Multi-Optimiser Training for GANs based on Evolutionary Computation

Yixia Zhang
School of Software
Nanjing University of
Information Science and Technology
Nanjing, China

Yu Xue
School of Software
Nanjing University of
Information Science and Technology
Nanjing, China

Ferrante Neri
School of Computer Science
and Electronic Engineering
University of Surrey
Guildford, United Kingdom
f.neri@surrey.ac.uk

***Abstract*—Generative adversarial networks (GANs) are widely recognized for their impressive ability to generate realistic data. Despite the popularity of GANs, training them poses challenges such as mode collapse and instability. To address these issues, many variants enhance GAN performance through improvements in network architecture, modifications to loss functions, and the inclusion of regularization techniques. However, a limited number of methods focuses on optimising GAN performance from the optimiser's perspective, despite the distinct roles different optimisers play in training. Existing GANs typically employ a single optimiser throughout, with Adam being the default choice for GAN training. Approaches using multiple optimisers have shown improved performance but are often task-specific.**

This paper introduces a novel, portable approach that leverages the strengths of multiple training methods, with an evolutionary supervisor coordinating the training of different sections of the network. Initially, real-number-encoded vectors representing the optimiser for each sub-parameter layer undergo progressive enhancement using a standard evolutionary algorithm (SGA). Through the evolutionary process, optimal optimiser combinations are retained based on the performance of the trained GAN. The proposed method, SGA-GANs, is validated on the CIFAR10 dataset by integrating the steps into five benchmark GAN models: GAN, DCGAN, BEGAN, WGAN, and WGAN-GP. Experimental results demonstrate that SGA-GANs outperforms single optimiser training methods, achieving superior evaluation results and generating higher-quality images. Source code can be found at https://github.com/lizzhang-spec/SGA-GANs.

***Index Terms*—large-scale optimisation, evolutionary algorithms, multi-optimiser training, generative adversarial networks**

I. Introduction

Generative Adversarial Networks (GANs) are a class of artificial intelligence algorithms used in machine learning for generating new data instances that resemble a given dataset, and it has been the subject of numerous studies in the fields of deep learning and generative modeling. Since GAN [1] was first introduced, mode collapse (where the generator network fails to capture the full diversity of the data distribution it is trained on, resulting in a limited set of similar or nearly identical generated samples) and training instability have been problems with GANs. In response to these problems, researchers have continued to develop various techniques and variants to optimise the training of GANs on different aspects of different tasks. Nowadays, GANs have been applied in various fields, including image synthesis [2], style transfer [3], data augmentation [4], super-resolution [5], image-to-image translation [6], and more [7]. Taking an overview of the various variants of GANs, most of the existing methods train and optimise the models from three perspectives, one is by architectural improvements (*e.g.*, increasing the depth of the neural network in the generator and discriminator, introducing residual connections) [8], [9], the second is to modify the loss function [10]–[12], and the third is to use regularisation and normalisation techniques to train the GANs [13], [14]. The variants of GANs based on these three aspects have been consistently improving their model training stability and evaluation performance. However, we find that the vast majority of GANs methods do not conduct further research on the optimiser used for the training process, and they commonly use a single optimiser throughout the GANs training process, with most of them using Adam [15], SGD [16] and RMSprop [17]. In most cases, only one optimiser is used for training, even though using a combination of multiple optimisers could be beneficial in achieving higher performance [18].

However, the training of a GAN can be considered a large-scale complex optimisation problem. We know that domain decomposition, *i.e.*, the separate optimisation of groups of variables, is one popular approach to quickly detect improved solutions, as seen in [19], [20]. This approach is justified by the study in [21], which demonstrates that experimental conditions in large-scale domains make problems operationally separable irrespective of their actual complexity. Furthermore, some studies show how multiple optimisers may concurrently tackle large-scale optimisation problems [22]. Hence, in this study, we aim to investigate the efficacy of decomposing the intricate parameter optimisation task of a GAN into multiple subspaces for optimisation processes. This strategy would allow each subspace to harness the unique advantages of various optimisers, potentially enhancing the overall model performance. To leverage the potential of various optimisers effectively, it is essential to employ a searching technique that assists each subspace in identifying the most suitable optimiser for its training.

Based on the studies on domain decomposition for large scale optimisation [23], this paper proposes a multi-optimiser training method for GANs based on evolutionary computation, aiming to allow each sub-parameter layer under the large

2024 IEEE Congress on Evolutionary Computation (CEC) | 979-8-3503-0836-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/CEC60901.2024.10612002

parameter space of GANs to be trained using the optimal optimiser. First, we divide the parameters of the generator and discriminator networks into a number of sub-parameter layers, each of which can be a fully connected layer, a convolution layer, or a ResNet layer. Second, we use real number encoding vectors to select the optimiser trained on each divided sub-parameter layer; then an evolutionary algorithm is applied to find the most suitable optimiser for each sub-parameter layer.

The proposed method, termed as standard genetic algorithm - generative adversarial network (SGA-GANs) comprises the following contributions:

- A new training method for GANs based on multiple optimisers coordinated by an evolutionary supervisor and performing a domain decomposition of the training space.
- The proposed SGA module shows good portability and improved model performance on multiple baseline GAN models.

The proposed approach makes parameter training of GANs more stable and results in higher quality of the generated images.

The remainder of this article is organised in the following way. Section II provides related works in GAN training. Section III describes the proposed method in detail. Section IV presents the results of this study. Finally, Section V provides the concluding remarks to this article.

II. Related Work: GAN Training

Optimisers play a crucial role in the training of neural networks. The choice of the optimiser can significantly impact the stability, convergence, and performance of GANs. Over the past few years, several studies have been carried out to improve the performance of GANs by adapting different optimisers.

In the initial stages of GAN [1] research, stochastic gradient descent (SGD) [16] was commonly used as the optimiser for both the generator and discriminator; however, it often led to training instability [24]. To improve the training stability, researchers began to explore the use of momentum-based optimisers such as nesterov accelerated gradient (NAG) [25] and Adam [15] in GANs training. These optimisers improved training stability by reducing oscillations and speeding up convergence. Some researchers subsequently brought a shift in optimisation by introducing new adversarial loss. WGAN [26] used a different loss function called the Wasserstein loss (also known as Earth Mover's Distance) and relied on gradient clipping to maintain the Lipschitz continuity of the discriminator. This change in the loss function impacted the choice of the optimiser and required special handling. As more and more research on GANs was proposed, the Adam optimiser gained popularity in GANs training due to its adaptive learning rates and momentum properties, which were well-suited for handling the complex optimisation landscape of GANs. Many GAN variants started to use Adam for both the generator and discriminator. At the same time, RMSProp [17], an adaptive gradient-based optimiser, was also widely used in GANs training. Over time, researchers have developed more advanced optimisers and GAN-specific optimisers tailored to address the unique challenges of GAN training. Some of these include AdaMax [15], AMSGrad [27], and L-BFGS [28]. Liang *et al* [29] proposed to train GAN with AdaMax to reduce overfitting in unsupervised time series anomaly detection. Kim *et al* [30] noticed that when batch sizes become smaller the convergence issue occurs even when the normalization technique has been used, so they adopted AMSGrad to stabilize the step size. Liu *et al* [31] applied L-BFGS optimiser for GAN compression work. Recent approaches have involved hybrid training strategies, such as using a combination of optimisers for the generator and discriminator. These strategies can help achieve better convergence and stability. In [32], authors proposed to train GAN using a mixture of SGD and LSVGD aiming to make a balance between precision and diversity of generated images. In addition, AdaBelief [33] was proposed to simultaneously achieve fast convergence as in adaptive methods (*e.g.*, Adam), good generalisation as in SGD, and training stability.

III. Methodology: SGA-GANs

This section describes in detail the various components of our proposed SGA-GANs. Section III-A describes the encoding and initialisation of the large parameter space of GANs, which is a prerequisite for the evolutionary optimisation process of GANs in Section III-B. The evolutionary optimisation process focuses on the selection of the best optimiser for each sub-parameter layer using the evolutionary computation technique SGA.

A. Encoding and Initialisation of GANs

Given a network of GANs, we first divide its entire parameter space into multiple sub-parameter layers. For the vast majority of GANs network designs, neural network architectures such as fully connected layer, convolution layer, ResNet layer, *etc.* are the main components of GANs, and these neural network architectures excel at capturing features. Furthermore, activation functions such as Sigmoid, ReLU, *etc.* are also important components of GANs network design, as they help mitigate the vanishing gradient problem, making the GAN easier to train. From the above observations, we divide the given GANs network according to the location of neural network architectures,*i.e.*, each neural network structure in the given large parameter space of the GANs is regarded as a sub-parameter layer that contains the activation function immediately following it. With this division, we can divide the GAN training problem and apply the following approach.

Considering that we cannot accurately estimate the training effect of different optimisers on different sub-parameter layers, we apply an evolutionary algorithm to encode the optimisers chosen by the sub-parameter layers as real-number codes. In practice, the task of finding the most suitable optimiser for each neural network layer is regarded as a real-number programming problem and the optimiser chosen for each layer of the network is correspondingly represented as a real-number array. Wherein, each sub-array of the real-number array is assigned to a sub-parameter layer in the given network, *i.e.*,

$$n_i = [n_1, n_2, ..., n_j], n_j = rand(-1.0, 1.0) \quad (1)$$

where n_i denotes the state of sub-array for the i-th sub-parameter layer in the given network, n_j is the state of the sub-array, returned by $rand$ as a random value in the range of the given float real numbers, *i.e.*, $(-1.0, 1.0)$. To initialise the optimiser used by each sub-parameter layer respective to the training process, a pre-defined optimiser array **Optims** must be given by

$$\mathbf{Optims} = [optim_1, optim_2, ..., optim_j]. \quad (2)$$

It is worth noting that the length of the sub-array in Eq. (1) and the length of the pre-defined optimiser array **Optims** in Eq. (2) are both j.

After the initialisation of the real-number array, we obtain the maximum value index of each sub-array in the real-number array, and we subsequently select an optimiser for each sub-parameter layer based on the maximum value indexes, where each maximum value index corresponds to the pre-defined optimiser array **Optims**, *i.e.*,

$$selected\_optim_i = \mathbf{Optims}[argmax(n_i)] \quad (3)$$

where $selected\_optimiser_i$ is the optimiser chosen by the i-th sub-parameter layer in the given network, $argmax$ returns the index of the maximum value in the array n_i (Eq. (1)). Through the encoding and initialisation described above, we can finally obtain an initialised individual **P**:

$$\mathbf{P} = [selected\_optim_1, ..., selected\_optim_i] \quad (4)$$

Fig. 1 presents an example of the encoding and initialisation of a GAN network structure. The top of this figure shows the structural design of the GAN generator and discriminator, which consists of a fully connected layer (*i.e.*, Linear) at the beginning and the end, and several convolution layers (*i.e.*, Conv2d) in the middle. In this case, the generator and discriminator of the GAN network are divided into a total of 7 sub-parameter layers *i.e.*, L1 $\sim$ L7, where activation layers such as LeakyReLu, Batch Nomalization, Tanh, Sigmoid, *etc.* are classified as the same sub-parameter layer as their corresponding fully connected or convolution layers. By the initialisation according to Eq. (1), we can obtain the initial state of each sub-array n_i in layers L1 to L7, where n_i denotes the initialised sub-array of the i-th sub-parameter layer. Here we predefine the optimiser array **Optims** as $[Adam, RMSprop, SGD]$, the maximum index value of each sub-array and the initialised individual **P** are obtained according to Eq. (3).

Note that, any encoding strategy can be similarly applied to this method as long as there are no duplicate encodings at the time of initialisation, and doing so ensures that Eq. (3) finds the maximum index value of the encoding and thus selects the optimiser for the sub-layer. Thus, integer encoding can also replace real encoding in this method, but binary encoding are not applicable.

B. Evolution of the GANs training algorithm

To assess the impact of the selected optimiser on each sub-parameter layer, we designed a fitness function to measure the performance of different sub-parameter layers. In GANs training, the generator aims to minimise a certain loss (*e.g.*, the discriminator's ability to distinguish between real and fake data), while the discriminator aims to maximise its loss (*e.g.*, the accuracy of distinguishing real from fake data). Since the chosen optimisers are bound to the iterative updates of the model parameters, ensuring that both the generator and the discriminator converge to their respective objectives, we utilise the generator loss and the discriminator loss to estimate model performance:

$$F_{loss} = -D_{loss} + G_{loss} \quad (5)$$

where D_{loss} and G_{loss} are calculated according to the related network models (as shown in Fig. 2, Section IV-B). Besides loss functions, we also utilize the Fréchet Inception Distance (FID) metric to assess quality and diversity of the generated data, *i.e.*,

$$F_{FID} = \|\mu_r - \mu_g\|_2^2 + Tr(\sum\nolimits_r + \sum\nolimits_g - 2(\sum\nolimits_r \sum\nolimits_g)^{\frac{1}{2}}) \quad (6)$$

FID measures the similarity between the distribution of generated samples and real samples. A lower FID score indicates that the generated data are closer to the real data distribution. In Eq. (6), μ_r and μ_g are the means, $\sum_r$ and $\sum_g$ are the covariance matrices of the real (subscripted as r) and generated data (subscripted as g) distributions, and $Tr()$ represents the trace operation. Then, the fitness function of an individual **P** for training GANs is defined as

$$F = F_{loss} + F_{FID} \quad (7)$$

A smaller value of F represents the better performance of the individual.

After defining the calculation of fitness, SGA is adopted to find the fittest individual through several evolutions. For each individual, the corresponding GANs are trained on a validation set (*e.g.*, 10% training images randomly sampled) and take the fitness on validation set as evaluation metric. Then a probability is assigned to each individual by comparing its fitness among the individuals in the current population:

$$Prob\_\mathbf{P}^k = \frac{F^{k-1}}{\sum_{k=1}^{K} F^k} \quad (8)$$

where $\mathbf{P}^k$ is the k-th individual in the population and K is the number of individuals in the population. The population in each iteration are regarded as parents, and selected according to Eq. (8). The selected parents breed another population as offspring using the following operations: selection, crossover, and mutation. In this phase we make use of the real-number encoding to effectively explore the search space with simulated binary crossover (SBX) and polynomial mutation [34]. In the implementation, we use the probability p_c to crossover a pair of selected parents and generate two offspring individuals, or the probability p_m to mutate two parents and generate two

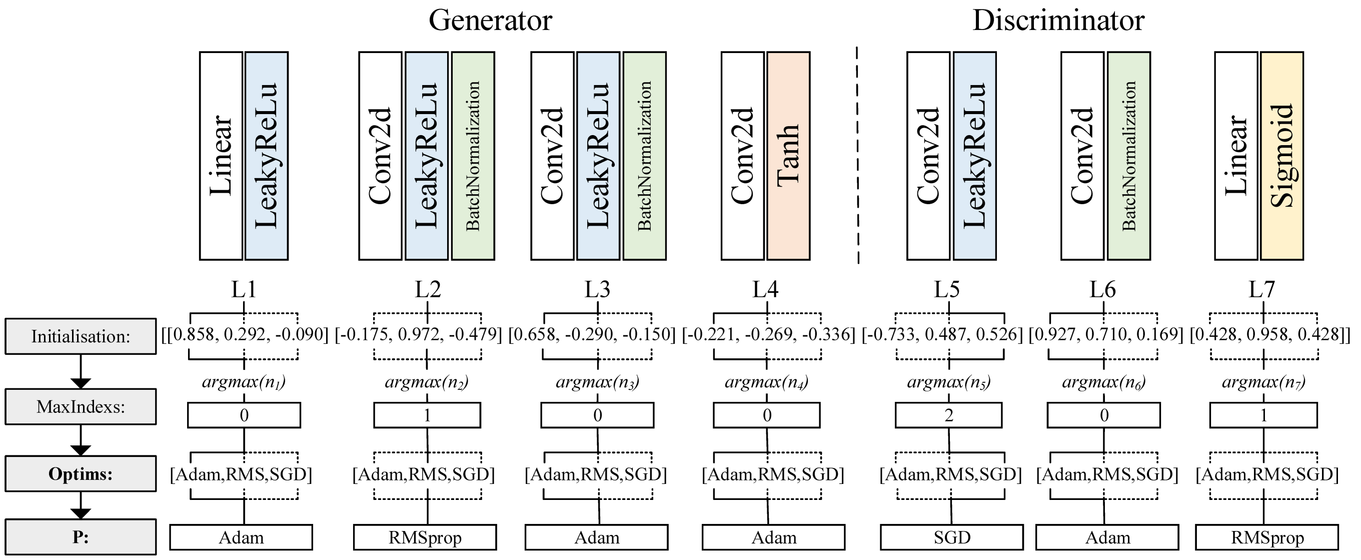


Fig. 1: An example of the encoding and initialisation of a GAN network structure. The division of the GAN network is shown at the top, and the bottom shows the encoding and initialisation of the optimiser selected for each layer. $argmax(\cdot)$ is illustrated in Eq. (3) which is used to find the maximum index of the initialised arrays, and $[Adam, RMSprop, SGD]$ is the predefined array of the alternative optimisers. The solid line in the figure indicates being selected and the dashed line indicates unselected.

mutated offspring solutions. To maintain the diversity of the algorithm, we do not use any elitism to preserve the best individuals from one generation to the next, instead, we choose to store in memory the local-best individual among the K individuals in each generation, and by repeating T generations, we finally obtain the best individual from the T local-best individuals.

Algorithm 1 outlines the detailed process of finding the best optimiser for each sub-parameter layer using the proposed methodology. It is worth noting that the proposed method is divided into two parts, *e.g.*, the evolutionary search process and the training process. Once obtaining the best optimisers from the search process, we apply them to the GANs network for training.

IV. Experimental Results

To validate the effectiveness of the proposed SGA-GANs, our experiment is carried out in two parts, the first part is to search for potentially good optimisers using the evolutionary process, and the second part is to apply the best optimisers chosen in the first part to the corresponding GANs network layer for training. All experiments are implemented under the PyTorch framework on an NVIDIA RTX3090 GPU.

Algorithm 1 Evolutionary process of finding the best optimiser for each GANs sub-parameter layer.

```
1: Require: Pre-defined **Optims**, hyper-parameters: maximum iteration $T$, population size $K$;
2: Initialise a population with $K$ individuals, each obtained based on Eq. (1), Eq. (2), Eq. (3);
3: Select the local-best individual $\hat{\mathbf{P}}^{(0)}$;
4: for $t = 1$ to $T$ do
5:     Calculate the fitness of each individual based on Eq. (7);
6:     Obtain selecting probabilities based on Eq. (8);
7:     Preserve the local-best individual $\hat{\mathbf{P}}^{(t-1)}$;
8:     for $k = 2$ to $K$ do
9:         Generate a random value $r \sim [0, 1]$;
10:        Conduct selection, crossover and mutation for generating new individuals according to $r$;
11:    end for
12:    Replace the population with the offspring population for the subsequent generation;
13: end for
Output: The best individual $\hat{\mathbf{P}}^{(T)}$.
```

A. Dataset Preparation

We utilize the CIFAR10 dataset, a collection of 60,000 32x32 colour images in 10 classes (*i.e.*, airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), with 6000 images per class. And there are 50,000 training images and 10,000 test images.

During the GANs evolutionary optimisation process, *i.e.*, the evolutionary process of finding the best optimiser for each GANs sub-parameter layer, we randomly sample 10% of the training images from each category as the validation set (5,000 images in total). After the evolutionary process, we can obtain the best combination of optimisers for the model and then use the remaining 45,000 training images to train the model. Finally we utilise the 10,000 test images as a benchmark to evaluate its generated images.

B. Introduction and Setup of the Benchmark Models

To verify the validity and portability of SGA-GANs, we conducted comparative experiments on five benchmark GANs models, which are GAN, deep convolutional GAN (DCGAN), boundary equilibrium GAN (BEGAN), wasserstein GAN (WGAN) and wasserstein GAN with gradient penalty (WGAN-GP). It is important to note the reason we choose these benchmark models, *i.e.*, each of them has unique features and improvements that have stimulated a significant amount of new work in GANs family and improved the quality of the generated content.

1) Introduction of the Benchmark Models: GAN [1] consists of two main components: a generator network and a discriminator network. The primary role of the discriminator (D) is to distinguish between real and generated samples, and the generator (G) aims to produce samples that are indistinguishable from real data. Formally, G receives a random noise z and generates a fake sample from this noise, denoted $G(z)$. The input of D is an image sample, which can be either a real sample x from the training dataset or a generated image $G(z)$, and its output is a float value between 0 and 1. The closer the value is to 1 means that the input sample is closer to the real sample, and vice versa. The training of the GAN is formulated as the optimisation of the following loss function:

$$\underset{G}{min}\, \underset{D}{max}\, \mathbb{E}_x[logD(x)] + \mathbb{E}_z[log(1 - D(G(z)))] \quad (9)$$

where $\mathbb{E}$ is the expectation. The loss function in (9) aims to minimise for G and maximise for D.

Four GAN variants were recently proposed to improve upon the GAN performance by means of architectural improvements and loss function modifications. Specifically, DCGAN [9] introduced architectural changes by utilizing deep convolutional neural networks in both the generator and discriminator, making it capable of learning hierarchical features from images effectively. This led to stable training and improved quality in image generation compared to the original GANs, which used fully connected layers and struggled with complex image data. BEGAN [35] introduced a novel loss function based on autoencoders and the concept of equilibrium. It achieves a balance between generator and discriminator, resulting in the generation of images with high visual quality and diversity. WGAN [26] innovated by introducing the wasserstein distance (also known as Earth Mover's distance) as the loss function, addressing the issues of mode collapse and training instability. This metric encourages smoother gradients, which helps in more stable training and the generation of diverse and high-quality images while mitigating problems associated with traditional GANs. WGAN-GP [13] extended the WGAN by introducing a gradient penalty term to enforce the Lipschitz constraint on the discriminator, ensuring that it provides meaningful gradients throughout the training process. This innovation significantly improved the stability of training and the quality of generated samples by penalizing sharp gradients and further mitigated mode collapse issues.

TABLE I: The number of sub-layers of the five benchmark models and the total number of training combinations corresponding to each model. Since the pre-defined optimisers is **Optims** $= [Adam, RMSprop, SGD]$, each layer has 3 choices.

Models	Sub-layers	Training Combinations
GAN	8	3^8
DCGAN	9	3^9
BEGAN	8	3^8
WGAN	8	3^8
WGAN-GP	8	3^8

After the brief introduction of the above models, let's review the specific setup for each model as follows (*i.e.*, the design of the loss function and the network structure).

2) Benchmark Models Setup: In the implementation, we train each benchmark model using the respective loss function and network structure from the original paper. The right side of Fig. 2 summarises the loss functions of the benchmark models. Wherein, min represents the minimisation operation, max is to maximise, x is the real data input, z is a noise input. For BEGAN model, k is a dynamically adjusted hyper-parameter that helps balance the contributions of the generator and discriminator. In WGAN-GP, λ is gradient penalty coefficient, and $\hat{x} \leftarrow \epsilon$x + (1 - ϵ) $\hat{x}$, $\epsilon \sim U$[0,1]. Source code including hyper-parameters of each model is taken from https://github.com/eriklindernoren/PyTorch-GAN.

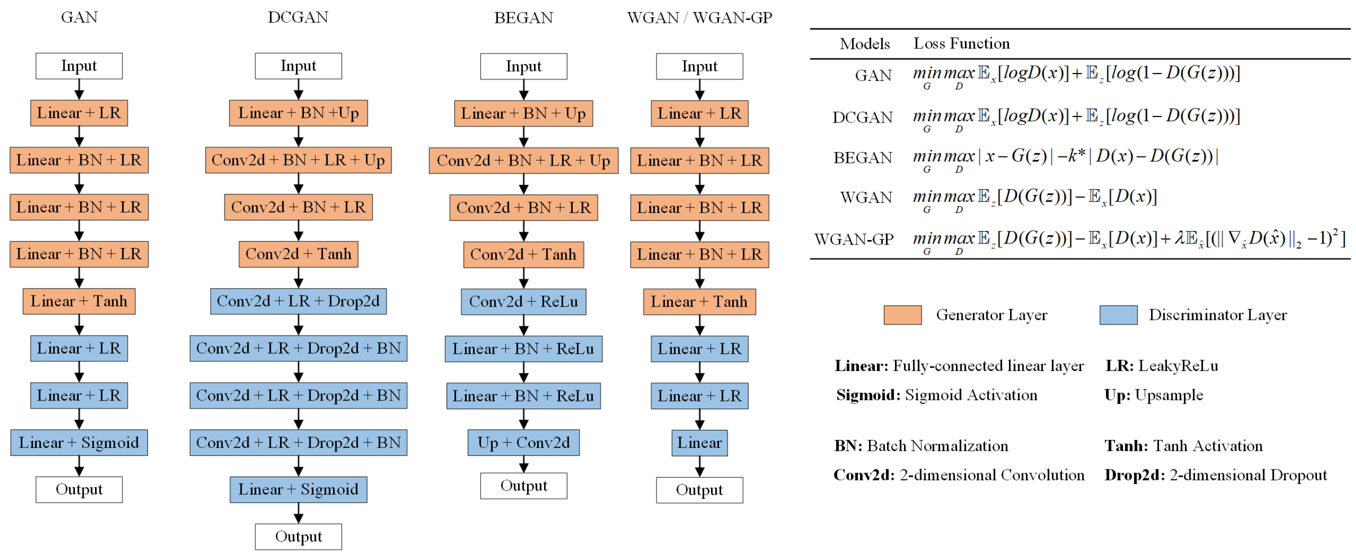
On the left-hand side of Fig. 2, we present the simplified architectural design of each benchmark model. According to the division method in Section III-A, we divide each network by blocks as shown in Fig. 2. For the pre-defined optimisers, we use the three most used optimisers in GANs family, *i.e.*, **Optims** $= [Adam, RMSprop, SGD]$. Table I summarises the number of sub-layers of the five benchmark models and the total number of training combinations corresponding to each model.

C. Hyper-parameter Settings

The number and order of the pre-defined optimisers (**Optims** $= [Adam, RMSprop, SGD]$) are consistent across all comparison experiments, where the hyper-parameters of each optimiser are set as shown in Table II. In the evolutionary process of finding the fittest optimisers, we set the maximum number of iterations of the population (*i.e.*, T) to 20, the population size (*i.e.*, K) to 10, with each individual training for 20 epochs, and the crossover (*i.e.*, p_c) and mutation (*i.e.*, p_m) probabilities of individuals to 0.7 and 0.1 respectively. These hyper-parameters are set empirically.

D. Results

In this section we show the results of the evolutionary process and the final results of the training by using the best optimisers, as well as the comparison with the single optimiser training approach.



Models	Loss Function
GAN	$\min_G \max_D \mathbb{E}_x[logD(x)] + \mathbb{E}_z[log(1 - D(G(z)))]$
DCGAN	$\min_G \max_D \mathbb{E}_x[logD(x)] + \mathbb{E}_z[log(1 - D(G(z)))]$
BEGAN	$\min_G \max_D \lvert x - G(z) \rvert - k^* \lvert D(x) - D(G(z)) \rvert$
WGAN	$\min_G \max_D \mathbb{E}_z[D(G(z))] - \mathbb{E}_x[D(x)]$
WGAN-GP	$\min_G \max_D \mathbb{E}_z[D(G(z))] - \mathbb{E}_x[D(x)] + \lambda \mathbb{E}_{\hat{x}}[(\lVert \nabla_{\hat{x}} D(\hat{x}) \rVert_2 - 1)^2]$

Fig. 2: The left-hand side displays the simplified architectural designs of GAN, DCGAN, BEGAN, WGAN and WGAN-GP, respectively. The right-hand side presents the loss function of each benchmark model.

TABLE II: Hyper-parameters setting of Adam, RMSprop, SGD.

Optimiser	Learning Rate	Other Parameters
Adam	0.0002	β_1=0.5, β_2=0.999
RMSprop	0.0001	momentum=0.9
SGD	0.01	momentum=0.9

TABLE III: Summary of the best optimiser arrays after the evolution of the 5 benchmark models. Wherein, 0: Adam, 1: RMSprop, 2: SGD.

Model	Optimisers of the Best Individual
GAN	[0, 1, 1, 1, 2, 2, 2, 1]
DCGAN	[0, 2, 0, 0, 0, 0, 1, 1, 2]
BEGAN	[0, 2, 0, 0, 2, 1, 1, 0]
WGAN	[1, 2, 0, 0, 2, 2, 0, 1]
WGAN-GP	[1, 1, 0, 1, 1, 0, 0, 0]

1) Results of the Evolutionary Process: With the dataset preparation in Section IV-A, the benchmark model setup in Section IV-B, and the parameter settings in Section IV-C, we obtained the evolutionary results of different benchmark models, which can be seen in Appendix A

The number of the selected strategies (*i.e.*, population size $K \times$ maximum iteration $T = 10 \times 20 = 200$) during the evolution of each model in this experiment is approximately 1% of the total number of training combinations (*i.e.*, as in Table I). Since no individual retention strategy was used during the evolutionary process, our method yielded more diverse strategies on each model compared to methods using elitist, with the fitness value of the optimal individual changing with each iteration, and the optimisers selected for sub-layers being updating with each iteration.

2) Comparison with Single Optimiser Training: To validate our proposed method in this paper, we conducted a comprehensive comparison between SGA-GANs and traditional single-optimiser approaches. Specifically, we compare the metric results, runtime, and generated images of five benchmark models during the training process of different optimisers. We employ the best combination of optimisers from the respective evolutionary process to establish and train each benchmark network, summarised in Table III.

Comparison of Metric Results. Throughout the training process, we recorded FID values for the generated images at 20-epoch intervals, spanning a total of 200 epochs. We compared the FID results with traditional single-optimiser training methods such as Adam, SGD, and RMSprop on consistent datasets, baseline models, and parameter configurations. Comparative results of FID on five benchmark models are presented at Appendix B.

Compared to the other three traditional optimisation methods using a single optimiser, our proposed method achieves a relatively lower FID value at the 200th epoch of each benchmark model. In addition, the average FID from the 20th to the 200th epoch achieved by our method outperforms the averages of single-optimiser-training methods, which validates the effectiveness of our method for the improvement of the quality of the generated images, and also confirms the good portability of our method.

Comparison of Runtime. During the optimisation of the parameters of each model (Table IV summarises the number of parameters for each model), it is worth noting that training each model with our optimised combination of optimisers takes slightly less time than the approaches of training using a single optimiser, as shown in Table V, which compares the time consumption for training each model using different

TABLE IV: Number of parameters in each model.

Model	G	D	Total
GAN	3,855,232	1,704,961	5,560,193
DCGAN	1,051,139	98,401	1,149,540
BEGAN	1,051,139	1,101,347	2,152,486
WGAN/WGAN-GP	3,855,232	1,704,961	6,560,193

TABLE V: Time consumption for training each model using different optimisers (hour).

Model	Using Adam	Using RMSprop	Using SGD	Ours
GAN	1.40	1.56	2.13	1.33
DCGAN	3.76	2.75	2.91	2.36
BEGAN	1.78	1.86	1.50	1.41
WGAN	0.86	0.85	0.85	0.78
WGAN-GP	1.26	1.26	1.15	1.10

optimisers.

Comparison of Generated Images. As presented Appendix C where the outcome of the comparison of images generated by different models is summarised, the results of training GAN, DCGAN, BEGAN, WGAN and WGAN-GP by using RMSprop and SGD optimisers show more defects, which are manifested as mosaic images generated by GAN and DCGAN, and blurry generated images on BEGAN, WGAN and WGAN-GP. Meanwhile, compared to training each benchmark model with RMSprop and SGD optimisers, training with the Adam optimiser achieves relatively better results, as the images generated by GAN and DCGAN are no longer mosaic images but images with the outlines of the items, and in particular, the images generated by DCGAN are more clear. The quality of the generated images using Adam on BEGAN, WGAN and WGAN-GP is less improved compared to using RMSprop and SGD, *i.e.*, they have slightly clearer item outlines on BEGAN and WGAN-GP, and there is no significant visual improvement on WGAN.

Reviewing the generated images of single optimiser training methods, the best visual results are achieved using the Adam optimiser for the five benchmark models in this experiment, which justifies the preference of most GANs research to train the models using the Adam optimiser. However, the training results of the proposed SGA-GANs validate our assumption in the Introduction that the use of a single optimiser limits the performance improvement of GANs to some extent. the generated images of our method on GAN, BEGAN, WGAN, and WGAN-GP are sharper and the shape of the target is more obvious compared to the single-optimiser methods. At the same time, compared to the good results of using Adam on DCGAN , the background of our generated images is less blurred.

V. Conclusion

A new method for training GANs by using multiple optimisers based on evolutionary computation (SGA-GANs) is presented in this paper. To improve the overall performance of GANs, we use a real-number data structure to encode the optimisers chosen for sub-network layers and then apply an evolutionary algorithm to evolve a population of arrays of the chosen optimisers by initialization, selection, evaluation, and variation, to preserve the best combination of oprimisers. Experiments show that the implementation of the proposed SGA-GANs outperforms traditional single optimiser training methods, and thus the generator can generate images with higher quality. At the same time, the SGA-GANs is also well portable, *i.e.*, any neural network can be trained and optimised using our method as long as it is possible to divide the whole neural network into several sub-layers. Numerical results indicate that the proposed approach is promising and can be exported on any neural network model, and it may be considered as a new tool to improve upon the network performance.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (NO.62376127, NO.61876089, NO.61876185, NO.61902281, NO.61403206), by Natural Science Foundation of Jiangsu Province (NO.BK20141005), by Natural Science Foundation of the Jiangsu Higher Education Institutions of China (NO.14KJB520025), and partially supported by the project of Distinguished Professors of Jiangsu Province.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[2] L. Gao, J. Zhu, J. Song, F. Zheng, and H. T. Shen, "Lab2pix: label-adaptive generative adversarial network for unsupervised image synthesis," in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 3734–3742.

[3] W. Xu, C. Long, R. Wang, and G. Wang, "Drb-gan: A dynamic resblock generative adversarial network for artistic style transfer," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 6383–6392.

[4] B. Bosquet, D. Cores, L. Seidenari, V. M. Brea, M. Mucientes, and A. Del Bimbo, "A full data augmentation pipeline for small object detection based on generative adversarial networks," *Pattern Recognition*, vol. 133, p. 108998, 2023.

[5] J. Guerreiro, P. Tomás, N. Garcia, and H. Aidos, "Super-resolution of magnetic resonance images using generative adversarial networks," *Computerized Medical Imaging and Graphics*, p. 102280, 2023.

[6] Z. Zheng, Y. Bin, X. Lu, Y. Wu, Y. Yang, and H. T. Shen, "Asynchronous generative adversarial network for asymmetric unpaired image-to-image translation," *IEEE Transactions on Multimedia*, 2022.

[7] A. Jabbar, X. Li, and B. Omar, "A survey on generative adversarial networks: Variants, applications, and training," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–49, 2021.

[8] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[9] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[10] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2794–2802.

[11] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223.

[12] A. Brock, J. Donahue, and K. Simonyan, “Large scale gan training for high fidelity natural image synthesis,” *arXiv preprint arXiv:1809.11096*, 2018.

[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of wasserstein gans,” *Advances in neural information processing systems*, vol. 30, 2017.

[14] Z. Li, P. Xia, R. Tao, H. Niu, and B. Li, “A new perspective on stabilizing gans training: Direct adversarial training,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 7, no. 1, pp. 178–189, 2022.

[15] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[16] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT’2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*. Springer, 2010, pp. 177–186.

[17] T. Tieleman and G. Hinton, “Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning,” *Technical report*, 2017.

[18] Y. Xue, Y. Tong, and F. Neri, “An ensemble of differential evolution and adam for training feed-forward neural networks,” *Information Sciences*, vol. 608, pp. 453–471, 2022.

[19] X. Li and X. Yao, “Cooperatively coevolving particle swarms for large scale optimization,” *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 210–224, 2012.

[20] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, “Cooperative co-evolution with differential grouping for large scale optimization,” *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 378–393, 2014.

[21] F. Caraffini, F. Neri, and G. Iacca, “Large scale problems in practice: The effect of dimensionality on the interaction among variables,” in *Applications of Evolutionary Computation - 20th European Conference, EvoApplications 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, Part I*, ser. Lecture Notes in Computer Science, G. Squillero and K. Sim, Eds., vol. 10199, 2017, pp. 636–652.

[22] J. G. Cavalcanti Costa, Y. Mei, and M. Zhang, “An evolutionary hyper-heuristic approach to the large scale vehicle routing problem,” in *2021 IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 2109–2116.

[23] J.-Y. Li, Z.-H. Zhan, K. C. Tan, and J. Zhang, “Dual differential grouping: A more general decomposition method for large-scale optimization,” *IEEE Transactions on Cybernetics*, vol. 53, no. 6, pp. 3624–3638, 2023.

[24] X. Yan, B. Cui, Y. Xu, P. Shi, and Z. Wang, “A method of information protection for collaborative deep learning under gan model attack,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 18, no. 3, pp. 871–881, 2019.

[25] Y. Nesterov, “A method for unconstrained convex minimization problem with the rate of convergence o (1/k2),” in *Dokl. Akad. Nauk. SSSR*, vol. 269, no. 3, 1983, p. 543.

[26] M. Arjovsky and L. Bottou, “Towards principled methods for training generative adversarial networks,” *arXiv preprint arXiv:1701.04862*, 2017.

[27] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of adam and beyond,” *arXiv preprint arXiv:1904.09237*, 2019.

[28] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, “A stochastic quasi-newton method for large-scale optimization,” *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 1008–1031, 2016.

[29] L. Xu, L. Zheng, W. Li, Z. Chen, W. Song, Y. Deng, Y. Chang, J. Xiao, and B. Yuan, “Nvae-gan based approach for unsupervised time series anomaly detection,” *arXiv preprint arXiv:2101.02908*, 2021.

[30] H. Kim, C. Wang, H. Byun, W. Hu, S. Kim, Q. Jiao, and T. H. Lee, “Variable three-term conjugate gradient method for training artificial neural networks,” *Neural Networks*, vol. 159, pp. 125–136, 2023.

[31] Y. Liu, Z. Shu, Y. Li, Z. Lin, F. Perazzi, and S.-Y. Kung, “Content-aware gan compression,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12 156–12 166.

[32] D. Wang, X. Qin, F. Song, and L. Cheng, “Stabilizing training of generative adversarial nets via langevin stein variational gradient descent,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 7, pp. 2768–2780, 2020.

[33] J. Zhuang, T. Tang, Y. Ding, S. C. Tatikonda, N. Dvornek, X. Papademetris, and J. Duncan, “Adabelief optimizer: Adapting stepsizes by the belief in observed gradients,” *Advances in neural information processing systems*, vol. 33, pp. 18 795–18 806, 2020.

[34] K. Deb, M. Goyal *et al.*, “A combined genetic adaptive search (geneas) for engineering design,” *Computer Science and informatics*, vol. 26, pp. 30–45, 1996.

[35] D. Berthelot, T. Schumm, and L. Metz, “Began: Boundary equilibrium generative adversarial networks,” *arXiv preprint arXiv:1703.10717*, 2017.

Appendix A
Results of Evolutionary Process

We show the evolutionary results of each benchmark model at https://github.com/lizzhang-spec/SGA-GANs/tree/main/Evolutionary-Results. Therein, the line graph on the left shows the fitness value corresponding to its best individual at each iteration, the right-hand shows the optimisers selected for each iteration of sub-layers , the number 0 denotes the Adam optimiser, 1 denotes the RMSprop optimiser and 2 denotes the SGD optimiser. Considering that the fitness function (*i.e.* Eq. (7)) designed for this method aims to minimise each sub-objective, a lower fitness value indicates a better performance of the individual. Therefore, the red minimum point in the line graph is the optimal individual in the evolutionary process of each model.

Appendix B
Comparison of Metric Results

We present the evaluation metric results of the generated images by GAN, DCGAN, BEGAN, WGAN and WGAN-GP at https://github.com/lizzhang-spec/SGA-GANs/tree/main/FID-Results. We use FID to evaluate the quality of generated images. The lower value of FID means that the distribution of generated images is closer to the distribution of real images. The last column of the bottom part of these figures is the average FID values from the 20th epoch to the 200th epoch.

Appendix C
Comparison of Generated Images

This part shows the generated images by different methods, *i.e.*, training by Adam only (_adam), RMSprop only (_RMSprop), SGD only (_sgd) and our method (_SGA). The results on GAN, DCGAN, BEGAN, WGAN and WGAN-GP models are shown at https://github.com/lizzhang-spec/SGA-GANs/tree/main/Generated-Images.