A novel crossover for evolutionary GAN

Michele Alessi

May 2024

Abstract

GANs models [1] have recently proven to be among the best generative models, reaching the state of the art in several fields.
Despite this, as highlighted by numerous studies [2], [3], [4], [5], they suffer from several problems including non-convergence, mode collapse (the generator produces a low sampling variety), and diminished gradient (the discriminator gets too successful that the generator gradient vanishes and learns nothing).
Several approaches have been attempted to overcome these problems, including the use of genetic algorithms to improve model performance [6], [7], [8], [9].
In this work, we will define a new crossover operator acting on a population of generators, and we will qualitatively show how this simple operation contributes to obtaining better results, improving generalization, and counteracting the mode collapse problem.

1 Background

Generative Adversarial Networks (GANs) [1] is a machine learning framework introduced by Ian Goodfellow et al. in 2014. Given a training set, GAN aims to learn the probability distribution of the training set itself, with the ultimate goal of generating new samples. In the GAN framework, there are two primary components: the *generator* and the *discriminator*. These components are trained simultaneously through a min-max game setting.

Generator The generator takes random noise as input and generates samples: it learns to transform input noise into data samples that are indistinguishable from real data.
Mathematically, the generator function is represented as $G(z)$, where z is a random noise vector sampled from a prior distribution (often Gaussian or Uniform).

Discriminator The discriminator acts as a binary classifier, aiming to distinguish between real data samples and fake samples generated by the generator: it is trained to assign high probabilities to real data and low probabilities to fake data.
Mathematically, the discriminator function is represented as $D(x)$, where x is a data sample, and $D(x)$ represents the probability that x comes from the real data distribution.

Training Training a GAN model involves two steps, both performed at the batch-processing level.
Given a batch of real data samples, the discriminator is trained to maximize the probability of assigning correct labels (1 for real and 0 for fake) to these samples. Simultaneously, the discriminator is trained to minimize the probability of assigning correct labels to *fake* samples generated by the generator.
The generator is trained to generate samples that are likely to be classified as real by the discriminator (actually, the generator tries to *fool* the discriminator). In other words, it aims to minimize the probability of the discriminator correctly classifying generated samples as fake. This can be formalized as a min-max problem as follows:

$$\min_G \max_D \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))] \quad (1)$$

where the first term corresponds to real data samples and the second term corresponds to fake samples generated by the generator network.

2 Prior Works on Evolutionary GAN

GAN has shown to be a very powerful framework in generative AI, but it suffers from training problems such as instability and mode collapse. Evolutionary GAN ([6], [7], [8], [9]) was then introduced as a possible solution for stable GAN training and improved generative performance. Differently from standard GANs, Evolutionary GANs utilize different adversarial training objectives as mutation operations and evolve a population of generators. An evaluation mechanism is implemented to measure the quality and diversity of generated samples (based on the gradient of the discriminator D), such that only well-performing generators are preserved through iterations. In this section we focus on the work of Wang et al. [6] since their proposed method deals only with mutation operators. This will allow us to easily add our crossover operator to their method. Other work ([7], [8], [9]) has been done using different crossover operators, and the following sections will highlight the main differences, along with the pros and cons.

Evolutionary GAN The main idea is to keep a population $\mathcal{P}$ of generators and inside each epoch, an evolutionary step is implemented as follows:

1. for each generator $g \in \mathcal{P}$, for each possible mutation:
   (a) update the weights of g through backpropagation
   (b) compute the fitness $\mathcal{F}$ of g
2. sort the population wrt the fitness
3. build the next population selecting the best generators.

Mutations In this framework, the mutations are introduced as different loss functions used for updating the generator parameters through backpropagation. In particular, three different mutations were defined:

$$\mathcal{M}^{minmax} = \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))] \quad (2)$$

$$\mathcal{M}^{heuristic} = -\mathbb{E}_{z\sim p(z)}[\log D(G(z))] \quad (3)$$

$$\mathcal{M}^{meanquare} = \mathbb{E}_{z\sim p(z)}[(D(G(z)) - 1)^2]. \quad (4)$$

Fitness The fitness of an individual is computed by combining two factors: the quality fitness score $\mathcal{F}_q$ and the diversity fitness score $\mathcal{F}_d$.

Algorithm 1 E-GAN. Default values $\alpha = 0.0002$, $\beta_1 = 0.5$, $\beta_2 = 0.99$, $n_D = 2$, $n_p = 1$, $n_m = 3$, $m = 16$.

Require: batch size m. D's updating steps per iteration n_D. Number of parents n_p. Number of mutations n_m. Adam hyper-parameters α, β_1, β_2, the hyper-parameter γ of $\mathcal{F}$.
Require: Initial D's parameters w_0. and G's parameters $\{\theta_0^1, \theta_0^2, \dots, \theta_0^{n_p}\}$.
for number of training iterations **do**
  for $k = 0, ..., n_D$ **do**
    Sample a batch of $\{x^{(i)}\}_{i=1}^m \sim p_{\text{data}}$ (training data), and a batch of $\{z^{(i)}\}_{i=1}^m \sim p_z$ (noise samples).
    $g_w \leftarrow \nabla_w[\frac{1}{m}\sum_{i=1}^m \log D_w(x^{(i)})$
      $+\frac{1}{m}\sum_{j=1}^{n_p}\sum_{i=1}^{m/n_p}\log\big(1 - D_w(G_{\theta^j}(z^{(i)}))\big)]$
    $w \leftarrow \text{Adam}(g_w, w, \alpha, \beta_1, \beta_2)$
  end for
  for $j = 0, ..., n_p$ **do**
    for $h = 0, ..., n_m$ **do**
      Sample a batch of $\{z^{(i)}\}_{i=1}^m \sim p_z$
      $g_{\theta^{j,h}} \leftarrow \nabla_{\theta^j}\mathcal{M}_G^h(\{z^{(i)}\}_{i=1}^m, \theta^j)$
      $\theta^{j,h}_{\text{child}} \leftarrow \text{Adam}(g_{\theta^{j,h}}, \theta^j, \alpha, \beta_1, \beta_2)$
      $\mathcal{F}^{j,h} \leftarrow \mathcal{F}_q^{j,h} + \gamma\mathcal{F}_d^{j,h}$
    end for
  end for
  $\{\mathcal{F}^{j_1,h_1}, \mathcal{F}^{j_2,h_2}, \dots\} \leftarrow \text{sort}(\{\mathcal{F}^{j,h}\})$
  $\theta^1, \theta^2, \dots, \theta^{n_p} \leftarrow \theta^{j_1,h_1}_{\text{child}}, \theta^{j_2,h_2}_{\text{child}}, \dots, \theta^{j_{n_p},h_{n_p}}_{\text{child}}$
end for

Those are combined in the final fitness

$$\mathcal{F} = \mathcal{F}_q + \gamma\mathcal{F}_d \quad (5)$$

where γ is a hyperparameter.
The quality fitness score $\mathcal{F}_q$ tells us how good a generator is in fooling the discriminator, and it is defined as:

$$\mathcal{F}_q = \mathbb{E}_{z\sim p(z)}[D(G(z))]. \quad (6)$$

The diversity fitness score $\mathcal{F}_d$ is used to promote the diversity of generated samples:

$$\mathcal{F}_d = -\log||\nabla D - \mathbb{E}_{x\sim p(x)}[\log D(x)] - \mathbb{E}_{z\sim p(z)}[(1 - D(G(z)))]. \quad (7)$$

If $\mathcal{F}_d$ is relatively high, the terms inside the logarithm have to be small: in particular, ∇D has to be small (intuitively a shallow slope), leading to much more flexibility in classifying the fake samples, and so more diversity for the generator. The complete algorithm is described in 1.

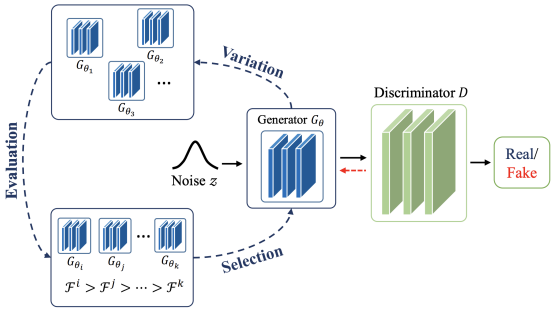

Figure 1: Evolutionary GAN workflow.

3 A New Crossover

We implemented a new type of crossover, running on-fly during the training. The idea is to make generators interact with each other in the following way:

1. Let $g \in \mathcal{P}$ be a generator inside the current population. As we have seen before, to perform backpropagation we need to generate some fake samples with g, feed the discriminator with those fake samples, compute the loss, and update the weights of g.
2. Hence, we define a crossover operator by letting those fake samples be generated by a randomly chosen generator $g' \in P$, with probability p_{cross}.
3. This happens just before the backpropagation step, and of course, it affects only the generation phase (*ie* g' is used to generate fake samples, then those samples are used to compute the loss, and the loss is used to update the weights of g through backpropagation).

The algortihm is described in 2.

PROs Note that this type of *indirect crossover* is very efficient in the sense that it runs *on-fly* during the training. In other words, it does not require further computations but simply uses the weights coming from another generator of the population. Consequently, we are able to implement this operator without minimally affecting the efficiency of the basic algorithm, while still managing to define a more general framework.

Algorithm 2 Crossover

for $G \in \mathcal{P}$ **do**
  $j \leftarrow \mathcal{U}([0, 1])$
  if $j > p_{cross}$ **then**
    // no crossover performed
    $\varepsilon \leftarrow \mathcal{U}([-1, 1])$
    Fake $\leftarrow G(\varepsilon)$
    Output $\leftarrow D$(Fake)
  else
    // get a random generator from $\mathcal{P}$ to update parameters of the current generator
    $J \leftarrow \mathcal{U}(0, \ldots, n_p)$
    $G = \mathcal{P}[J]$
    $\varepsilon \leftarrow \mathcal{U}([-1, 1])$
    Fake $\leftarrow G(\varepsilon)$
    Output $\leftarrow D$(Fake)
  end if
end for

4 Preliminary Results

In this section, we provide some preliminary results, comparing them with the results of [6]. Currently, we have only managed to partially train the model: [6] trained their model on Nvidia GTX 1080Ti GPUs, requiring approximately 30 hours on a single GPU to train a model for 64 $\times$ 64 images using the DCGAN architecture. In contrast, we trained our version on a MacBook with an Apple M1 Pro chip, taking just 5 hours of training.

As can be seen from Figs. 2-5, the results appear to be promising. In particular, results on the interpolation task suggest good generalization performances of this method with respect to the baseline.

Figure 2: Generation: [6].

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua

Figure 3: Generation: Ours.

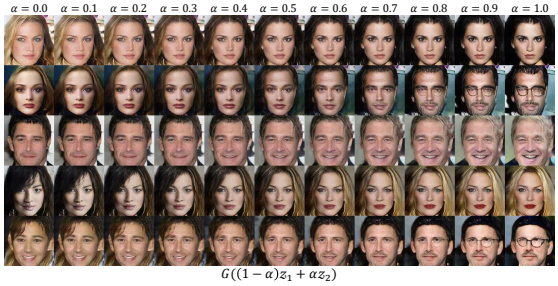

Figure 4: Latent interpolation: [6].

Bengio. Generative adversarial networks, 2014.

[2] Divya Saxena and Jiannong Cao. Generative adversarial networks (gans survey): Challenges, solutions, and future directions, 2023.

[3] Hoang Thanh-Tung and Truyen Tran. Catastrophic forgetting and mode collapse in gans. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2020.

[4] Wei Li, Li Fan, Zhenyu Wang, Chao Ma, and Xiaohui Cui. Tackling mode collapse in multi-generator gans with orthogonal vectors. *Pattern Recognition*, 110:107646, 2021.

[5] Samuel A. Barnett. Convergence problems with generative adversarial networks (gans), 2018.

[6] Chaoyue Wang, Chang Xu, Xin Yao, and Dacheng Tao. Evolutionary generative adversarial networks, 2018.

[7] Junjie Li, Junwei Zhang, Xiaoyu Gong, and Shuai Lü. Evolutionary generative adversarial networks with crossover based knowledge distillation, 2021.

Figure 5: Latent interpolation: Ours.

[8] Junjie Li, Jingyao Li, Wenbo Zhou, and Shuai Lü. Ie-gan: An improved evolutionary generative adversarial network using a new fitness function and a generic crossover operator, 2022.

[9] Guohao Ying, Xin He, Bin Gao, Bo Han, and Xiaowen Chu. Eagan: Efficient two-stage evolutionary architecture search for gans, 2022.