity for Image Quality Assessment.

[6] K. He, X. Zhang, S. Ren and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.