Speech Synthesis Based on GAN

Zhang Wentao, Guo Yachen

Abstract—The rapid development of neural networks has brought more possibilities for applying artificial intelligence. In recent years, Generative Adversarial Networks (GANs) have attracted global attention due to their unsupervised training mechanism and the potential for development through mutual adversarial enhancement. Speech generation models have wide practical significance and can be applied to the underlying implementation of functions such as text-to-speech or virtual voices. Currently, most mature speech models in mainstream deep learning models are based on Convolutional Neural Networks (CNNs). The main research on GAN models is currently focused on image training, while related work on audio training is still in its early stages. This paper explores the feasibility of speech generation using GANs and designs a speech generation model based on GANs to obtain high-quality generated speech, laying the foundation for subsequent other speech functions.
This paper first elaborates on the development of GANs. It provides a brief introduction to basic GAN and speech synthesis concepts. Next, it describes the design of a GAN model composed of a generator based on the Google speech synthesis model and a discriminator based on spectral normalization. It provides a detailed explanation and analysis of the related debugging results. Finally, this paper evaluates the generated results and analyzes the related engineering problems.

June 5, 2021

Index Terms—TTS, GAN, Speech, Synthesis

1 Introduction

1.1 Research Background and Significance.

In recent years, the rapid development of neural networks has brought more possibilities for the practical application of artificial intelligence [1]. Among them, the Generative Adversarial Network (GAN) proposed by Ian Goodfellow in 2014 has attracted widespread attention in the industry. Its unsupervised and label-free characteristics make it simpler, more efficient, and more potent than other neural network models. Since 2014, numerous GAN-based optimization models have emerged. The Deep Convolutional GAN (DCGAN) has achieved excellent training results, leading GANs into the mainstream development stage. Currently, GANs have shown great success in various fields [2].

Speech generation technology has a relatively early starting point, with the appearance of a new computer speech output system in the late 1970s. This system used the most basic speech units to concatenate relevant synthetic speech. Subsequently, various models based on speech feature synthesis have continuously emerged. The most representative one is the DECTalk, successfully developed in 1982.

In recent years, GANs have made significant progress in image generation. However, GANs have received little attention in the audio field. Currently, most mature speech models are based on Convolutional Neural Networks (CNNs), with prominent examples being Google's WaveNet [3] model and Baidu's Deep Voice model. However, these models have drawbacks, such as not supporting parallel training and slow speed.

Due to the limited exploration of GANs in speech synthesis, this paper focuses on exploring the feasibility of using GANs for speech generation and designing a speech generation model based on GANs. The aim is to simplify the training difficulty and improve the speed and reliability of training through the unique adversarial and balancing properties of GAN models. This is done to obtain good-quality generated speech and lay a foundation for subsequent speech-related functionalities.

1.2 Review of Current Research

Currently, research on GANs in China mainly focuses on applications and reviews. However, the overall number of papers is still relatively small. According to the search results on the China National Knowledge Infrastructure (CNKI), there are only 2685 records of GAN-related papers published on the platform. Most research efforts focus on how to optimize GAN-derived models and implement them in practical applications, with only a few papers independently designing GAN-derived models, such as the master's thesis by Zixuan Wang in 2020 titled "FLOWGAN: Research on Key Technologies for Encrypted Traffic Recognition Based on Generative Adversarial Network." Research on GANs in the field of speech in China is scarce. Only 56 records of GAN-related papers on speech were published on CNKI, all of which are application-oriented, with no independent model design.

Research on GANs abroad mainly focuses on applications, reviews, and model design. GANs have spawned various models with different characteristics, such as the Deep Convolutional GAN (DCGAN) based on deep convolutional neural networks and the Conditional GAN (cGAN) with relevant conditional constraints. Research on GANs in audio synthesis has made some progress abroad, with two high-quality representative models being WaveGAN and GANSynth. Both successfully applied GANs to simple audio datasets. The former used a GAN model to generate raw audio, obtaining a dataset of spoken digit commands from zero to nine. The latter created a dataset of single-note recordings from various instruments by training a GAN to generate the reversible spectrogram of musical notes.

Neekhara et al. proposed an adversarial vocoder model that can synthesize amplitude spectrograms from melody spectrograms generated by Tacotron 2, incorporating related local weighting and phase estimation techniques. Overall, although GANs have made some development in speech synthesis, research on GANs in the field of speech synthesis still needs to be a popular topic [4].

1.3 Research Content and Research Methods

There is still significant room for the development of GAN in speech synthesis. This paper focuses on designing a GAN model consisting of a generator based on the Google speech synthesis model and a discriminator based on spectral normalization. For any neural network training in deep learning frameworks, feature extraction issues must be taken seriously. Learning analysis can only be conducted when the dataset has effective identifying features. This paper designs a speech preprocessing format that matches the GAN model from the perspective of the Mel-Frequency domain to assist the GAN model in better training. Experimental contents were then conducted, and the related debugging results were analyzed and explained in detail. Finally, the results were analyzed, and complex engineering issues were discussed accordingly.

This paper conducted a wide-ranging investigation of the development of GAN in speech synthesis and the use of related models through the survey method to understand the research status and related issues. Secondly, this paper analyzed the experimental research of other researchers on GAN through the literature research method and reasonably borrowed other, more complete GAN models to assist in designing the core of this paper's GAN and other related modules. Meanwhile, relevant technical reference documents were studied and referred to for help with the required technical content for the experiment. Finally, this paper used experimental and comparative methods for the related model design. It evaluated the rationality of the model through a series of experiments, analyzing the related experimental data to evaluate the model's validity and selecting the best model design based on data comparison.

2 Introduction to Related Technical

2.1 Generative Adversarial Networks

2.1.1 Overview of GAN in Speech

GAN, proposed by Ian Goodfellow in 2014, is a novel implicit neural network training model inspired by the Nash equilibrium in game theory [5]. GAN consists of two independent neural networks [6], the generator, and the discriminator.

The discriminator plays a role similar to traditional deep learning models, which is to distinguish and verify data in the dataset while learning the main features of the data to improve its ability to distinguish data. The generator's task is to generate fake data, attempting to fool the discriminator's discernment. The generator's data comes entirely from random noise distributed uniformly, which is expanded through a reverse neural network model to generate data similar to the target data. When the discriminator judges the

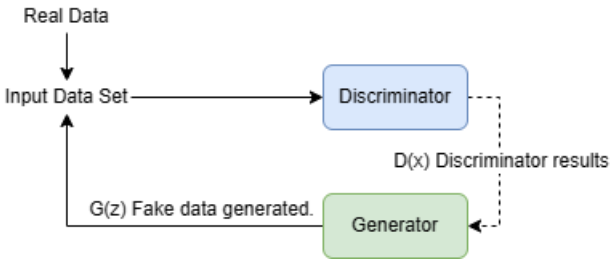

Fig. 1. GAN schematic

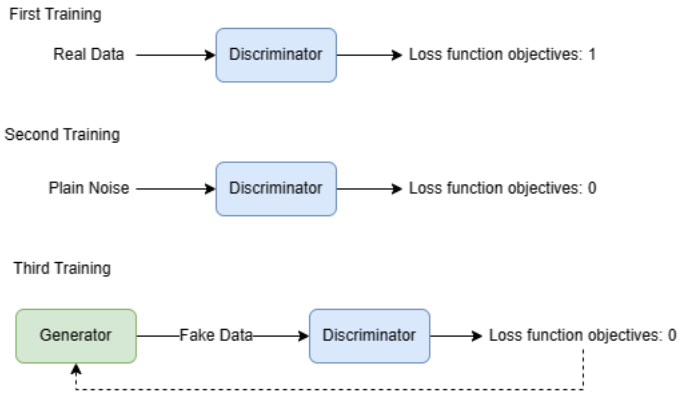

Fig. 2. GAN three times training decomposition diagram

data, it gives a probability indicating the degree of truthfulness, limited between 0 and 1. After the discriminator judges the fake data generated by the generator, the probability value is fed back to the generator. The generator adjusts the relevant parameters according to the probability value for better generation results [7]. The objective function of GAN is:

$$min_G max_D V(D,G) = E_{x-P(x)}[\log D(x)] + E_{z-P(z)}[\log\Big(1 - D\big(G(z)\big)\Big)] \quad (1)$$

GANs avoid using complex Markov chains for gradient acquisition, instead employing backpropagation algorithms to compute gradients based on the probability values of the degree of truth or falsity mentioned above and to adjust the relevant parameters. The structure of GAN is simple, and the learning cost is low without excessive complex inference calculations. However, it can generate objectively effective data results due to its adversarial characteristics. The data results generated by GAN also have high quality, and compared with other deep learning models, GAN has greater development advantages.

While GANs have many advantages, they also inevitably have some drawbacks. First, the training of GANs must effectively control the learning speed of the generator and discriminator to be balanced. When the learning speed of the generator is faster than that of the discriminator, the generator can easily fool the discriminator, making it difficult to adjust and optimize subsequently; when the learning speed of the discriminator is faster than that of the generator, the discriminator generally gives lower scores, causing the generator to lose its optimization direction and unable to

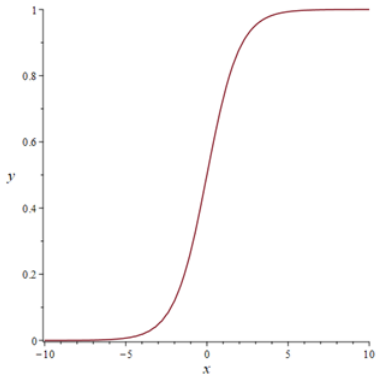

Fig. 3. Sigmoid function

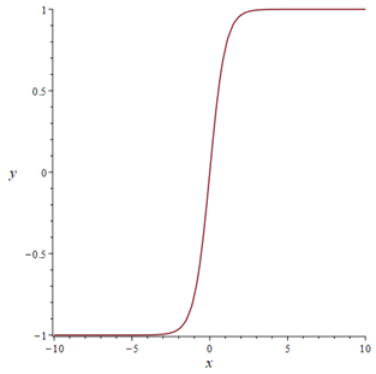

Fig. 4. tanh function

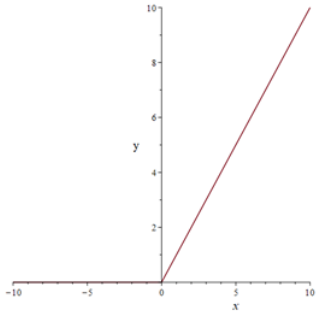

Fig. 5. ReLU function

optimize effectively and rationally, resulting in blind optimization or even reverse optimization. Second, GANs suffer from mode collapse, i.e., some samples with only minor differences are treated as multiple samples with significant differences, leading to convergence to a single result; and GANs also have the problem of gradient disappearance, i.e., when there is no overlap or negligible overlap between real and generated samples, the divergence of the objective function optimized by the network becomes a constant [8].

Compared with other traditional speech synthesis areas and other traditional deep learning network frameworks, GANs can more effectively conduct unsupervised training, allowing computers to find core internal relationships independently without the need for excessive data preprocessing. At the same time, traditional speech synthesis models tend to use phoneme elements for synthesis and concatenation, which rely heavily on basic preprocessing and analysis of each sample, while GANs are relatively robust, able to resist certain disadvantages of improper sample processing, and have higher training speed than traditional speech synthesis systems.

2.1.2 Activation Functions

The precursor of activation functions was used to simulate digital circuit switches. Due to its similarity to the behavior of linear perceptrons in neural networks, it was applied to map the output of a neuron to the next one using a special function relationship. The core purpose of activation functions is to introduce nonlinearity to neural network models, enabling them to approximate any nonlinear function. Currently, there are several popular activation functions, including:

1) Sigmoid function

$$sigmoid : f(x) = \frac{1}{1 + e^x} \tag{2}$$

The sigmoid function has the advantage of simple derivative calculation and a stable output mapping range between (0, 1), resulting in a limited output range and stable optimization. However, it suffers from severe function saturation and gradient vanishing problems, which means that the gradient value approximates zero when the function approaches a huge value, making it difficult to optimize the model effectively.

2) Hyperbolic tangent function

$$tanh : f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

The hyperbolic tangent function (tanh) is an optimization of the sigmoid function, which addresses the non-zero mean in the sigmoid. However, it still needs to solve the problem of gradient vanishing in the sigmoid function [9].

3) Rectified Linear Unit

$$ReLU = max(0, x) \tag{4}$$

The Rectified Linear Unit (ReLU) function [10] is currently one of the most popular activation functions used in neural networks. It effectively addresses the problem of vanishing gradients present in earlier activation functions. Its simplified function design also results in significantly faster computation. Moreover, compared to the sigmoid and tanh functions, ReLU exhibits much faster convergence due to its inherent one-sided saturation property [11].

However, ReLU also has some disadvantages. First, its output does not satisfy zero-mean normalization. Second, it is prone to the Dead ReLU Problem, where certain neurons' respective parameters may never be activated or updated during the later stages of training.

4) Leaky ReLU

$$LeakyReLU = max(ax, x) \tag{5}$$

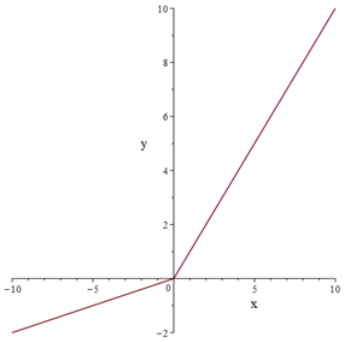

Fig. 6. Leaky ReLU function ($\alpha = 0.2$)

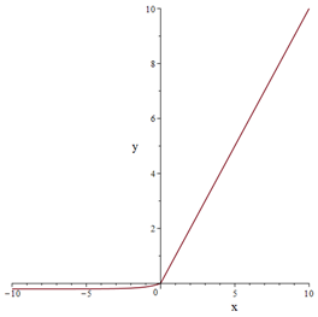

Fig. 7. ELU function ($\alpha = 0.2$)

The Leaky Rectified Linear Unit (Leaky ReLU) function, an optimized version of the ReLU function, was proposed for acoustic modeling. It addresses the Dead ReLU Problem by introducing a small slope to the negative direction. While theoretically, Leaky ReLU has all the advantages of ReLU. It has yet to be proven that Leaky ReLU is always superior to ReLU in practical applications [12].

5) Parameterized ReLU
Parameterized Rectified Linear Unit (PReLU) is an activation function that designs the slope parameter alpha in Leaky ReLU as a learnable parameter. This provides more flexibility but may require more adjustment costs overall.
6) Exponential Linear Unit

$$ELU : f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{else} \end{cases} \tag{6}$$

The Exponential Linear Unit (ELU) theoretically combines all the advantages of the activation above functions, achieving the goal of output zero-mean normalization and having one-sided saturation for effective convergence. However, its more complex equation increases the computational burden. Its practical application depends on testing and selection based on the specific situation.

2.1.3 Optimizer

The optimizer is used to adjust the parameters in a model. It calculates the values of parameters that need to be adjusted. It adjusts the relevant parameters in the neural network to achieve the optimal solution for the model.

1) Gradient Descent

$$W_{t+1} = W_t + \eta_t \Delta J(W_t) \tag{7}$$

In formula:
W_t — Learning parameters at time t
η_t — Learning Rate
$\Delta J(W_t)$ — Gradient
The Standard Gradient Descent algorithm (GD) is the most primitive and fundamental method for adjusting parameters by continually reducing them along the gradient direction, minimizing cost parameters. However, the GD method has many drawbacks, including slow training speed due to updating parameters at every step and the risk of getting stuck in a local optimum, finding saddle points or local minima rather than the global minimum.
2) Batch Gradient Descent

$$W_{t+1} = W_t + \eta_t \sum_{i=1}^{n} \Delta J(W_t, X^{(i)}, Y^{(i)}) \tag{8}$$

In formula:
$X^{(i)}$ — Input Sample
$Y^{(i)}$ — Output Sample
Batch Gradient Descent (BGD) combines parameter updates with all input samples, adjusting only after batch sample inputs. This speeds up the overall training process and allows for basic regional adjustments, reducing the likelihood of getting stuck in local optima.
3) Stochastic Gradient Descent

$$W_{t+1} = W_t - \eta_t g_t \tag{9}$$

$$g_t = \Delta J_{i_s}(W_t, X^{(i_s)}, Y^{(i_s)}) \quad i_s \in \{1, 2, \ldots, n\} \tag{10}$$

In formula:
i_s — Randomly Selected Direction
Stochastic Gradient Descent (SGD) is a method that utilizes randomness and perturbations to converge on the optimal solution ultimately. It can address the local optima that are susceptible to being disturbed by stochastic perturbations. However, more is needed to solve the problem of local optima fundamentally. Additionally, due to its stochastic nature, the computational speed of SGD is significantly improved. However, randomness also introduces noise, which can cause the direction of weight updates to be incorrect.
4) Mini-Batch Gradient Descent
Mini-Batch Gradient Descent (MBGD) is a combined approach that integrates the benefits of both Batch Gradient Descent (BGD) and Stochastic Gradient Descent (SGD). Compared to BGD, MBGD reduces the size of the batch by utilizing a subset of the samples, accelerating the overall training speed.
5) Momentum

$$m_t = \mu \times m_{t-1} + g_t \tag{11}$$

$$\Delta\theta_t = -\eta \times m_t \tag{12}$$

The momentum method simulates momentum in physics by introducing a new variable to accumulate the previous momentum instead of the corresponding gradient. This technique effectively optimizes the issue of GD-type algorithms not accumulating data from previous iterations, thereby accelerating the overall convergence. Additionally, momentum can enhance the update magnitude and help escape from local minima, effectively alleviating situations where local optima exist.

6) Nesterov

$$g_t = \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta \times \mu \times m_{t-1}) \tag{13}$$

Nesterov's method is an improvement upon the momentum method. It allows for considering the previous variable m_{t-1} in calculating the gradient, which introduces a certain correction amount.

7) Adaptive Gradient

$$n_t = n_{t-1} + g_t^2 \tag{14}$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{n_t + \epsilon}} \times g_t \tag{15}$$

In formula:

ϵ — Global Learning Rate

Adaptive Gradient (AdaGrad) is the first algorithm to introduce adaptive adjustments. It includes a constraint term that can appropriately amplify when gradients are small in the early stages and appropriately reduce when gradients are too large in the later stages. AdaGrad is suitable for sparse gradients; however, it still relies on the set learning rate.

8) Adadelta

This algorithm is an improvement upon AdaGrad. Compared to AdaGrad, Adadelta only accumulates fixed-size terms rather than all gradient squares. Moreover, Adadelta no longer requires a learning rate by employing a similar operation to the Newton iteration method. Its advantages include effectively accelerating early-stage training, but it can cause severe shaking near the optimal solution in the later stages.

9) RMSprop

$$AdaGrad : r \leftarrow r + g \bigodot g \tag{16}$$

$$RMSprop : r \leftarrow \rho r + (1-\rho) g \bigodot g \tag{17}$$

In formula:

ρ — Decay factor, used to control how much historical gradient information is obtained

It is also a modified version of the AdaGrad algorithm. Experimentally, RMSprop's algorithm has proven more effective than AdaGrad in speed and accuracy.

10) Adam [13]

This algorithm is a combination of the momentum method and the adaptive learning rate method. It possesses both the accelerating convergence feature of the momentum method and the automatic learning rate adjustment feature. It is one of the most widely used optimization algorithms due to its stronger optimization efficiency and wider applicability.

2.1.4 Loss function

The loss function is used to evaluate the difference between the predicted values of a model and the true values. It is an important function for measuring the performance of a model.

1) 0 - 1 function

$$L\big(Y, f(x)\big) = \begin{cases} 1, Y \neq f(x) \\ 0, Y = f(x) \end{cases} \tag{18}$$

When the predicted value equals the true value, the result is 0, and 1 otherwise. This function is often used to determine the number of corresponding errors directly. However, since achieving perfect results in practical situations is impossible, some requirements are often relaxed in daily use, allowing for a certain degree of difference.

$$L\big(Y, f(x)\big) = \begin{cases} 1, |Y - f(x)| \geq T \\ 0, |Y - f(x)| < T \end{cases} \tag{19}$$

2) Absolute value loss function

$$L\big(Y, f(x)\big) = |Y - f(x)| \tag{20}$$

The absolute loss function is calculated as the difference between the predicted and true values. This type of loss function can describe the loss situation in contrast to the 0-1 loss function. However, it still has drawbacks, such as a need for more control over the numerical range in the case of large differences and potential difficulty in identifying significant features.

3) Logarithmic loss function

$$L\big(Y, f(x)\big) = -\log P(Y|f(x)) \tag{21}$$

The logarithmic function, also known as the logistic regression function, is a mapping function that maps the value range of the absolute loss function to the probability distribution range and thus can represent the probability distribution very well. However, it has some drawbacks, such as low robustness.

4) MSELoss

$$L\big(Y, f(x)\big) = \big(Y - f(x)\big)^2 \tag{22}$$

The mean squared error function (MSE loss) computes the squared difference between the predicted and true values.

5) L1 Loss [14]

$$L(Y, f(x)) = \frac{\sum_{i=1}^{n} |y_i + y_i^p|}{n} \tag{23}$$

The mean absolute error loss function (L1 loss) is used to compute the average loss value for a set of results. This type of function is commonly used when there are multiple output values.

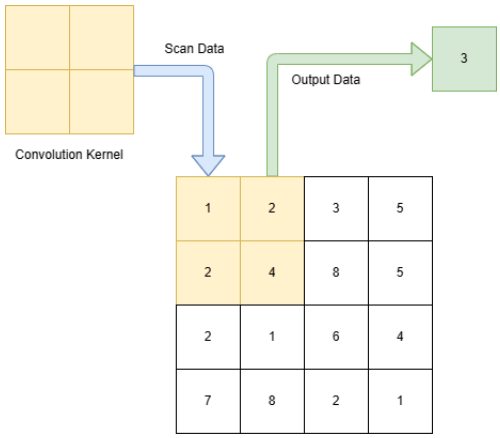

Fig. 8. Convolution kernel schematic

2.1.5 Convolution

Convolution techniques are developed to deal with the analysis of localized features. It focuses on analyzing the features of each local part and extract. It simplifies them to obtain a summary of localized information. This technique can effectively extract the overall feature framework without being influenced by too many irrelevant small details. However, when adjusting the convolution-related parameters, we must also consider that the core parameters must be adjusted according to the corresponding situation. Improper parameters may lead to excessive loss of details and affect the final training results.

The following characteristics mainly determine the core features of convolution:

1) Convolution Kernel
   The convolution kernel refers to the size of the area for local analysis taken at once. The convolution kernel determines the standard for filtering details. An appropriate convolution kernel can directly affect the efficiency of convolution.
2) Stride
   Stride refers to the distance that the convolution kernel moves each time it is scanned in a certain direction. The setting of the stride should be determined in combination with the convolution kernel and the data situation. In the case of a very large dataset, a stride that is too small may not be able to extract the corresponding information quickly and effectively. However, in the case of limited data size, a stride that is too large may result in the discarding of many effective data.
3) Padding
   Padding refers to adding meaningless all-zero or all-one data around the data when performing convolution scanning because the data on edge cannot be convolved separately by the convolution kernel. In order to effectively analyze the edge data, it is necessary to add padding around the data so that the convolution kernel can effectively scan the edge-related data.
4) Group
   Convolution layers can also be grouped to train an input result simultaneously. For example, when the groups are set to 2, it is equivalent to having two convolution layers, each of which calculates half of the input channels and produces half of the output channels. Finally, the relevant results are merged to obtain a complete output.
5) Deconvolution
   In generating content using convolution, related deconvolution techniques are also used. That is, by reversing convolution techniques, the data can be expanded reversely.

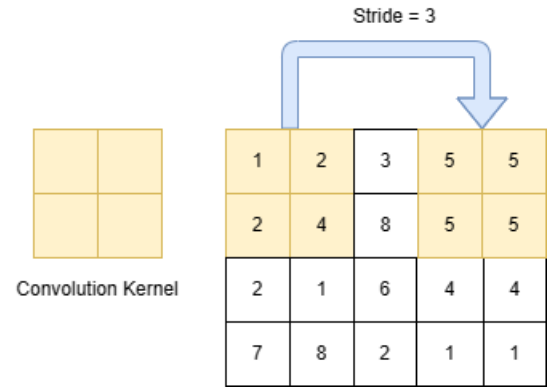

Fig. 9. Stride schematic

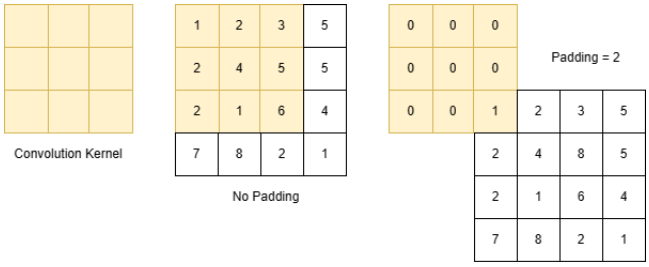

Fig. 10. Padding schematic

2.1.6 Batch

Batch processing technology solves the problem of data processing and model optimization adjustment. The batch size determines how much data are analyzed and processed before optimizing the model. The size of the batch directly affects the generation of model weights. A batch that is too large will affect the use of memory, while a batch that is too small will make the advantage of batch processing ineffective. Therefore, choosing a suitable batch size requires corresponding adjustment and optimization in application operations.

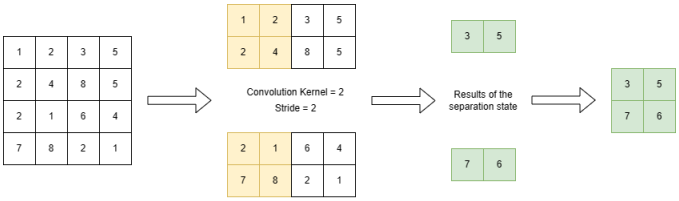

Fig. 11. Group training schematic

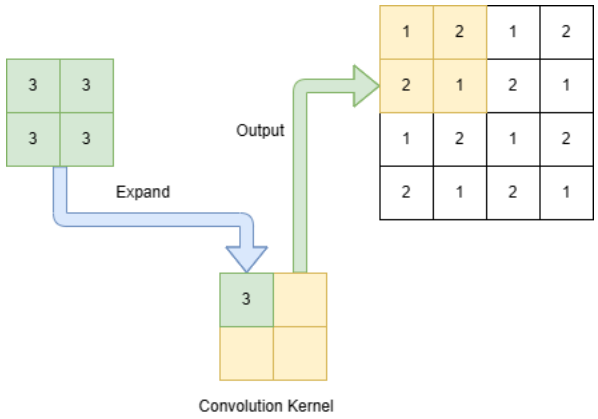

Fig. 12. Deconvolution schematic

2.2 Introduction to Speech Signal Processing

A speech signal is an analog signal. Unlike image signals, speech signals require feature analysis and preprocessing during acquisition. For GANs to effectively process related data, the data being processed must have sufficient feature characteristics. Therefore, carrying out reasonable preprocessing work on speech signals is extremely important.

Speech is a time-varying signal, and the characteristics of phonemes within it are constantly changing. The original speech signal needs to be truncated into small segments in processing speech signals. Within each segment, the speech signal remains stable. The window used to truncate the segment is called a frame, and the segments obtained by sliding the window over the speech signal are called frames. This truncation operation is called windowing.

The windowing operation also requires precise control over the frame length. Each frame should contain a complete phoneme, while microscopically, it should have enough vibration cycles to ensure faithful Fourier transform computations. The frame length should not be too long, which would result in an unstable signal with too much information, nor too short, resulting in a low-resolution Fourier analysis with insufficient information.

However, it is impossible to obtain a complete periodic function in windowing operations. In many cases, overlapping features may occur between frames, resulting in noise that directly affects the effectiveness of subsequent computations. Therefore, to emphasize the information obtained in windowing and reduce noise, a filter is added to the target waveform during windowing to weaken the content at the two ends of each frame. This is beneficial for emphasizing the main content and obtaining better results in Fourier transforms.

However, important and valid information exists in the overlapping features we eliminate. In that case, blind filtering can lead to information loss. Therefore, overlap sampling is also performed in windowing operations, which reduces the sliding distance and ensures that each part of the content can be independently and effectively analyzed.

The features of speech signals mainly reside in the frequency domain. Thus it is necessary to transform the speech signal into the frequency domain for corresponding analysis. Three main types of frequency domain transformation methods exist filter bank methods, Fourier spectrum analysis algorithms derived from Fourier transform, and related algorithms derived from Mel-frequency spectra.

Filter bank methods are one of the earliest frequency spectrum analysis methods adopted. Its use is simple and robust. The core approach is extracting corresponding frequency information by filtering the digital model through multiple filters. Fourier spectrum analysis is studied through Fourier transform and short-time analysis of signals, with the core being a short-time fast Fourier transform. Mel-frequency spectrum transformation is the latest spectrum model designed based on auditory psychological characteristics of speech. Its analysis content is more in line with human perception [15].

As speech signals have language-specific features, some corresponding feature extraction algorithms have also been developed to recognize speech effectively. Currently, mainstream speech feature extraction algorithms include Mel frequency cepstral coefficients (MFCC), linear predictive coefficients (LPC), linear predictive cepstral coefficients (LPCC), discrete wavelet transform (DWT), and others [16]. These speech feature extraction algorithms can effectively filter and recognize important features of speech, thereby accelerating speech analysis and recognition ability. However, speech feature extraction is not suitable for speech synthesis work because it needs more details, resulting in synthesized signals only exhibiting basic phonemic features but not complete speech features.

3 Model Design and Validation

3.1 Model Design Preparation

As larger batch training requires a significant amount of G-RAM, it is difficult to conduct experiments on a regular home computer. Even the entry-level Jetson Nano by NVIDIA cannot meet the experimental requirements. Therefore, cloud servers were used as the final experimental platform. The experimental machine was loaded with CUDA 1.10.0, Intel Xeon Cascade Lake (2.5 GHz), and NVIDIA T4 graphics card (8G).

The speech database used in this study is the THCHS30 database from Tsinghua University, and 5273 sets of data were used for the experiments. All speech files are in 256kbps format and have a 16000Hz sampling rate, with a source format of 16-bit integers. The audio was subjected to normalization and limit processing, with an upper limit of 20dB and a lower limit of -100 dB. At the same time, windowing was performed during the preprocessing of speech files, with a window number of 2048 and a window length of 120ms. The files were written and stored in NumPy format. The stored data includes the raw speech time domain data and the speech frequency domain data transformed by Mel-spectrogram.

The analyzed data was classified and stored during the preprocessing stage to store the experimental data properly. Firstly, train and test folders were created to store the training and testing files. Audio and mel folders were also created to store the encoded raw waveform files and their corresponding transformed spectrogram files.

In the middle of the training phase, the logdir folder was established using the parameter-saving feature in torch,

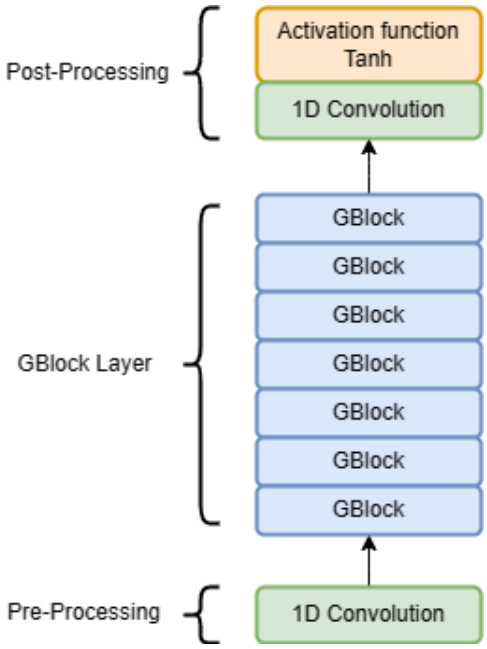

Fig. 13. Generator structure diagram

which stores the training results and related parameters after each training cycle. If the related parameters exist, the training continues to generate results. However, if the related parameters do not exist, they are re-generated.

Listing 1. File Tree

```
Main
|── Training
|    |── Data
|    |    |── Time-Domain
|    |    +── Frequency-Domain
|    +── Test
|         |── Time-Domain
|         +── Frequency-Domain
+── Log
```

3.2 GAN Modeling

3.2.1 Generator Design

This paper's generator comprises seven independent blocks and two residual convolution structures. Each independent block is named GBlock and consists of four stacks. Using multiple independent blocks effectively reduces the workload, allowing the focus to be on the design of the independent blocks rather than the overall network design. Before entering the GBlock layer, a one-dimensional convolutional layer is added to the model for basic data preprocessing to adapt it to its computational dimensions. In the post-processing stage, data is organized and corrected through a one-dimensional convolutional layer and a tanh activation function.

To adapt to the trend of continuously biased global optimization conditions in convolution, the four-layer convolution kernel is set with continuously expanding dilation factors: 1, 2, 4, and 8 to control the size of the holes, thereby obtaining an appropriate convolution kernel. The continuously expanding dilation factor allows the convolution kernel to expand continuously, thereby obtaining a larger field of view and a more global visual range. This can help capture more macroscopic data characteristics, and pay more attention to overall feature details in the later processing stage, rather than being limited to minor details, thereby avoiding pattern collapse.

TABLE 1
Generator GBlock Module Data Sheet

Module Name	Input Channel	Output Channel	Noise Channel	UpSampling Factor
GBlock1	768	768	128	1
GBlock2	768	768	128	1
GBlock3	768	384	128	2
GBlock4	384	384	128	2
GBlock5	384	384	128	2
GBlock6	384	192	128	3
GBlock7	192	96	128	5

Each GBlock in this study was designed with four similar stacks, augmented with a two-level residual network and batch normalization operations. Each stack consists of a ReLU activation function followed by a one-dimensional convolution. In particular, the first stack contains an upsampling module between the activation function and convolution, which corrects the issue of the output frequency being smaller than the input frequency. The generator convolutions mainly used dilated convolutions to increase the receptive field of the convolution kernel by filling in the gaps between convolution kernels, maximizing the retention of details, and avoiding the loss of edge features. Additionally, a two-level residual network was incorporated into the design to ensure no excessive gradient disappearance problem during the later training process. Each stack underwent batch normalization operations to adjust the input data and help the neural network learn the rules in the data, preventing undesirable data from interfering with the training of the activation function and convolution module.

3.2.2 Discriminator

The discriminator in this work was designed using the SN-GAN model [18]. Each convolutional layer consists of a one-dimensional layer based on mel-spectrogram analysis and a leaky activation function.

Unlike the generator, the model does not use convolutional layers to prepare the data; instead, it simply flattens the data in one dimension to preserve the maximum data details. Then, seven independent DBlocks were stacked to improve the efficiency of the development process. The model adjusts the weight function with a normalization factor α to bias the optimization towards more stable situations. The default value for the activation function α is set to 0.2. The convolutional kernel parameters were set with corresponding dilation factors to expand their visual fields continuously. They tended to train on global features in the later stages to prevent mode collapse.

3.3 Optimizer & Loss Function Design

The generator and discriminator employ the popular Adam optimizer with a small default learning rate to ensure

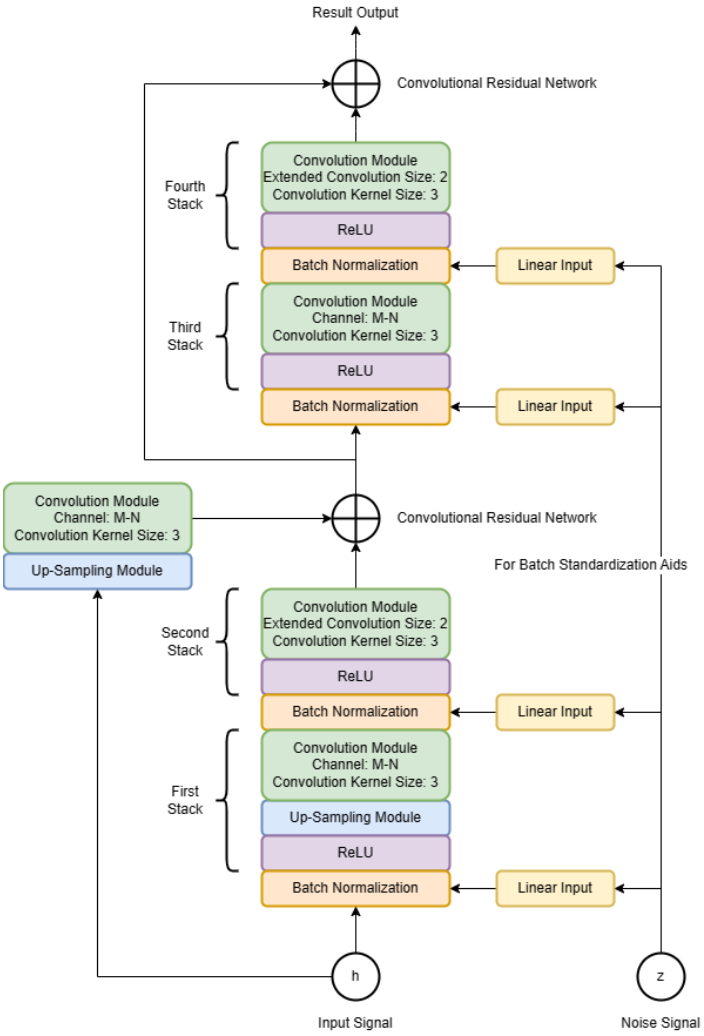

Fig. 14. Structure of GBlock [17]

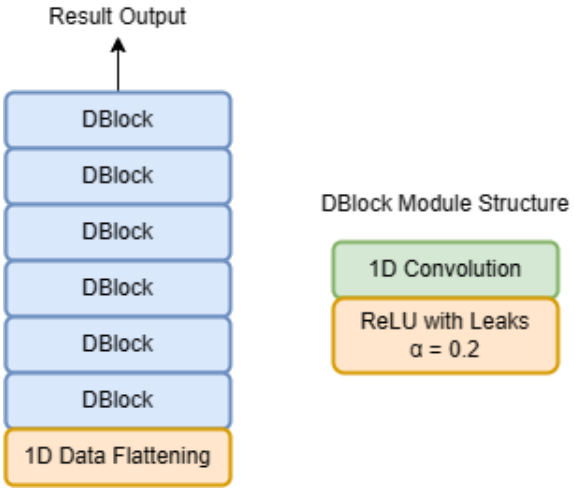

Fig. 15. Discriminator and DBlock Structures

TABLE 2
Discriminator and DBlock Data Sheet

Module Name	Input Channel	Output Channel	Convolution Kernel	Stride	Padding	Group
Conv1d-1	1	16	15	1	/	1
Conv1d-1	16	64	41	4	20	4
Conv1d-1	64	256	41	4	20	16
Conv1d-4	256	1024	41	4	20	64
Conv1d-5	1024	1024	41	4	20	256
Conv1d-6	1024	1024	5	1	2	1
Conv1d-7	1024	1	3	1	1	1

reasonable parameter adjustment and prevent oscillation [19]. The loss function used for the first two training steps of the discriminator is MSELoss, and the final backward calculation loss is the sum of the loss values obtained from real and fake data.

$$Loss_D = Loss_{RealData} + Loss_{FakeData} \tag{24}$$

The loss design for the generator is more complex. Two different frequency-based loss functions are used to calculate the generator's loss. The design incorporates both GAN ideas and the idea of comparing generated data directly with original data to obtain the loss:

1) The discriminator calculates the loss function, $Loss_D$, based on the generated data from the generator.
2) The generated data and original data are compared from a frequency domain perspective. One method involves calculating the spectral norm, while the other uses the mean absolute error loss function to calculate two log-compressed values.
3) The generator's loss function, $Loss_G$, is calculated by combining the discriminator optimization parameter (λ_{adv}) that is set. In this experiment, (λ_{adv}) is set to 4 by default.

$$Loss_{SpectralConvergence} = \frac{||message_{target} - message_{generate}||}{||message_{target}||} \tag{25}$$

$$Loss_{LogSTFTMagnitude} = \frac{\sum_{i=1}^{n} |\log(message_{target}) - \log(message_{generate})|}{n} \tag{26}$$

$$Loss_G = Loss_D \times lambda_{adv} + Loss_{SC} + Loss_{Mag} \tag{27}$$

3.4 Fine-Tune & Final Design

One of the key considerations during training is the choice of batch size. The batch size directly affects the optimization direction and the strength of the optimization. A too-small batch can result in a wrong direction or insufficient optimization strength. At the same time, a batch that is too

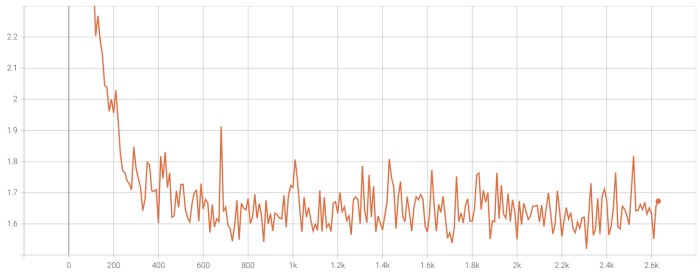

Fig. 16. batch=8 g loss graph

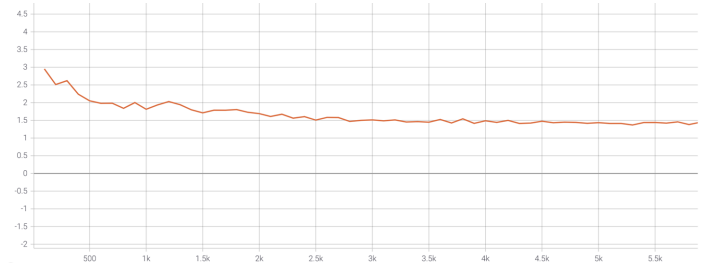

Fig. 17. batch=30 g loss graph

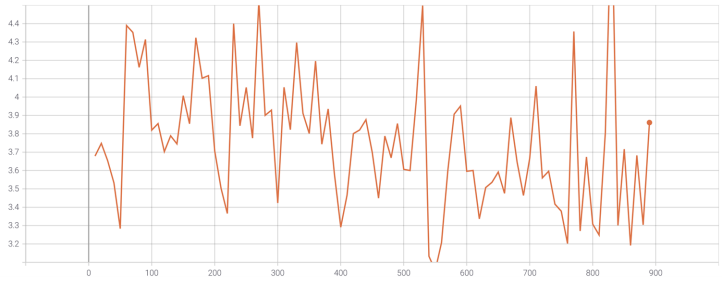

Fig. 18. Introduce discriminator at 50 steps g loss graph

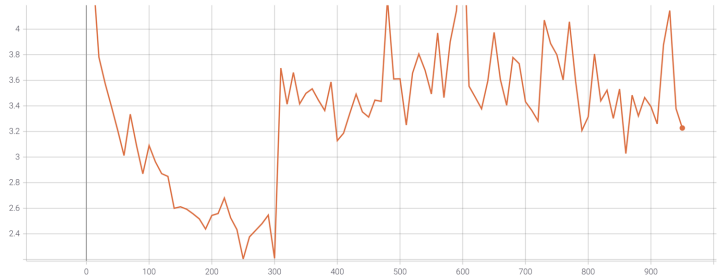

Fig. 19. Introduce discriminator at 200 steps g loss graph

large can lead to high memory requirements or insufficient optimization strength. In the early model design stage, batch sizes below ten were used. Tests have shown that a too-small batch can only exacerbate convergence oscillations or even affect normal convergence. Based on the test results, a batch size of 30 can still achieve good convergence. However, it requires a longer training period to manifest, and the optimization effect is also very significant. For example, during the initial warm-up training of 20,000 steps, if a batch size of 8 is used, the loss can only converge to around 1.6 with significant oscillations. The convergence trend could be clearer and easier to continue. However, with a batch size of 30, the loss can converge significantly to around 1.5 with a clear convergence process. Batch sizes above 15 also exhibit good convergence properties. From the perspective of computation time, the computation time for batch sizes before 20 is stable at around 0.5 seconds per step, while the computation time for a batch size of 30 increases significantly to around 1 second per step. This test result indicates that the computational power is far beyond the batch size requirements, and the main problem lies in the size of memory. Considering the convergence requirements, training time, and computation cost, the final batch size is 15. This choice considers the overall convergence requirements, training time, and computation cost.

Secondly, due to the rapid growth of the discriminator, it is necessary to perform pre-warm-up training for the generator to achieve a better adversarial balance. If the discriminator is added too early with enough pre-training time, it will result in better convergence. Therefore, a long pre-training period is necessary before the training process to enable the generator to counter the discriminator effectively. Testing data reveals that when the batch size is eight and the discriminator is added after 50 steps, the discriminator grows too fast, causing the generator to crash. The generator loss exhibits severe oscillations with some convergence tendency. However, when the discriminator is added after a delay, significant convergence is observed in the early stages, and rapid convergence is observed later. Data from adding the discriminator after 200 steps shows a clear convergence tendency in the early stages. The amplitude of oscillations starts to converge after the discriminator is added. However, significant oscillations still occur later, though there is a clear overall convergence trend. This suggests that the overall pre-training time still needs to be improved, and more pre-training is needed to prepare for subsequent training.

The final chosen configuration involves setting the batch size to 15 and setting the discriminator's learning rate and the generator's 0.0001. The generator is added after 20,000 steps, and the lambda value is set to 4. The training process involves 300,000 steps. Throughout the training process, adversarial conditions are monitored to prevent training collapses.

4 Results & Effect Analysis

4.1 Loss Function Analysis

From the loss of the discriminator, we can observe that the system is still in a state of balanced opposition after introducing the discriminator. Both the loss value of the synthetic data and the real data exhibit some oscillation, but it is a relative oscillation. Therefore, when combining the d loss value, a good convergence effect can be achieved towards 0.5. Both data sets from the discriminator converge to 0.25, indicating an ideal performance according to the expected mean squared error loss function. This suggests that the overall adversarial conditions are well-balanced, demonstrating that the model still possesses strong vitality and stability. The adv loss serves as an overall evaluation of the discriminator, which remains very low and shows minimal fluctuations after introducing the discriminator. This indicates that its corrective ability towards the generator

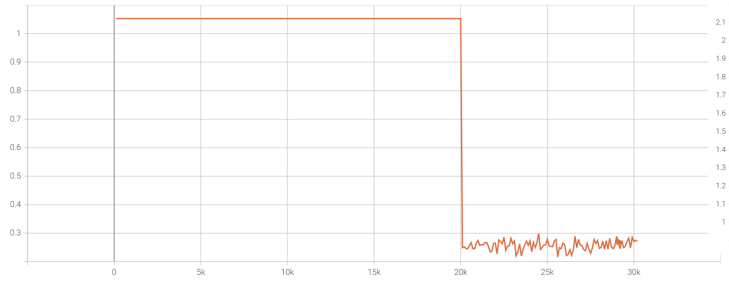

Fig. 20. adv loss graph

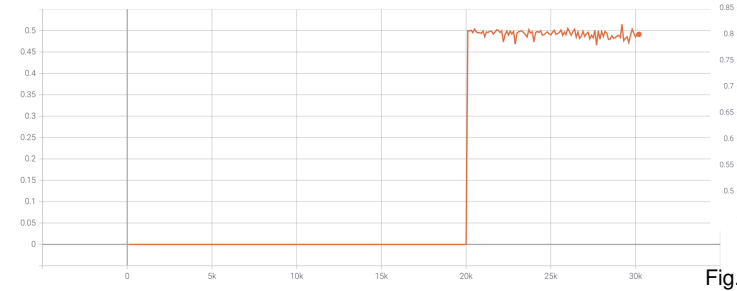

Fig. 21. d loss graph

has remained stable and reliable, continuously providing optimization power for the generator. Overall, the discriminator's situation demonstrates the GAN model's feasibility and reliability.

The generator maintained a good convergence trend in the early preheating training. It continued to perform well after the discriminator was introduced. The g loss rebounded to a high level with the introduction of the mid-term GAN framework but still maintained a state of convergence. Both mag loss and sc loss showed a stable convergent trend, with their values converging to around

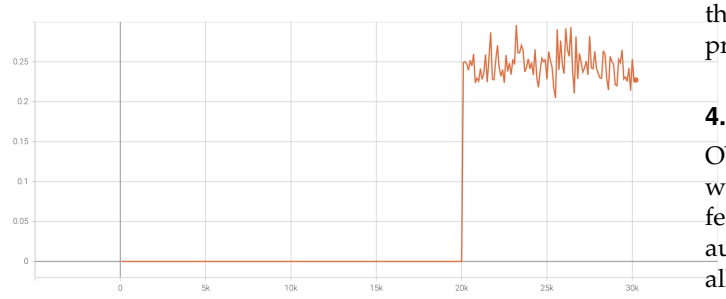

Fig. 22. fake loss graph

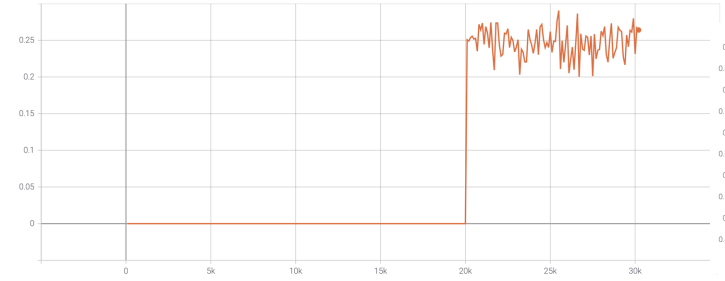

Fig. 23. real loss graph

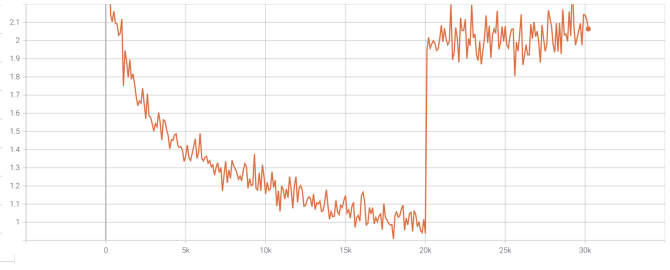

Fig. 24. g loss graph

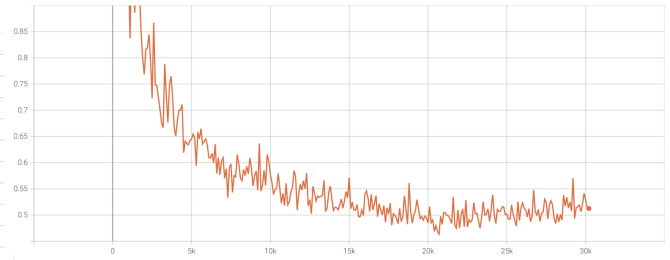

Fig. 25. mag loss graph

0.5, consistent with the expected stable loss functions.

The generator exhibited a very stable convergence trend in the early stage. The high-batch state provided sufficient preheating preparation for the overall convergent training. After the discriminator was introduced in the later stages of training, the generator's loss function exhibited noticeable oscillations but maintained a very stable convergence state. After introducing the discriminator, the mag loss remained stable at around 0.5 and generated a small, stable oscillation nearby. The sc loss showed larger oscillations but continued the pre-training convergence trend and remained very clearly convergent. The generator's loss demonstrates that the GAN model still has strong vitality in audio processing and can effectively promote model convergence, accelerate the overall speed and situation of convergence, and thus produce good synthesized speech.

4.2 Audio Generation Quality Analysis

Objectively, the generated audio has restored the native waveform in the time domain and has corresponding audio features in the corresponding area. The amplitude of the audio has been weakened, but it does not affect the overall waveform similarity. The generated audio has restored the semantic-related features in the frequency domain and coincides with the original speech spectrum. However, in

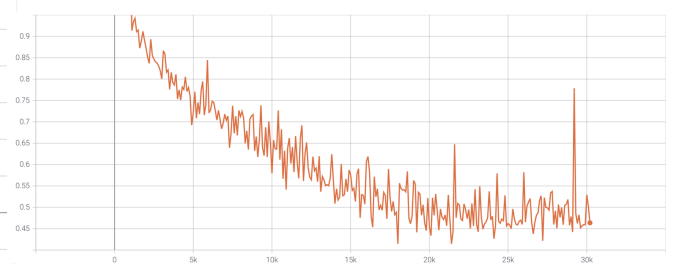

Fig. 26. sc loss graph

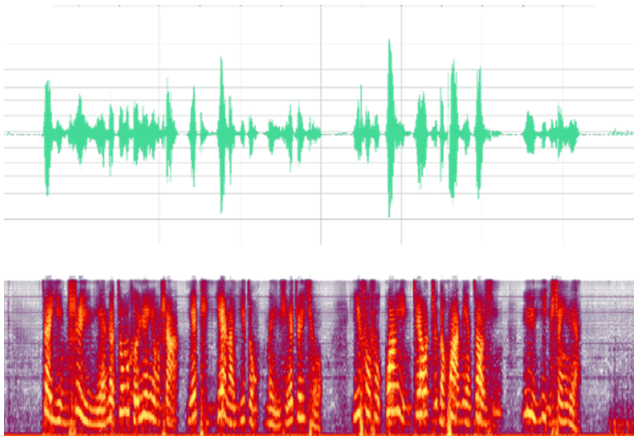

Fig. 27. Generated Spectrum

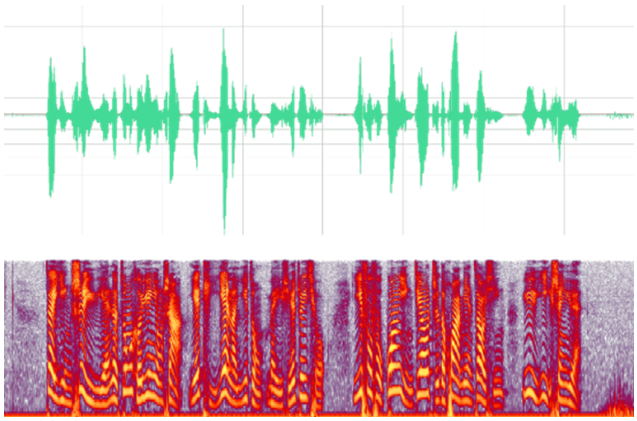

Fig. 28. Real Spectrum

the entire frequency domain, especially in the low-frequency part, a lot of noise information still needs to be eliminated, which affects the specific audio processing situation. Overall, the relevant features of speech have been very well restored, but the noise part of the details still needs to be trained more deeply. From a subjective perspective, the generated data spectrum has well simulated the speech and semantic content in the original file. The human ear can recognize more obvious timbre features and language content. However, the clarity features still need to be better restored, and there is still much noise.

5 Conclusion

As a new neural network deep learning model, Generative Adversarial Networks (GAN) have great potential for mining in neural network training. Currently, in China, training of GAN models is still in its early stages, and there is still much space for exploration waiting for researchers to explore. GAN also has tremendous potential in speech synthesis. Its core nature of adversarial unsupervised learning can help speed up the overall model convergence and achieve very good generation results. In addition, its tolerance for speech sets can greatly reduce the related work of speech pre-processing, making it easier to train a complete speech generation model. From a practical point of view, attention should be paid to pre-training and batch-related situations for GAN speech models. GAN speech models can achieve good generation results only in situations with sufficient pre-training and large enough batch sizes.

From a macro perspective, the design and development of GAN in speech synthesis still need improvement, and there is still much room for expansion and development in the existing models. The optimization and improvement consist of two aspects: the optimization of the audio preprocessing stage, how to extract feature operations more concisely using a superior performance data set with higher effective information and less data redundancy while not losing any useful information. The other aspect is the corresponding optimization of the training model. Designing more efficient audio models that facilitate overall training speed and efficiency improvement is essential. The digital signal processing and GAN research fields need efforts to collaborate and design better models.

References

[1] 张肇聿, 王一琳, and 李志, "基于人工智能技术的 25 个行业发展趋势," 无人系统技术, vol. 1, 2019.

[2] 杨大为, "生成式对抗网络 gan 及应用," 信息系统工程, no. 6, pp. 81–82, 2018.

[3] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[4] 魏伟华, "语音合成技术综述及研究现状," 软件, vol. 41, no. 12, pp. 214–217, 2020.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[6] 杜立平, 宋燕红, 李晓东, and 金鑫, "Gan 在 ai 领域的应用研究," 北京电子科技学院学报, vol. 3, 2018.

[7] 姚勤炜, 陈华杰, 张杰豪, and 侯新雨, "基于反馈调节的生成对抗网络训练方法," 信号处理在地球物理——浙江省信号处理学会 *2018* 年学术年会论文集, 2018.

[8] 邹秀芳 and 朱定局, "生成对抗网络研究综述," 计算机系统应用, vol. 28, no. 11, pp. 1–9, 2019.

[9] 陈鑫, "基于卷积神经网络的脉搏纹时效性分析," Master's thesis, 北京邮电大学, 2019.

[10] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[11] 王红霞, 周家奇, 辜承昊, and 林泓, "用于图像分类的卷积神经网络中激活函数的设计," 浙江大学学报 (工学版), vol. 53, no. 7, pp. 1363–1373, 2019.

[12] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

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

[14] P. Hinz, "The layer-wise l1 loss landscape of neural nets is more complex around local minima," *arXiv preprint arXiv:2105.02831*, 2021.

[15] 张小峰, 谢钧, 罗健欣, and 俞璐, "深度学习语音合成技术研究," 计算机时代, no. 9, pp. 24–28, 2020.

[16] 沈泉波, "相关向量机在语音识别中的应用研究," Ph.D. dissertation, 太原: 中北大学, 2013, 2013.

[17] M. Bińkowski, J. Donahue, S. Dieleman, A. Clark, E. Elsen, N. Casagrande, L. C. Cobo, and K. Simonyan, "High fidelity speech synthesis with adversarial networks," *arXiv preprint arXiv:1909.11646*, 2019.

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

[19] 杜一铭, "基于结构改进的深度卷积生成对抗网络研究及应用," Master's thesis, 重庆邮电大学, 2019.