

CHAPTER 2

GAN models in natural language processing and image translation

E. Thirumagal^{a,b} and K. Saruladha^a

^aPondicherry Engineering College, Department of Computer Science and Engineering, Puducherry, India

^bREVA University, Bengaluru, India

2.1 Introduction

In recent years, GANs have shown significant progress in modeling image and speech complex data distributions. The introduction of GAN and VAE made training big datasets in an unsupervised manner possible.

2.1.1 Variational auto encoders

The variational auto encoders (VAEs) [1] were used for generating images before GANs. The VAE has a probabilistic encoder and probabilistic decoder. The real samples “ r ” are fed into the encoder. The encoder outputs an encoded image, with which the noise “ n ” is added whose distribution is given by $X_e(n|r)$. The distribution $X_e(n|r)$ is given as input to decoder whose distribution is given by $Y_d(r|n)$ which will generate the fake image “ r .” The loss function $L(e, d)$ between encoder and decoder is computed for every iteration. The VAE uses a mean square loss function, which is given by

$$L(e, d) = E_{n \sim X_e(n|r)} [Y_d(r|n)] + KLD(X_e(n|r) || Y_d(n)) \quad (2.1)$$

where

$$KLD(X_e(n|r) || Y_d(n)) = \sum X_e(n|r) \log \left(\frac{X_e(n|r)}{Y_d(n)} \right)$$

The Kullback-Leibler divergence (KLD) is the distance metric that computes the similarity between the real sample given to the encoder X_e and the generated fake image from decoder Y_d . If the loss function yields more value, it means the decoder does not generate fake images similar to the real samples. The backpropagation will take place for every iteration until the decoder generates the image similar to the real image. By using stochastic gradient descent, the weights and bias of the encoder and decoder will be adjusted and again image generation will happen. The optimal value of the loss function is 0.5.

When the loss function of the decoder becomes 0.5, it means the decoder generates the image similar to the real image.

2.1.1.1 Drawback of VAE

VAE uses Kullback-Leibler divergence (KLD). When the generated image distribution $Y_d(n)$ does not match the real image distribution $X_e(n|r)$, then $Y_d(n)$ value will become 0. The KLD will lead to ∞ (infinity), which means learning will not take place for the encoder and decoder. This leads to the invention of GANs.

2.1.2 Brief introduction to GAN

Generative adversarial networks (GANs) [2–4] are generative neural network models introduced by Ian Goodfellow in 2014. Recently, GANs have been used in numerous applications such as discovery and prevention of security attacks, clothing translation, text-to-image conversion, photo blending, video games, etc. GANs have generator (G) and discriminator (D) which can be convolutional neural network (CNN), feed forward neural networks, or recurrent neural networks (RNNs). The generator (G) will generate fake images, by taking random noise distribution as input. The real samples and the generated fake images are given as input to the discriminator (D), which will output whether the image is from the real sample or from the generator (1 or 0). The loss functions are computed to check whether (i) the generator is generating images close to real samples and (ii) the discriminator is correctly discriminating between real and fake images. If the loss function yields a big value, then backpropagate to *Generator* and *Discriminator neural networks*, adjust its weights and bias which is called as optimization. There are various optimization algorithms such as stochastic gradient descent, RMSProp, Adam, AdaGrad, etc. Therefore, both G and D learn simultaneously.

This chapter is organized as follows: The various GAN architectures are discussed in Section 2.2. Section 2.3 describes the applications of GANs in natural language processing. The applications of GANs in image generation and translation are discussed in Section 2.4. The Section 2.5 discusses the evaluation metrics that can be used for checking the performance of the GAN. The tools and the languages used for GAN research are discussed in Section 2.6. The open challenges for further research are discussed in Section 2.7.

2.2 Basic GAN model classification based on learning

From the literature survey made, the classification of various GANs is made based on the learning methods as shown in Fig. 2.1. The learning can be supervised or unsupervised. Supervised learning is making the machine learning with the labeled data. The classification and regression algorithms come under supervised learning. The unsupervised learning [5] will take place when the data is unlabeled. The machine will act on the data

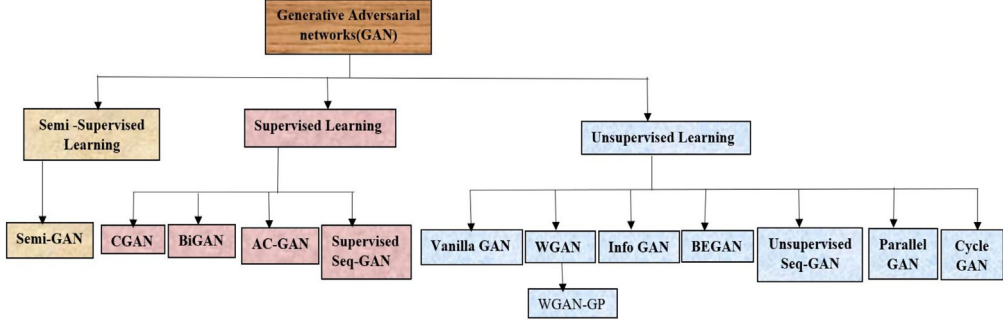


Fig. 2.1 GAN architecture classification.

based on the similarities, differences, and patterns. The clustering and association algorithms come under unsupervised learning.

2.2.1 Unsupervised learning

The generative models before GANs used the Markov chain [6] method for training which has various drawbacks such as high computational complexity, low efficiency, etc. As shown in Fig. 2.1, vanilla GAN, WGAN, WGAN-GP, Info GAN, BEGAN, Unsupervised Sequential GAN, Parallel GAN, and Cycle GAN are categorized under unsupervised learning [7]. The above said GANs will take the data or the real samples without labels as input. The architectures of all the above GANs are shown in the following sections. This section details about each GAN architecture with its loss functions and optimization techniques.

2.2.1.1 Vanilla GAN

The Vanilla GAN [8, 9] is the basic GAN architecture. The real samples are given by “ r .” The random noise is given by “ n .” The random noise distribution $p_n(n)$ is given as input to the G which will generate the fake image. The real sample distribution $p_d(r)$ and fake images are given as input to D . D will discriminate whether the image is real (which means 1) or fake (which means 0) which is shown in Fig. 2.2. Then by using the binary cross entropy loss function, the loss of G and D will be calculated by Eqs. (2.3) and (2.4). Binary cross-entropy loss function (Goodfellow [2]) is given by Eq. (2.2).

$$L(x', x) = x \log x' + (1 - x) \log (1 - x') \quad (2.2)$$

where x' is the generated fake image and x is the real image.

When the image is coming from real sample “ r ” to D , then D has to output 1. So, substitute $x' = D(r)$ and $x = 1$ in Eq. (2.2) will lead to the following equation:

$$L_{\text{GAN}}(D) = E_{r \sim p_d(r)} [\log (D(r))] \quad (2.3)$$

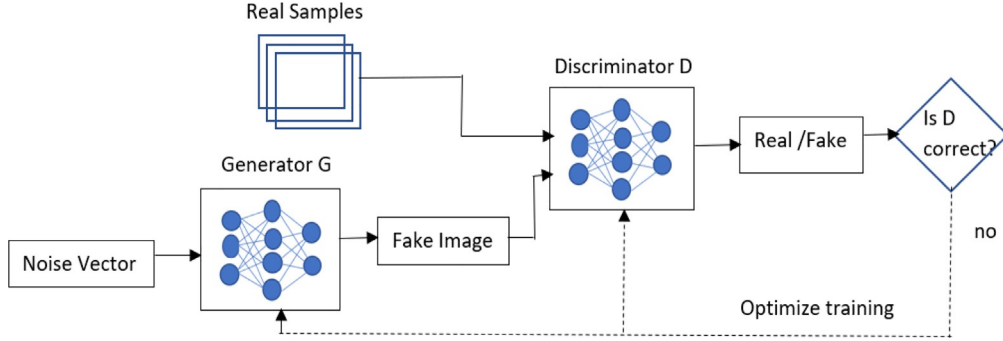


Fig. 2.2 Vanilla GAN, WGAN, WGAN-GP architecture.

When the image is coming from G , $G(n)$ to D , then D has to output 0. So substitute $x' = D(G(n))$ and $x = 0$ in Eq. (2.2) will lead to the following equation:

$$L_{\text{GAN}}(G) = E_{n \sim p_{n(n)}} [\log(1 - D(G(n)))] \quad (2.4)$$

Using min-max game theory, the D has to output 1 if the image is the real sample. Hence the D has to be maximized. The D has to output 0 if the image has come from the generator. Hence the G has to be minimized. The loss function is given by

$$\min_G \max_D L_{\text{GAN}}(G, D) = \min_G \max_D \left\{ E_{r \sim p_d(r)} [\log(D(r))] + E_{n \sim p_{n(n)}} [\log(1 - D(G(n)))] \right\} \quad (2.5)$$

The optimal value of D is given by

$$D^* = \frac{p_d(r)}{p_d(r) + p_g(r)} \quad (2.6)$$

If the optimal value of D (0.5) is obtained, then D cannot differentiate between real and fake images. The optimal value of G is given by

$$G^* = -\log 4 + 2 \times \text{JSD}(p_d(r) \| p_g(r)) \quad (2.7)$$

where

$$\text{JSD}(p_d(r) \| p_g(r)) = \frac{1}{2} \{ \text{KLD}(p_d(r) \| (p_d + p_g)) + \text{KLD}(p_g(r) \| (p_d + p_g)) \}$$

If the loss function yields more value than by using backpropagation and stochastic gradient descent [6], weights and bias will be adjusted for every epoch until D discriminates properly.


2.2.1.2 WGAN

The WGAN [10] stands for Wasserstein GAN. The architecture is the same as that of Vanilla GAN as shown in Fig. 2.2. In order to avoid the drawback of using Jensen Shannon Divergence distance, the Wasserstein distance metric has been used. JSD matches the real image and fake image distributions in the vertical axis. Whereas the Wasserstein distance matches the real image and fake image distributions in the horizontal axis. The Wasserstein distance is otherwise called as earth mover distance.

The Wasserstein distance between the real image distribution “ P_r ” and the generated image distribution “ P_g ” is given by

$$W(P_r, P_g) = \inf_{\delta \in \pi(P_r, P_g)} E_{(a, b) \sim \delta} [|x - y|] \quad (2.8)$$

Π is the transport plan which tells how the distribution changes from real and generated image.

																							
Assume the block 1 moving from 11 to 12 and block 2 is moving from 14 to 13. $\Pi 1$ (Transport Plan 1)	Assume the block 1 moving from 11 to 13 and block 2 is moving from 14 to 12. $\Pi 2$ (Transport Plan 2)																						
<table><tr><td></td><td>12</td><td>13</td><td></td></tr><tr><td>11</td><td>1</td><td>0</td><td rowspan="2">1+1 = 2</td></tr><tr><td>14</td><td>0</td><td>1</td></tr></table>		12	13		11	1	0	1+1 = 2	14	0	1	<table><tr><td></td><td>12</td><td>13</td><td></td></tr><tr><td>11</td><td>0</td><td>1</td><td rowspan="2">2+2 = 4</td></tr><tr><td>14</td><td>1</td><td>0</td></tr></table>		12	13		11	0	1	2+2 = 4	14	1	0
	12	13																					
11	1	0	1+1 = 2																				
14	0	1																					
	12	13																					
11	0	1	2+2 = 4																				
14	1	0																					
Whichever the transport plan yeilds less distance, it will be taken as Wasserstein distane.																							

Eq. (2.8) is intractable. To make it tractable, using Kantorovich–Rubinstein duality the W -distance is given by

$$W(P_r, P_g) = \sup_{\|D\|_{L \leq 1}} E[D(r) - D(G_g(n))] \quad (2.9)$$

When taking the slope between real and fake image distributions, if the slope is less than or equal to k , it is called as k -Lipchitz constant. When k is 1 it is 1-Lipchitz constant. Wasserstein GAN uses 1-Lipchitz constant and to achieve it, the weights will be clipped in the range $(-1, 1)$. The discriminator loss function is given by

$$L_{\text{WGAN}}(D) = E_{r \sim p_{d(r)}} [D(r)] \quad (2.10)$$

The generator loss function is given by

$$L_{\text{WGAN}}(G) = E_{n \sim p_{n(n)}}[D(G(n))] \quad (2.11)$$

The overall parametric loss function is given by

$$L_{\text{WGAN}}(G, D) = \min_G \left\{ \max_{w \in W} E_{r \sim p_{d(r)}}[D(r)] - E_{n \sim p_{n(n)}}[D(G(n))] \right\} \quad (2.12)$$

In WGAN, the discriminator will not return 0 or 1 rather it will return the Wasserstein distance. WGAN uses RMSProp optimizer which alters the weights and bias of G and D for every iteration until D cannot discriminate between real and fake images.

2.2.1.3 WGAN-GP

The WGAN-GP [11, 12] (Wasserstein GAN-Gradient Penalty) architecture is identical to Vanilla GAN and WGAN as shown in Fig. 2.2. To avoid the drawback of weight clipping in order to use the Lipchitz constant, the gradient penalty term is incorporated with WGAN loss function. The gradient penalty term is computed when the gradient norm value moves away from 1. The loss function of WGAN-GP is given by

$$L_{\text{WGAN}}(G, D) = \min_G \left\{ \max_{w \in W} E_{r \sim p_{d(r)}}[D(r)] - E_{n \sim p_{n(n)}}[D(G(n))] \right\} + \lambda E_{x \sim P(x)} \left[\left(\|\nabla_x D(x)\| - 1 \right)^2 \right] \quad (2.13)$$

The gradient penalty term is included with the loss function of WGAN. In Eq. (2.13), x -sampled from noise “ n ” and real image “ r ” is given by, $x = t \times x + (1 - t)x$, where t is sampled between 0 and 1. λ -hyperparameter. The adam optimizer, if used with WGAN-GP, it generates good clear images.

2.2.1.4 Info GAN

Info GAN [8, 13] is the information maximizing GAN. The semantic information is added with noise and given to the G . G outputs the fake image. The fake image that is generated and the real sample are given to D which will output 0 (fake) or 1 (real) shown in Fig. 2.3. Then the loss function is computed. The stochastic gradient descent is used for optimizing the neural network. The noise “ n ” with semantic information “ si ” is fed to G which is given by $G(n, si)$. The mutual information $MI(si; G(z, n))$ has to be maximized between semantic information “ si ” and generator $G(n, si)$.

The mutual information $MI(c; G(z, n))$ is the amount of information obtained from knowledge of $G(z, n)$ about semantic information si . Maximizing mutual information is not very easy, so variational lower bound of mutual information $MI(si; G(z, n))$ by

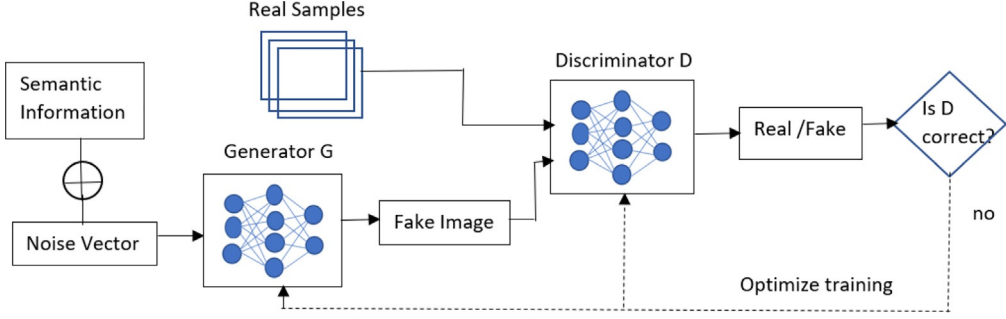


Fig. 2.3 Info GAN architecture.

defining one more semantic distribution $Q(si|r)$. The variational lower bound $LB(G, Q)$ of mutual information is given by

$$LB(G, Q) = E_{si \sim P(si), x \sim G(n, si)} [\log Q(si|r)] + H(si) \leq I(si; G(z, n)) \quad (2.14)$$

where $H(si)$ is the entropy of latent codes. The loss function is given by

$$\begin{aligned} \min_{G, Q} \max_D L_{\text{InfoGAN}}(G, D) = & E_{r \sim p_d(r)} [\log(D(r))] \\ & + E_{n \sim p_n(n)} [\log(1 - D(G(n)))] - \lambda \times LB(G, Q) \end{aligned} \quad (2.15)$$

Wake Sleep algorithm has been used with InfoGAN. The lower bound of the generator $\log PG(x)$ has been optimized and updated in the wake phase. The auxiliary distribution Q is updated in the sleep phase by up sampling from generator distribution instead of real data distribution. The cost is only a little more than the vanilla GAN.

2.2.1.5 BEGAN

BEGAN [14, 15] stands for Boundary Equilibrium GAN. This BEGAN is mainly developed to achieve Nash equilibrium. The architecture is the same as that of vanilla GAN with one difference: for maintaining equilibrium, the proportional control theory has been used as shown in Fig. 2.4. In BEGAN generator acts as a decoder and the discriminator acts as autoencoder and also discriminates between real and fake images. In BEGAN, instead of matching the data distributions of real image and generated image, the autoencoder loss has been calculated for real image and generated image. The Wasserstein distance has been computed between the autoencoder loss of real and generated images. The autoencoder loss is given by

$$L(s) = |s - AF(s)|^\eta \quad (2.16)$$

where $L(s)$ is a loss for training autoencoder. “ s ” is the sample of dimension “ d ,” AF is the autoencoder function which converts the sample of dimension “ d ” to sample of dimension “ d ,” η is the target norm takes value $\{1, 2\}$.

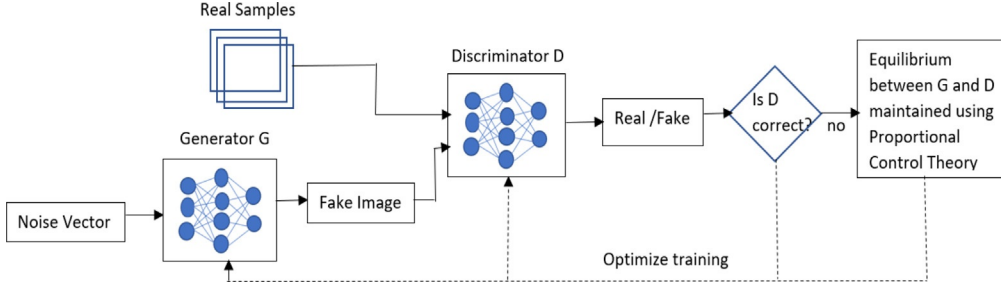


Fig. 2.4 BEGAN architecture.

The loss of the D is given by

$$L_D = L(r) - k_i \times L(G(n_D)) \quad (2.17)$$

The loss of the G is given by

$$L_G = L(G(n_G)) \quad (2.18)$$

BEGAN makes use of the proportional control model to preserve equilibrium $E[L - (G(n))] = \gamma \times E[L(r)]$ where γ is the hyperparameter which takes the value $(0, 1)$. For maintaining equilibrium, it uses the variable k_i which takes a value $(0, 1)$ to control the generator loss during gradient descent. Where k_i is given by

$$k_{i+1} = k_i + \lambda_k (\gamma \times L(r) - L(G(n_G))) \quad (2.19)$$

Initially take $k_0 = 0$. λ_k is the learning rate of k .

2.2.1.6 Unsupervised sequential GAN

The Sequential GAN [16, 17] involves a sequence of G and D . The noise vector “ z ” is given as input to Generator G_1 . The G_1 produces fake image1 “ f ” as the output. The fake image1 and real sample “ r ” is given as input to discriminator D_1 which will discriminate between real image1 and fake image1. The fake image1 is given as input to generator G_2 . The G_2 produces fake image2 as the output. The fake image2 and the real image2 is given as input to discriminator D_2 which will discriminate between the real and fake image as shown in Fig. 2.5. The loss function by considering G_1 and D_1 is given by

$$L_{adv}(G_1, D_1, n, r) = E_{r \sim p_{d(r)}} [\log(D_1(r))] + E_{n \sim p_{n(n)}} [\log(1 - D_1(G_1(n)))] \quad (2.20)$$

The loss function by considering G_2 and D_2 is given by

$$L_{img2img}(G_2, D_2, f, r) = E_{r \sim p_{d(r)}} [\log(D_2(r))] + E_{f \sim p_{f(f)}} [\log(1 - D_2(G_2(f)))] \quad (2.21)$$

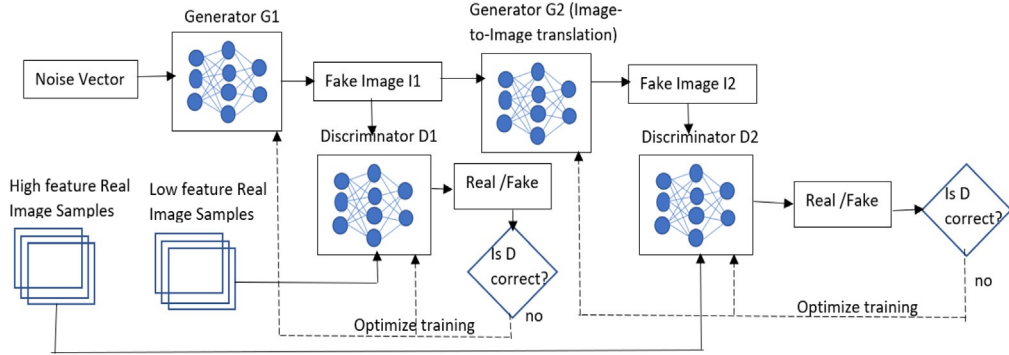


Fig. 2.5 Unsupervised sequential GAN architecture.

The loss function of unsupervised sequential GAN is given by

$$L_{\text{unseqGAN}}(G1, D1, G2, D2) = L_{\text{adv}}(G1, D1, n, r) + L_{\text{img2img}}(G2, D2, f, r) \quad (2.22)$$

2.2.1.7 Parallel GAN

The architecture of the Parallel GAN [18] is shown in Fig. 2.6. Whenever there are bimodal images to be processed or when there is a need to generate multiple images at the same time then the parallel GAN can be used. Noise vector will be given to generator G1 and G2 parallelly. The G1 and G2 generate fake image1 and fake image2 parallelly. The real image1 and fake image1 are given as input to input to discriminator D1 which will discriminate between real image1 and fake image1. The real image2 and fake image2 are given as input to input to discriminator D2 which will discriminate between

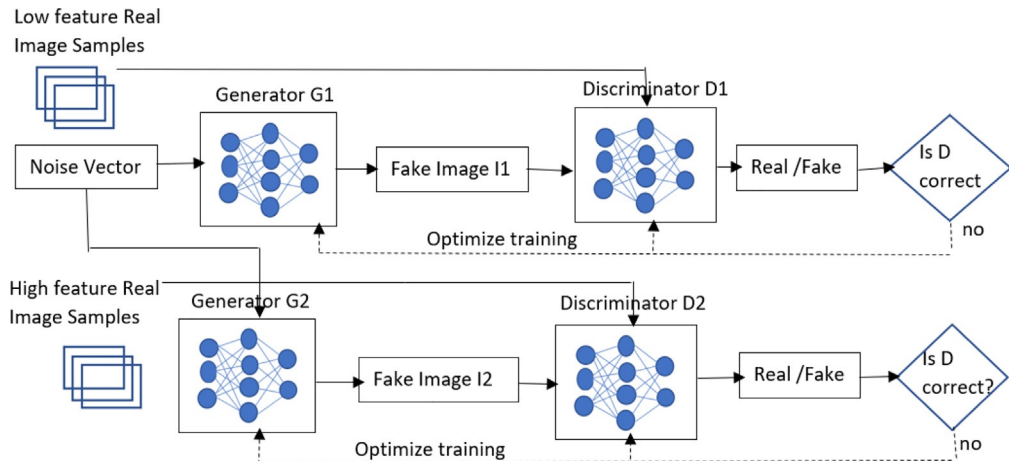


Fig. 2.6 Parallel GAN architecture.

real image2 and fake image2 parallelly with $D1$. The binary cross-entropy loss is computed for $(G1, D1)$ and $(G2, D2)$ parallelly. If $D1$ and $D2$ are not discriminating between real and fake images properly, then by using back propagation and stochastic gradient $(G1, D1)$ and $(G2, D2)$ weights and bias will be adjusted for every iteration until the $D1$ and $D2$ discriminate correctly.

2.2.1.8 Cycle GAN

The Cycle GAN is otherwise called cycle-consistent GAN [19, 20]. The noise vector “ z ” is given as input to generator $G1$. The $G1$ produces feature map “ f ” as the output. The feature map and real sample “ r ” is given as input to discriminator $D1$ which will discriminate between the real sample and feature map. The feature map is given as input to generator $G2$. The $G2$ produces a fake image as the output. The fake image and the real image are given as input to discriminator $D2$ which will discriminate between the real and fake image as shown in Fig. 2.7. The loss function by considering $G1$ and $D1$ is given by

$$L1(G1, D1, n, r) = E_{r \sim p_{d(r)}} [\log(D1(r))] + E_{n \sim p_{n(n)}} [\log(1 - D1(G1(n)))] \quad (2.23)$$

The loss function by considering $G2$ and $D2$ is given by

$$L2(G2, D2, f, r) = E_{r \sim p_{d(r)}} [\log(D2(r))] + E_{f \sim p_{f(f)}} [\log(1 - D2(G2(f)))] \quad (2.24)$$

The cycle consistency loss is given by

$$L_{\text{cycle}}(G1, G2) = E_{n \sim p_{n(n)}} [\|G2(G1(n)) - n\|_1] + E_{f \sim p_{f(f)}} [\|G1(G2(f)) - f\|_1] \quad (2.25)$$

The Cycle GAN loss is given by

$$L_{\text{cycleGAN}}(G1, G2, D1, D2) = L1(G1, D1, n, r) + L2(G2, D2, f, r) + L_{\text{cycle}}(G1, G2) \quad (2.26)$$

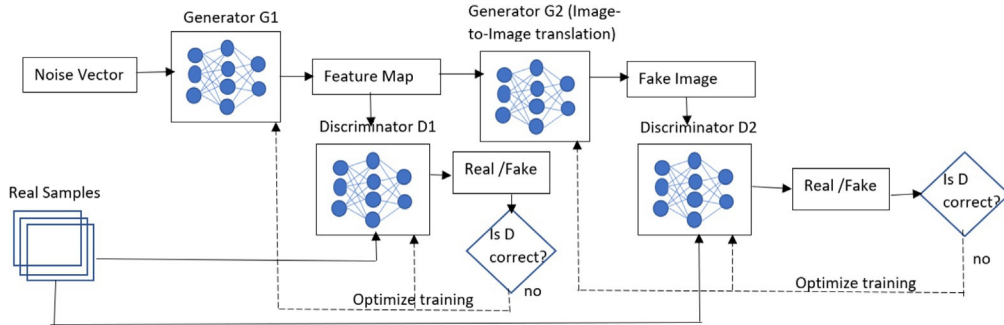


Fig. 2.7 Cycle GAN architecture.

2.2.2 Semisupervised learning

Semisupervised learning is that the discriminator D will be trained by class labels, i.e., D will do supervised learning. The generator will not be trained with the class labels hence the learning will be unsupervised. The Semi GAN comes under this semisupervised learning category which is discussed in the following section.

2.2.2.1 Semi GAN

The architecture of semi GAN [21, 22] is shown in Fig. 2.8. The class labels are added with the real samples and given as input to discriminator D , so the learning becomes supervised. The noise vector is given as input to generator G which generates the fake sample. The real samples with the class labels and the fake image generated by generator G are given as input to D . The D will discriminate between real and fake image and also classifies the image to which class it belongs to. The loss functions are computed for G and D . If the D is not discriminating properly, then by using backpropagation and stochastic gradient the parameters of the G and G will be adjusted for every iteration until D discriminates correctly. The discriminator loss function is given by

$$L_{\text{semiGAN}}(D) = E_{r \sim p_{d(r)}} [\log(D(r|c))] \quad (2.27)$$

The generator loss function is given by

$$L_{\text{semiGAN}}(G) = E_{n \sim p_{n(n)}} [\log(1 - D(G(n)))] \quad (2.28)$$

Using min-max game theory, the D has to output 1 if the image is the real sample. Hence the D has to be maximized. The D has to output 0 if the image is from the generator. Hence the G has to be minimized. The loss function is given by

$$\min_G \max_D L_{\text{SemiGAN}}(G, D) = \min_G \max_D \left\{ E_{r \sim p_{d(r)}} [\log(D(r|c))] + E_{n \sim p_{n(n)}} [\log(1 - D(G(n)))] \right\} \quad (2.29)$$

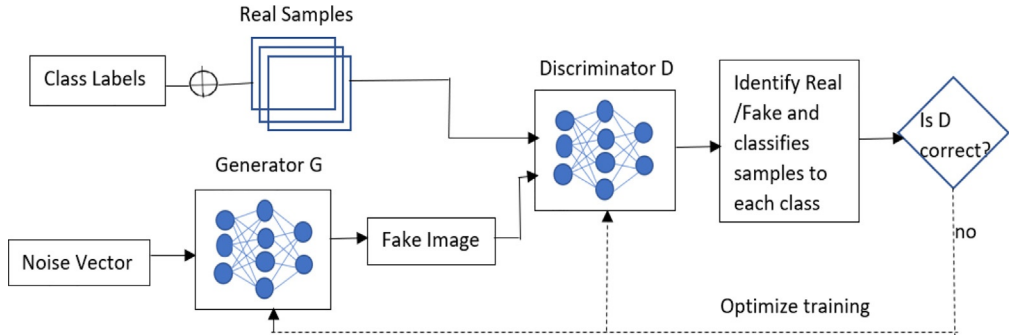


Fig. 2.8 Semi GAN.

The generator is not trained with the class labels but the discriminator has been trained with the class labels. The following sections describe about supervised learning.

2.2.3 Supervised learning

Supervised learning is making the machine learning with the labeled data. CGAN, BiGAN, AC GAN, and supervised sequential GAN are the GAN architectures which will learn in a supervised manner. The architectures of the abovementioned GAN are shown in the following sections. This section details about each GAN architecture and the loss functions and optimization techniques have been used in each architecture.

2.2.3.1 CGAN

CGAN stands for conditional GAN [23, 24]. The architecture of CGAN is shown in Fig. 2.9. The class labels are attached with the real samples. Generator G and discriminator D are trained with the class labels. The noise vector along with the class labels are given as input to G . The G outputs the fake image. The class labels, real, and fake image generated by G are given as input to D . The D discriminate between the real and fake image also find out to which class the image belongs to. The loss function is the same as that of vanilla GAN with one difference that the class labels “ c ” are added with the real sample, discriminator, and the generator terms. The binary cross-entropy loss [25] is used and the stochastic gradient descent is used for optimizing G and D when D is not discriminating properly.

The discriminator loss function is given by

$$L_{CGAN}(D) = E_{r \sim p_d(r)} [\log(D(r|c))] \quad (2.30)$$

The generator loss function is given by

$$L_{CGAN}(G) = E_{n \sim p_n(n)} [\log(1 - D(G(n|c)))] \quad (2.31)$$

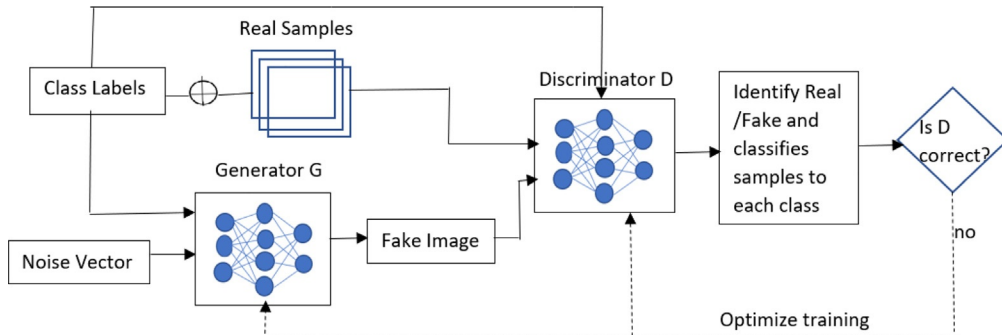


Fig. 2.9 CGAN architecture.

Using min-max game theory, the D has to output 1 if the image is the real sample. Hence the D has to be maximized. The D has to output 0 if the image is from generator. Hence the G has to be minimized. The loss function is given by

$$\min_G \max_D L_{CGAN}(G, D) = \min_G \max_D \left\{ E_{r \sim p_d(r)} [\log(D(r|c))] + E_{n \sim p_n(n)} [\log(1 - D(G(n|c)))] \right\} \quad (2.32)$$

CGANs [26] with multilabel predictions can be used for automated image tagging where the generator can generate the tag vector distribution conditioned on image features.

2.2.3.2 BiGAN

BiGAN stands for bidirectional GAN [8, 27, 28]. The architecture of BiGAN is shown in Fig. 2.10. The noise vector is given as input to generator G which generates the fake image. The real sample is given as an input to the encoder with output the encoded image with which the noise is added. The encoded image, noise, real image, and generated fake image are given as input to discriminator D . The discriminator discriminates between real and fake images. The loss functions are computed for G and D . If D is not discriminating properly, then by using backpropagation and stochastic gradient the parameters of the G and G will be adjusted for every iteration until D discriminates correctly. The discriminator loss function is given by

$$L_{\text{BiGAN}}(D) = E_{r \sim p_d(r)} \left[E_{n \sim p_E(n|r)} [\log(D(r, n))] \right] \quad (2.33)$$

The discriminator has been trained with the real data, noise, and encoded image distribution. The generator loss function is given by

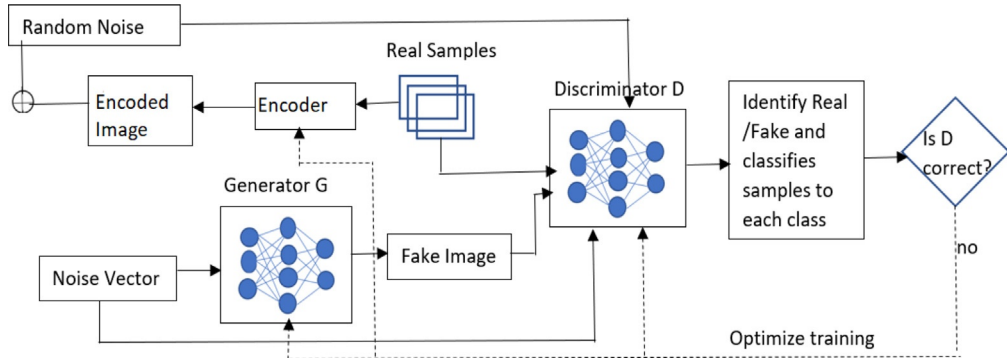


Fig. 2.10 BiGAN architecture.

$$L_{\text{BiGAN}}(G) = E_{n \sim p_n(n)} \left[E_{r \sim p_G(r|n)} [\log(1 - D(r, n))] \right] \quad (2.34)$$

Using min-max game theory, the D has to output 1 if the image is the real sample. Hence the D has to be maximized. The D has to output 0 if the image is from the generator. Hence the G has to be minimized. The loss function is given by

$$\begin{aligned} \min_{G,E} \max_D L_{\text{BiGAN}}(D, E, G) = \min_{G,E} \max_D \left\{ E_{r \sim p_d(r)} \left[E_{n \sim p_E(n|r)} [\log(D(r, n))] \right] \right. \\ \left. + E_{n \sim p_n(n)} \left[E_{r \sim p_G(r|n)} [\log(1 - D(r, n))] \right] \right\} \quad (2.35) \end{aligned}$$

2.2.3.3 ACGAN

The architecture of ACGAN [12, 29] is shown in Fig. 2.11. The architecture is the same as that of CGAN with one difference the class labels “ c ” are conditioned with the real samples and noise vector which is given as input to generator G . The class labels are not conditioned with the discriminator D . The training is based on the log probability of correct source whether the image is real or fake image generated by G is real or fake and log probability of correct class to which the sample belongs to. The stochastic gradient descent is used to adjust the weights and bias of G and D for every iteration if D is not discriminating correctly.

The log probability of the correct source whether the image is from the real sample or the image is generated by generator G is given by

$$L_{\text{source}} = E[\log P(\text{source} = \text{real} | R_{\text{real}})] + E[\log P(\text{source} = \text{fake} | R_{\text{fake}})] \quad (2.36)$$

The log probability of the correct class to which the image belongs to or classified correctly is given by

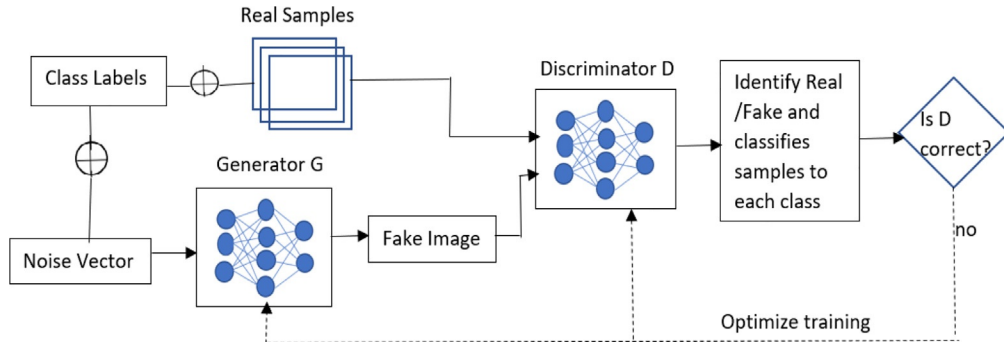


Fig. 2.11 ACGAN architecture.

$$L_{\text{class}} = E[\log P(\text{class} = c | R_{\text{real}})] + E[\log P(\text{class} = c | R_{\text{fake}})] \quad (2.37)$$

The image samples are given by “R.” The conditional probability has been used. The training has to be carried out in the way that D has to maximize $L_{\text{source}} + L_{\text{class}}$ and G has to maximize $L_{\text{source}} - L_{\text{class}}$.

2.2.3.4 Supervised seq-GAN

The supervised sequential GAN [25, 30, 31] architecture is shown in Fig. 2.12. The real image is given as input to the encoder that outputs the encoded image. The encoded image is given as input to the G1 which in turn generates the fake image1. The fake-image1 is given as input to G2 which will generate fakeimage2. The noise vector, encoded image, and fake image1 are given as input to D1. The noise vector, encoded image, and fake image2 are given as input to D2. D1 and D2 will discriminate between real and fake images. The loss function by considering G2 and D2 is given by

$$L_{\text{adv}}(G1, D1, n, r) = E_{r \sim p_{d(r)}}[\log(D1(r))] + E_{n \sim p_{n(n)}}[\log(1 - D1(G1(n)))] \quad (2.38)$$

The loss function by considering G2 and D2 is given by the following equations.

$$L_{\text{img2img}}(G2, D2, f, r) = E_{r \sim p_{d(r)}}[\log(D2(r))] + E_{f \sim p_{f(f)}}[\log(1 - D2(G2(f)))] \quad (2.39)$$

$$L_{\text{encoder}}(r, n) = E_{r \sim p_{d(r)}} \left[E_{n \sim p_E(n|r)}[\log(D(r, n))] \right] + E_{n \sim p_{n(n)}} \left[E_{r \sim p_G(r|n)}[\log(1 - D(r, n))] \right] \quad (2.40)$$

The loss function of unsupervised sequential GAN is given by the following equation

$$L_{\text{SupseqGAN}}(G1, D1, G2, D2) = L_{\text{adv}}(G1, D1, n, r) + L_{\text{img2img}}(G2, D2, f, r) + L_{\text{encoder}}(r, n) \quad (2.41)$$

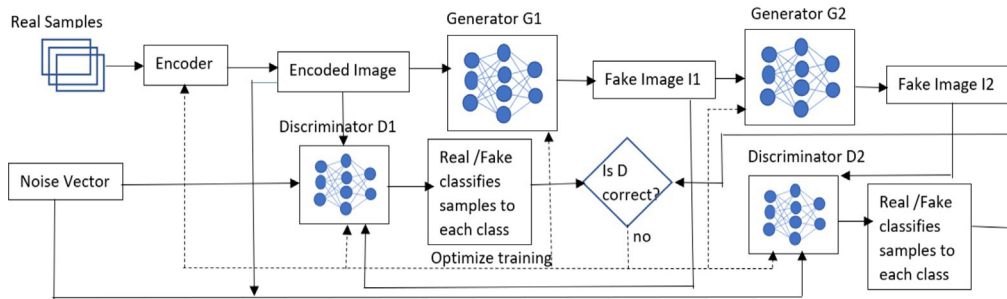


Fig. 2.12 Supervised sequential GAN.

2.2.4 Comparison of GAN models

This section discusses the comparison of GAN models. [Table 2.1](#) summarizes the activation function, loss function, distance metrics, and optimization techniques used by the GAN models.

Table 2.1 Loss functions and distance metrics of GAN's.

GAN	Activation function	Loss function	Distance metric	Optimization technique
Vanilla GAN	Rectified linear unit (ReLU)	Binary cross entropy loss	Jenson Shannon divergence	Back propagation with stochastic gradient decent RMS prop
WGAN	ReLU, leaky ReLU, tanh	Kantorovich-Rubinstein duality loss	Wasserstein distance	Adam
WGAN—GP	ReLU, leaky ReLU, tanh	Kantorovich-Rubinstein duality + penalty term added when the gradient moves away from 1	Wasserstein distance	
Info GAN	Rectified linear unit (ReLU)	Binary cross entropy + variational information regularization	Jenson Shannon divergence	Stochastic gradient decent
BEGAN	Exponential linear unit (ELU)	Auto encoder loss + proportional control theory	Wasserstein distance	Adam
Unsupervised seq-GAN	Rectified linear unit (ReLU)	Binary cross entropy + image to image conversion loss	Jenson Shannon divergence and Kullback-Leibler divergence	RMS prop
Parallel GAN	Rectified linear unit (ReLU)	Binary cross entropy loss	Jenson Shannon divergence	Stochastic gradient decent
Cycle GAN	ReLU, sigmoid	Binary cross entropy loss + cycle consistency loss	Jenson Shannon divergence	Batch normalization
Semi GAN	ReLU	Binary cross entropy loss with labels included with real samples	Jenson Shannon divergence	Stochastic gradient decent

Table 2.1 Loss functions and distance metrics of GAN's—cont'd

GAN	Activation function	Loss function	Distance metric	Optimization technique
CGAN	ReLU	Binary cross entropy loss with labels included	Jenson Shannon divergence	Stochastic gradient decent
BiGAN	ReLU	Binary cross entropy loss + guarantee G and E are inverse	Jenson Shannon divergence	Stochastic gradient decent
AC GAN	ReLU	Log likelihood of real source + log likelihood of correct label	Jenson Shannon divergence	Stochastic gradient decent
Supervised seq-GAN	ReLU	Binary cross entropy + image to image conversion loss + auto-encoder loss	Jenson Shannon divergence and Kullback-Leibler divergence	RMS prop

2.2.5 Pros and cons of the GAN models

This section discusses the pros and cons of GAN models. [Table 2.2](#) summarizes the pros and cons of the various GAN models.

2.3 GANs in natural language processing

Currently, many GAN architectures are emerging and yielding good results for the natural language processing applications. There have been various GAN architectures proposed in the recent years, including SeqGAN with policy gradient that is used for generating speech, poems, and music which outperforms other architectures. The RankGAN is used for generating sentences where the discriminator will act as the ranker. The following subsection elaborates on the various GAN architectures proposed for the applications of NLP.

2.3.1 Application of GANs in natural language processing

This section discusses the various GAN architectures such as SeqGAN, RankGAN, UGAN, Quasi-GAN, BFGAN, TH-GAN, etc., proposed for the applications of natural language processing.

Table 2.2 Pros and cons of GAN models.

GAN	Pros	Cons
Vanilla GAN	GAN can generate samples that are more similar to the real samples. It can learn deep representations of the data	When the real image distribution and generated fake image distribution is not overlapping then Jensen Shannon divergence between real and fake image distribution value becomes $\log 2$. The derivative of $\log 2$ is 0, which means learning will not take place at the initial start of back propagation
WGAN	The experiments conducted using WGAN reveals that it does not lead to the problem of mode collapse	The Lipschitz constant is been applied. Weight clipping is a simple technique but it will lead to poor quality image generation
WGAN—GP	The training is very balanced. Hence the machine could be trained effortlessly which will make the model converge properly	Attaining Nash equilibrium state is very hard. Batch normalization cannot be used as gradient penalty will be applied to all data samples
Info GAN	It is used for learning data representations which are disentangled by using information theory extensions	The mutual info has been included to generator that will eliminate the significant attributes from data and assign them to semantic information while learning is in progress. Fine tuning λ hyperparameter if not done accurately, it will not generate good quality image
BEGAN	The training is fast and stable	The hyperparameter γ must be fine tuned properly. The appropriate learning rate has to be set properly. If not done properly, it is not generate good clarity image
Unsupervised seq-GAN	It extract more deep features	Hard to achieve Nash equilibrium
Parallel GAN	Multiple images can be generated at same time	Hard to achieve Nash equilibrium
Cycle GAN	The requirement for dataset is low. Randomly two image styles can be converted	When doing image-to-image translation, considering various parameters such as color, texture, geometry, etc. is very difficult
Semi-GAN	It is an effective model which can be used for the regression tasks	The generator cannot generate more realistic image to fool the discriminator as it is strong enough to discriminate since it is been trained with the class labels

CGAN	The class labels are included which increases the performance of the GAN and it can be used for many applications such as shadow maps generation, image synthesis, etc.	The training is not stable. The stability in training can still be improved
BiGAN	As class labels are included, it can generate good realistic images	The drawback is that the real image sample which is been given to the encoder must be of good clarity and it cannot perform well when the data distributions are complex
AC GAN	As class labels are included, it can generate good realistic images	The GAN training is not stable. The ACGAN training can be still improved
Supervised seq-GAN	It extract more deep features	The real sample given to the encoder should be of good clarity otherwise it will not generate realistic image

2.3.1.1 Generation of semantically similar human-understandable summaries using SeqGAN with policy gradient

In recent years, generating text summaries have become attractive in the area of natural language processing. The SeqGAN with policy gradient architecture has been proposed for generating text summary. The proposed SeqGAN with policy gradient architecture [32] has three neural networks namely one generator (G) and two discriminators, viz., $D1$ and $D2$ as shown in Fig. 2.13. The G is the sequential model which takes the raw text as input and generates the summary of the text as output. The $D1$ trains G to output summaries which are human readable. Hence G and $D1$ form the GAN. The $D1$ is trained to distinguish between input text and the summary generated by G . G is trained to fool $D1$. As $D1$ trains the generator to generate the human-readable summary, it is called as human-readable summary discriminator. The summary generated by the generator might be irrelevant with only G and $D1$. Hence another discriminator $D2$ is added to the architecture for checking the semantic similarity between the input raw text and the generated human-readable summary. SeqGAN incorporates reinforcement learning. Policy gradient is the optimization technique used for updating the parameters (weights and bias) of G by obtaining rewards from $D1$ and $D2$. Hence $D1$ will train G to generate a semantically similar summary and $D2$ will train G to generate human-readable summary.

Semantic similarity discriminator

The semantic similarity discriminator is trained as the classifier using the text summarization dataset shown in Fig. 2.14. This discriminator will teach the generator to generate a semantically similar and more concise summary. The raw text and the human-readable summary are given as inputs to the encoders individually to generate the encoded representations namely R_i and R_s . The R_i and R_s are concatenated, product and difference are performed and given to the four-class classifiers which classify the human-readable summary into four classes namely similar, dissimilar, redundant, and incomplete class. The softmax outputs the probability distribution.

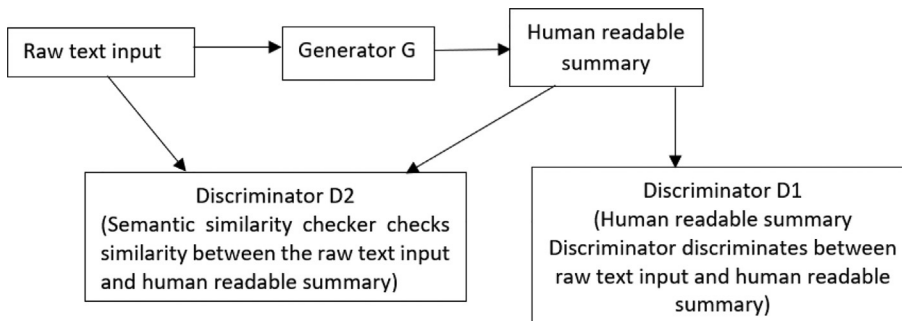


Fig. 2.13 SeqGAN for generation of human-readable summary.

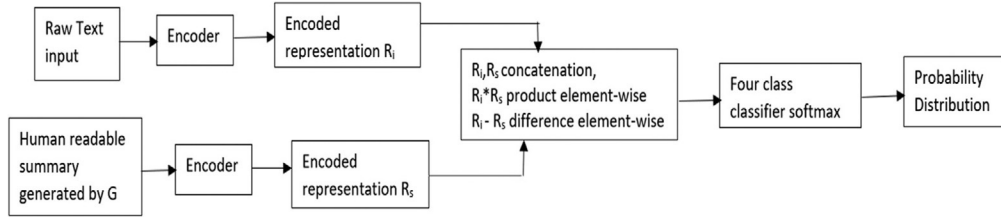


Fig. 2.14 Semantic similarity discriminator.

2.3.1.2 Generation of quality language descriptions and ranking using RankGAN

Language generation plays a major role in many NLP applications such as image caption generation, machine translation, dialogue generation systems, etc. Hence the RankGAN has been proposed to generate high-quality language descriptions. The RankGAN [33] consists of two neural networks such as generator G and ranker R . The generative model used is long short-term memory (LSTM) to generate the sentences which are called machine-written sentences. Instead of the discriminator being trained to be a binary classifier, RankGAN uses a ranker which has been trained to rank the human-written sentences more than the machine-written sentences. The ranker will train the generator to generate machine-written sentences which are similar to human-written sentences. In this way, generator fools the ranker to rank the machine-written sentences more than the human-written one. The policy gradient method is used for optimizing the training.

The architecture of RankGAN is shown in Fig. 2.15. The G generates the sentences from the synthetic dataset. The human-written sentences with the generated machine-written sentences are given as input to the ranker. The reference human-written sentence

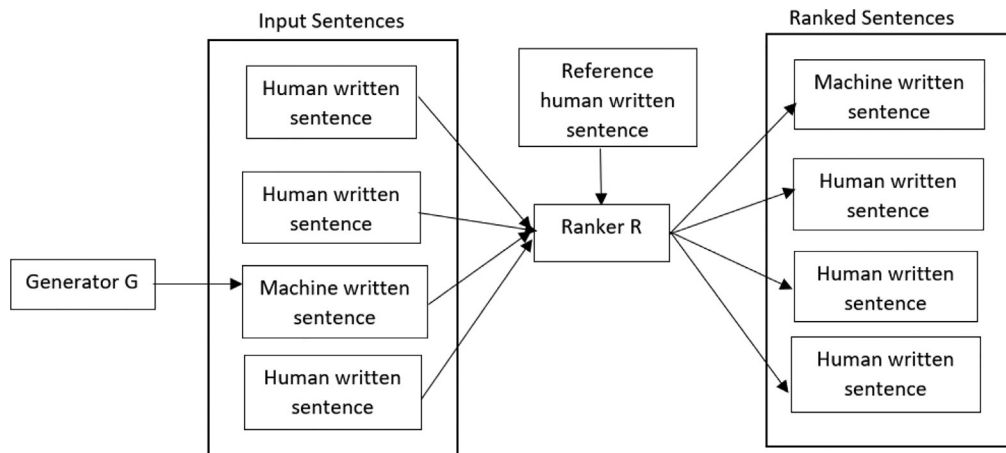


Fig. 2.15 Architecture of RankGAN.

is also given as input to the ranker. The ranker has to rank human-written sentences more than the machine-written sentence. The generator G will be trained to fool the ranker, hence the ranker will rank the machine-written sentence more than the human-written sentence. The ranker will compute the rank score by using

$$R(i|S, C) = E_{s \in S} [P(i|s, C)] \quad (2.42)$$

where $P(i|s, C) = \frac{\exp(\beta \alpha(i|s))}{\sum_{s' \in C'} \exp(\beta \alpha(i'|s))}$ and $\alpha(i|s) = \text{cosine}(x_i, x_s)$

x_i is the feature vector of input sentences. x_s is the feature vector of reference sentences. The parameter β value is set during the experiment empirically. The reference set S is constructed by sampling reference sentences from human-written sentences. C is the comparison set sampled from both human-written and machine-generated sentence set. s is the reference sentence sampled from set S .

2.3.1.3 Dialogue generation using reinforce GAN

Dialogue generation is the most important module in applications such as Siri, Google assistant, etc. The reinforce GAN [34] was proposed for dialogue generation using reinforcement learning. The architecture of reinforced GAN is shown in Fig. 2.16. The reinforce GAN has two neural network architectures namely generator G and discriminator D . The input dialogue history is given to the generator which outputs the machine-generated dialogue. The {input dialogue history, machine-generated dialogue} pair is given to the hierarchical encoder which outputs the vector representation of dialogue. The vector representation is given as input to the discriminator D which in turn outputs the probability that the dialogue is human generated or machine generated. The policy gradient optimization technique is used. The weights and bias of G and D are adjusted by the rewards generated by them during training. The discriminator outputs will be used as rewards to train the generator so that the generator can generate a dialogue which is more similar to the human-generated dialogue.

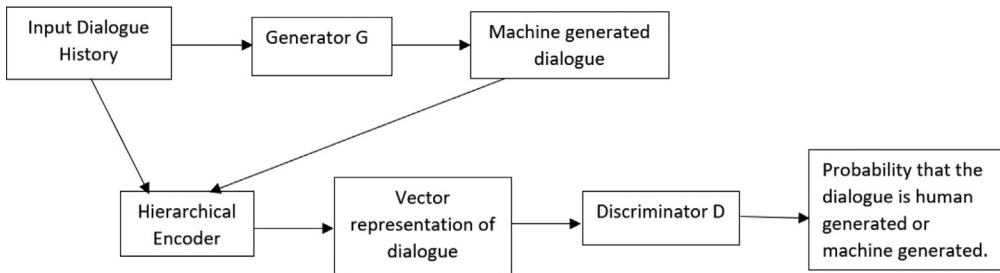


Fig. 2.16 Architecture of reinforce GAN.

2.3.1.4 Text style transfer using UGAN

Text style transfer is an important research application of natural language processing which aims at rephrasing the input text into the style that is desired by the user. Text style transfer has its application in many scenarios such as transferring the positive review into a negative one, conversion of informal text into a formal one, etc. Many techniques that are used for text style transfer are unidirectional, i.e., it transfers the sentence from positive to negative form. UGAN (Unified Generative Adversarial Networks) [35] is the only architecture which does multidirectional text style transfer shown in Fig. 2.17. Input to the architecture will be the sentence and the target attribute, for example input: sentence: “chicken is delicious” and target attribute: “negative.” Output of the architecture will be the transferred sentence. Output: “chicken is horrible” and vice versa. UGAN has two networks namely generator and discriminator. The LSTM is the generator network which takes the sentence and the target attribute as input and generates the output sentence as per the target attribute. The output transferred sentence generated by LSTM is given as the input to the discriminator. The discriminator uses the RankGAN rank score computation equations to rank the original sentence and the generated sentence. The classification of the sentence whether “positive” or “negative” is done by the discriminator.

2.3.1.5 Tibetan question-answer corpus generation using Qu-GAN

In recent years, many question answering systems have been designed for many languages using deep learning models. It is hard to design a question-answering systems for languages with less resources such as Tibetan. To solve this problem, QuGAN [36] has been proposed for a question answering system. The architecture of the QuGAN is shown in Fig. 2.18. Initially, by using maximum likelihood, some amount of data is sampled from

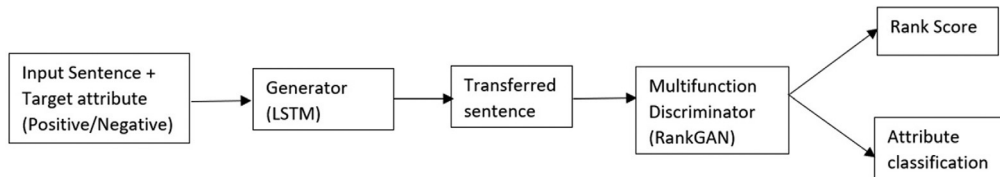


Fig. 2.17 Architecture of UGAN.

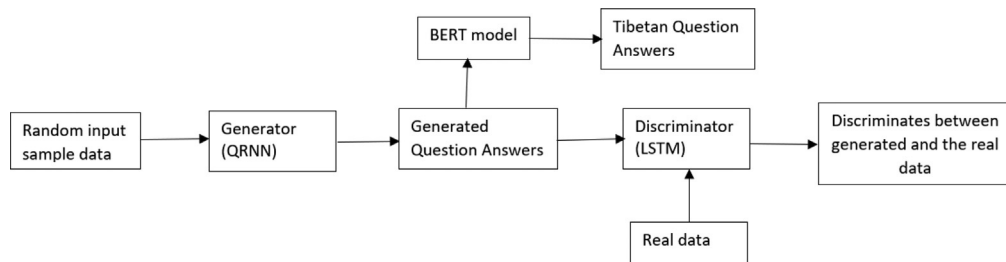


Fig. 2.18 Architecture of QuGAN.

the data in the database. This is done to reduce the distance between the probability distribution of the real and the generated data. The randomly sampled data is given to the generator (quasi recurrent neural network—QRNN) which generates the question and answers which in turn is given to the BERT model to correct the grammatical errors and syntax. The generated and the real data are given as an input to the discriminator (long short-term memory—LSTM) which classifies between the real and the generated data. The policy gradient and the Monto-Carlo search optimization techniques are used to optimize the training of the neural networks by adjusting their weights and bias.

2.3.1.6 Generation of the sentence with lexical constraints using BFGAN

Nowadays for generating meaning sentences, lexical constraints are incorporated to the model which has applications in machine translation, dialogue system, etc. For generating lexically constrained meaningful sentences, BFGAN (backward forward) [37] has been proposed as shown in Fig. 2.19. BFGAN has two generators namely forward and backward generators and one discriminator. The LSTM dynamic attention-based model called as attRNN is used as the generators. The discriminator can be CNN-based binary classifier to classify between real sentences and machine-generated meaningful sentences. The input sentence is split into words and given as input to the backward generator which generates the first half of the sentence in the backward direction. The backward sentence is reversed and fed as input to the forward generator which in turn outputs the complete sentence with lexical constraints. The discriminator is used for making the backward and forward generators powerful by training them using the Moto Carlo optimization technique. The real sentence and the generated sentences are given as input to the discriminator which will classify between real and the machine-generated complete sentence with the lexical constraints incorporated in it.

2.3.1.7 Short-spoken language intent classification with cSeq-GAN

Intent classification in dialog system has grabbed attention in industries. For intent classification, cSeq-GAN [38] has been proposed shown in Fig. 2.20. cSeq-GAN has two

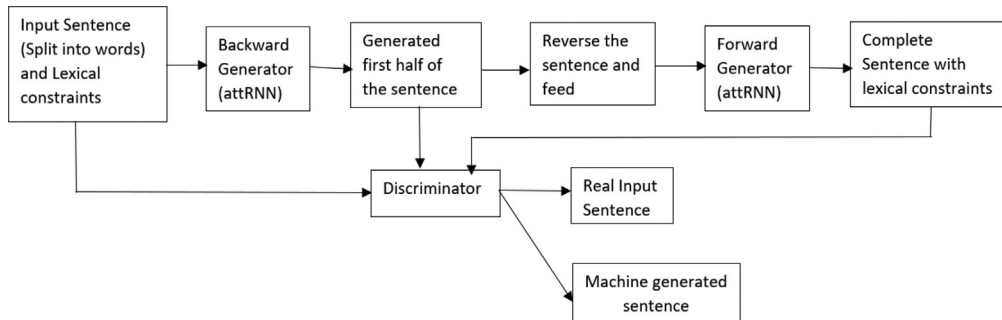


Fig. 2.19 Architecture of BFGAN.

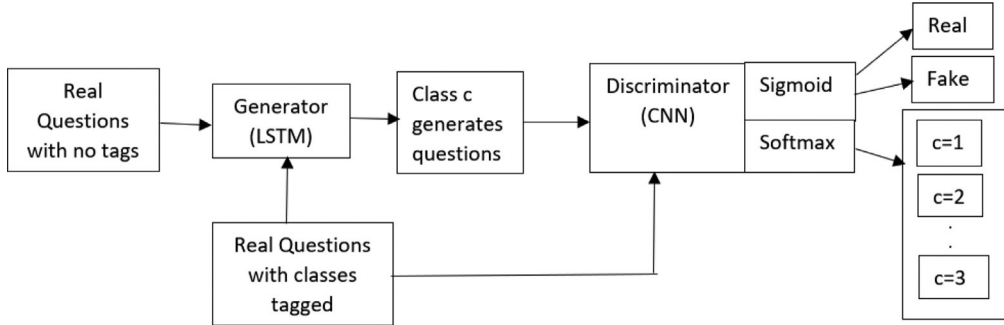


Fig. 2.20 Architecture of cSeq-GAN.

neural networks namely the generator (LSTM) and the discriminator (CNN). The real questions with no tags and with tags are given as input to the generator. The generator in turn generates questions with classes. The generated and the real questions with tags are given as input to the discriminator. The CNN is used as the discriminator which has been implemented with both the sigmoid and the softmax layer. The sigmoid layer classifies the real and the generated questions. The softmax layer is used for classifying the questions to the respective intent class. The policy gradient optimization technique is used for adjusting the weights and bias of the generator and discriminator during training.

2.3.1.8 Recognition of Chinese characters using TH-GAN

Historical Chinese characters are of low-quality images. In order to enhance the quality of the historical Chinese character images, TH-GAN (transfer learning-based historical Chinese character recognition) [39] has been proposed shown in Fig. 2.21. The generator used is the U-Net architecture. The WGAN model has been used. The source Chinese character is given as input to the generator which outputs generated Chinese character. The target image, real character image, and the generated character images are given as input to the discriminator. The discriminator classifies between the real and the fake character image. The policy gradient is the technique used for adjusting weights and bias of the generator and the discriminator during training. The following session discusses the NLP datasets.

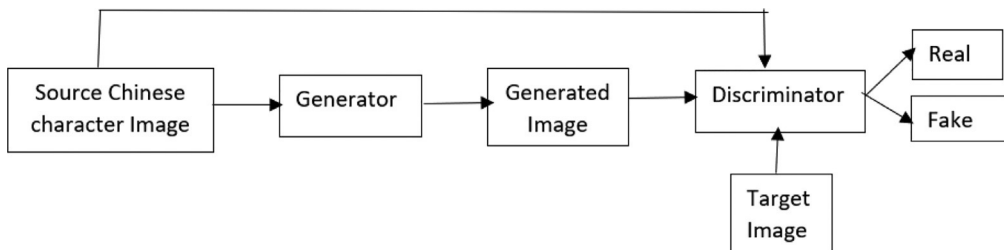


Fig. 2.21 Architecture of TH-GAN.

2.3.2 NLP datasets

The open-source free NLP datasets available for research is shown in [Table 2.3](#).

2.4 GANs in image generation and translation

In recent years, for image generation and translation, many GAN architectures have been proposed such as cycleGAN, DualGAN, DiscoGAN, etc. The following section discusses the various applications of GANs in image generation and translation.

2.4.1 Applications of GANs in image generation and translation

The following subsections discuss the various applications of GANs in image generation and translation.

2.4.1.1 Ensemble learning GANs in face forensics

Fake images generated by newer image generation methods such as face2face and deepfake are really hard to distinguish using previous face-forensics methods. To overcome the same, a novel generative adversarial ensemble learning method [40] has been proposed as shown in [Fig. 2.22](#). In this GAN, two **generators** with the same architecture are used but both of them are trained in different ways. The feedback face generator gets the feedback from the discriminators and generates the more fine-tuned image. As a **discriminator** ResNet and DenseNet are used. The ability to discriminate between real and fake images is achieved by combining the feature maps of both ResNet and DenseNet. Image is fed to both the network and 1024-dimensional output feature is extracted by using global average pooling and then a 2048-dimensional feature vector is generated by taking output features by both the networks and concatenating them, later SoftMax function is used to normalize the 2D scores. During the training process of the GAN, the spectral normalization method is used for the stabilization of the process.

2.4.1.2 Spherical image generation from the 2D sketch using SGANs

Most of the VR applications rely mostly on panoramic images or videos, and most of the image generation models just focus on 2D images and ignore the spherical structure of the panoramic images. To solve this, a panoramic image generation method based on spherical convolution and GAN called SGAN [41] is proposed shown in [Fig. 2.23](#). For input, a sketch map of the image is taken, which provides a really good geometric structure representation. A custom designed **generator** is used to generate the spherical image and it reduces the distortion in the image using spherical convolution, loss of least squares is used to describe the constraint for whether the discriminator is able to distinguish the image generated from the real image. The spherical convolution is used for observing the data from multiple angles. **Discriminator** is used to distinguish between generated

Table 2.3 NLP datasets.

NLP datasets	Description	Link
CNN/Daily Mail dataset	It is the text summarization dataset which as two features namely the documents need to be summarized (article) and the target text summary (highlights)	https://github.com/abisee/cnn-dailymail
News summarization dataset	It has the author details, date of the news, headlines and the detailed news link	https://www.kaggle.com/sunnysai12345/news-summary
Chinese poem dataset	It has small poems. Each poem is with 4–5 lines and each line is with 4–5 words	https://github.com/Disiok/poetry-seq2seq https://github.com/XingxingZhang/rnnpg
COCO (common objects in context) captions	It is the object detection and caption dataset. The dataset has five sections such as info, licenses, images, annotations, and category	http://cocodataset.org/#download
Shakespear's plays	It consists of 715 characters of Shakespeare plays. It has continuous set of lines spoken by each character in a play. It can be used for text generation	https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/shakespeare/load_data
Open subtitles dataset	It has the group of translated movie subtitles. It has subtitles of 62 languages	https://github.com/PolyAI-LDN/conversational-datasets/tree/master/opensubtitles
YELP	It is a business reviews and user dataset. It has 5,200,000 user business reviews Information about 174,000 businesses The data about 11 metropolitan areas	https://www.kaggle.com/yelp-dataset/yelp-dataset
Amazon	It is an amazon review dataset	https://registry.opendata.aws/?search=managedBy:amazon
Caption	It consists of approximately 3.3 million image caption pairs	https://ai.googleblog.com/2018/09/conceptual-captions-new-dataset-and.html
Noisy speech	Noisy and clean speech dataset. It can be used for speech enhancement applications	https://datashare.is.ed.ac.uk/handle/10283/2791
OpinRank	It consists of 3,000,000 reviews on cars, hotels collected from tripadvisor	http://kavita-ganesan.com/entity-ranking-data/#.XuxKF2gzY2z
Legal case reports	It consists of text summaries of about 4000 cases. It can be used training text summarization tasks	https://archive.ics.uci.edu/ml/datasets/Legal+Case+Reports

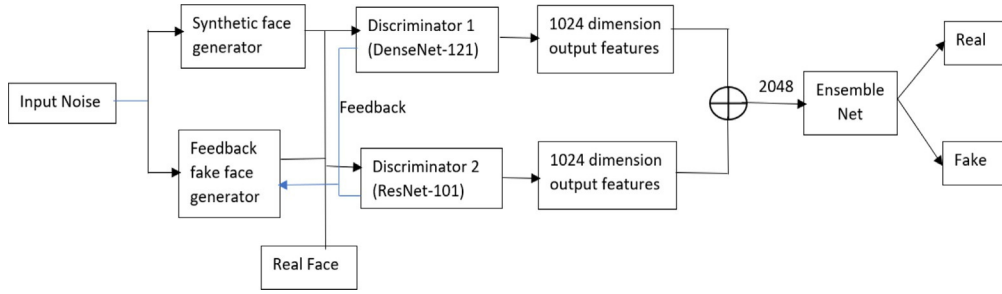


Fig. 2.22 Architecture of ensemble learning GAN.

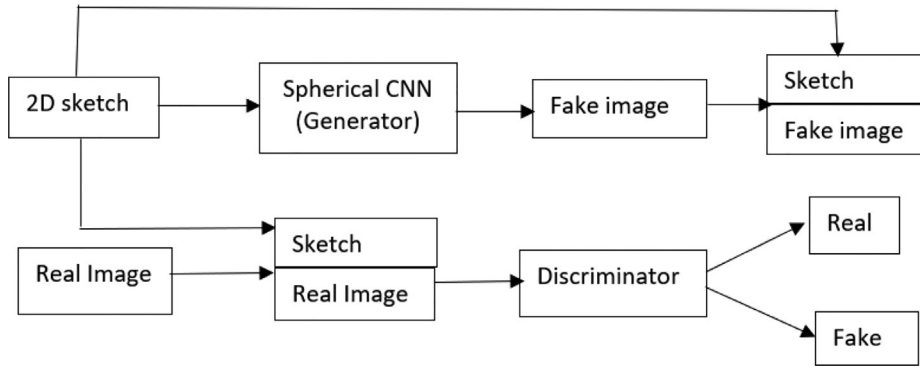


Fig. 2.23 Architecture of SGAN.

images and real images and in the case of image generation, a multiscale discriminator is used, which is quite common and adds the advantage of decreasing the burden on the network.

2.4.1.3 Generation of radar images using TsGAN

Radar data becomes really hard to understand due to imbalanced data and also becomes the bottleneck for some operations. To defend the radar operations, a two-stage general adversarial network (TsGAN) [42] has been introduced, as shown in Fig. 2.24. **In the first stage**, it generates samples which are similar to real data and distinguishes its eligibility. To generate radar image sequences, each frame is decomposed as content information and motion information. Also, for capturing data such as the flow of clouds, RNN is used. For **discriminators**, two of them are being used, one for distinguishing between radar image and generated image and the second one for motion information, like image generation sequence. **The second stage is** used to define the relationship between intervals and adjacent frames. The rank discriminator is used for computing

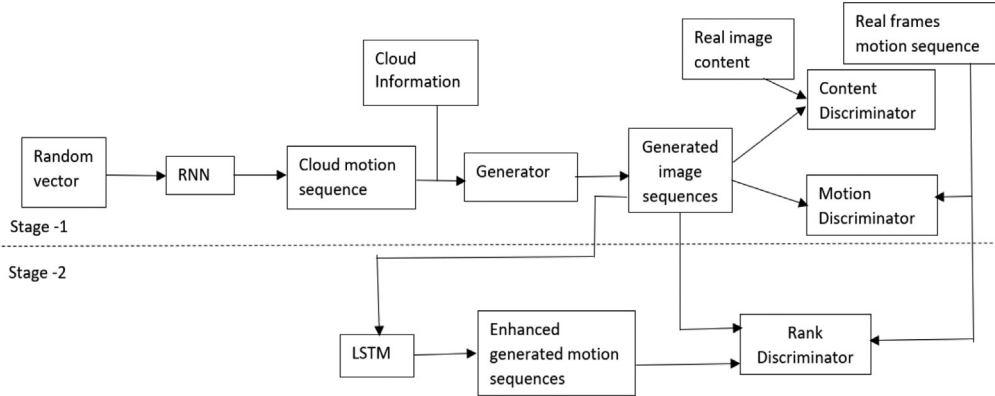


Fig. 2.24 Architecture of TsGAN.

the rank loss between generated motion sequences, real motion sequences and the enhanced generated motion sequences.

2.4.1.4 Generation of CT from MRI using MCRCGAN

The MRI (magnetic resonance images) are really useful in radiation treatment planning with the functional information that provides as compared with CT (computed tomography). But there are some applications where MRI cannot be used because of the absence of electron density information. To apply MRI for these types of applications, MCRCGAN (multichannel residual conditional GAN) [43] has been introduced, which generates pseudo-CT as shown in Fig. 2.25. MCRCGAN has two parts, **generator** which generated the pseudo-CT image according to the input MR images, and **discriminator** is used to distinguish between p-CT images with the real ones and measure the degrees/number of mismatches since it helps the network feed accordingly for the next iteration for better efficiency. MCRCGAN actually adopts the multichannel ResNet as the generator and CNN as the discriminator.

2.4.1.5 Generation of scenes from text using text-to-image GAN

Generating an image from text is a vividly interesting research topic with very unique use cases but it is quite difficult since the language description and images vary a different part

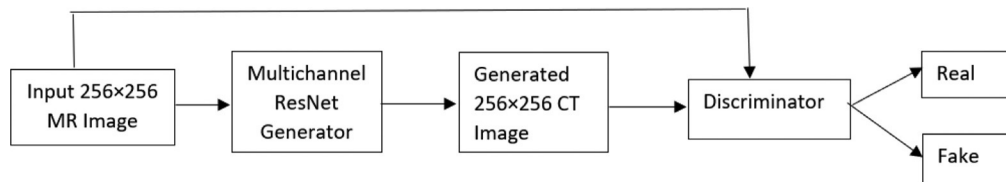


Fig. 2.25 Architecture of MCRCGAN.

of the world and the current models which generate images tend to mix the generation of background and foreground which leads to object in images which are really submerged into the background. To make sure that the generation of the image is done by keeping in mind about the background and foreground. To achieve this VAE (variational autoencoder) and GAN proved to be robust. Here the **generator** contains three modules, namely, downsampling module, upsampling module, and residual module. The architecture of text-to-image GAN [44] is shown in Fig. 2.26.

2.4.1.6 Gastritis image generation using PG-GAN

For detection of gastric cancer, X-ray images of gastric are used. Multiple X-ray images are relatively large in size so LC-PGGAN (loss function-based conditional progressive growing GAN) [45] has been introduced as shown in Fig. 2.27. This GAN generates images which are effective for gastritis classification and have all the necessary details to look for any sort of symptoms. For the generation of synthetic images, divided patched images are used. The whole process is divided into two different sections. (1) **low-resolution step**: Here fake and real images are given to the discriminator which sends the loss values to (2) **high-resolution step**: here fake images along with patches with random sampling and real images with patches are given to the discriminator to finalize the output.

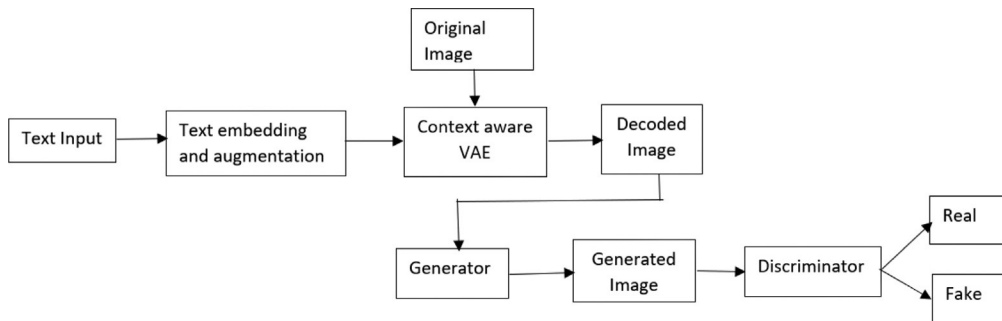


Fig. 2.26 The architecture of text-to-image GAN.

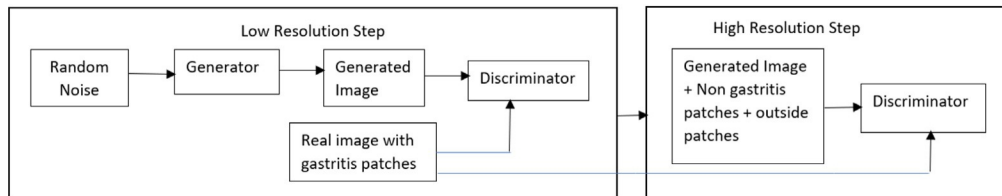


Fig. 2.27 Architecture of LC-PGGAN.

2.4.1.7 Image-to-image translation using quality-aware GAN

Image-to-image translation is one of the widely practiced with GAN and to do the same many works has been proposed but all of them depend on pretrained network structure or they rely on image pairs, so they cannot be applied on unpaired images. To solve these issues, a unified quality-aware GAN-based framework [46] was proposed as shown in Fig. 2.28. Here two different implementations of quality loss are done, one is based on the image quality score between the real and reconstructed image and another one is based on the adaptive deep network-based loss to calculate the score between the real and reconstructed image from the generator. Here the generators generate such as each constructed image has a similar or close score to the real image. The loss function includes adversarial loss, reconstruction loss, quality-aware loss, IQA loss, and content-based loss.

2.4.1.8 Generation of images from ancient text using encoder-based GAN

Ancient texts are of great use since it helps us to get to know about our past and maybe some keys to our future, to retrieve or understand these texts, an encoder-based GAN [9] has been introduced to generate the remote sensing images retrieved from the text retrieved from different sources as shown in Fig. 2.29. To train this particular network, we have used satellite images and ancient images. Here **generator** is conditioned with the training set text encodings and corresponding texts are synthesized. **The discriminator** is used to predicting the sources of input images, for whether they are real or synthesized. Text encoder and Noise generator is used prior to the input.

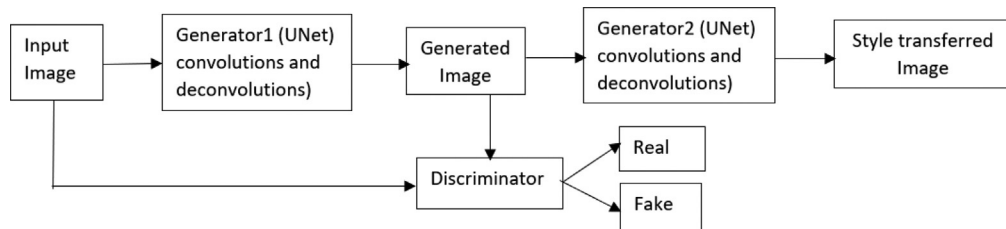


Fig. 2.28 Architecture of quality aware GAN.

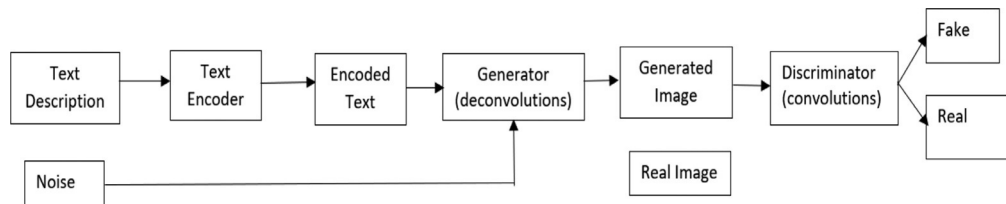


Fig. 2.29 Architecture of encoder-based GAN.

2.4.1.9 Generation of footprint images from satellite images using IGAN

For many architectural purposes and planning, building footprints plays an important role. To convert satellite images into footprint images, a IGAN (improved GAN) [26] was proposed as shown in Fig. 2.30. This GAN uses CGAN with the cost function from Wasserstein distance and integrated with gradient penalty. The **generator** is provided with noise and satellite image, using Leaky ReLU as activator function it generates a footprint image which then sent to **discriminator** helps to get the score, and if the score does not get as close as the real image, it goes to generator again and the iterations provide better results every time. The dataset was based on Munich and Berlin which gave 256×256 images to work on. Also, segmentation is used on images to get the visible gradients.

2.4.1.10 Underwater image enhancement using a multiscale dense generative adversarial network

Underwater image improvement has become more popular in underwater vision research. The underwater images suffer from various problems such as underexposure, color distortion, and fuzz. To address these problems, multiscale dense block generative adversarial network (MSDB-GAN) [47] for enhancing underwater images has been proposed as shown in Fig. 2.31. The random noise and the image to be enhanced are given as input to the generator. The multiscale dense block is embedded within the generator. The MSDB is used for concatenating all the local features of the image using the

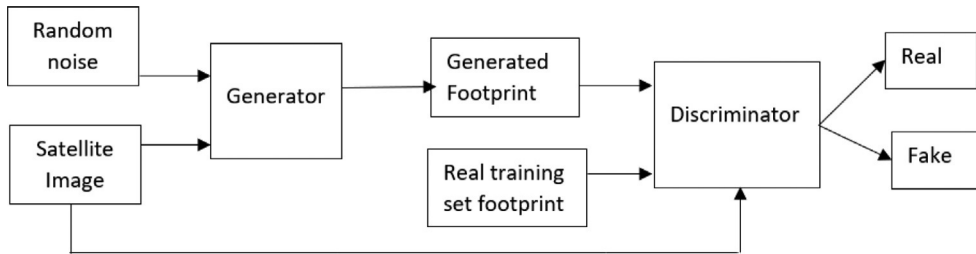


Fig. 2.30 Architecture of IGAN.

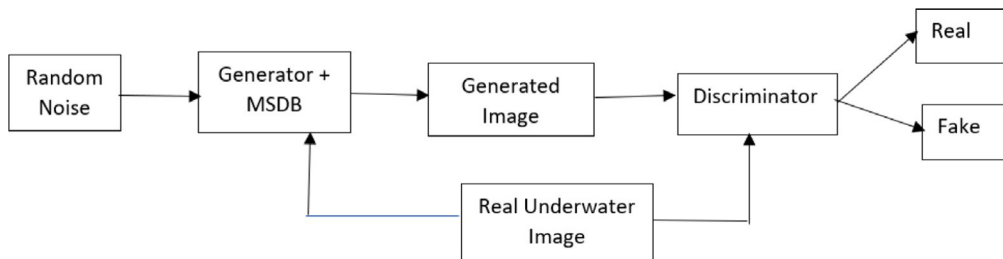


Fig. 2.31 Architecture of MSDB-GAN.

Leaky ReLU activation function. The discriminator discriminates between the real and the generated image.

2.4.2 Image datasets

The open-source free image datasets available for research are shown in Table 2.4.

The following section discusses the various evaluation metrics.

Table 2.4 Image datasets.

Image datasets	Description	Link
CelebA-HQ	It consists of 30,000 face images of high resolution	https://www.tensorflow.org/datasets/catalog/celeb_a_hq
AOI	It consists of 685,000 footprints of the buildings	https://spacenetchallenge.github.io/datasets/spacenetBuildings-V2summary.html
MRI brain tumor	It consist of 96 images of MRI brain tumor	https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection
CUB	It consists of 200 various bird species images	http://www.vision.caltech.edu/visipedia/CUB-200.html
Oxford 102	It consists of 102 various flow category images	https://www.robots.ox.ac.uk/~vgg/data/flowers/102/
CelebA	It consists of 200,000 celebrity images	http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html
OpenStreetMap	The map data can be downloaded by selecting the smaller areas from the map	https://www.openstreetmap.org/#map=5/21.843/82.795
Visual Genome	It consists of 108,077 Images with captions of people, signs, buildings, etc.	http://visualgenome.org/api/v0/api_home.html
Open Images	It consists of approximately 9,000,000 images been annotated with labels and bounding boxes for 600 object categories	https://storage.googleapis.com/openimages/web/download.html
<i>CIFAR 10/100</i>	<i>CIFAR 10</i> consists of 60,000 images of 10 classes. <i>CIFAR 100</i> is extended by 100 classes. Each class consists of 600 images	https://www.cs.toronto.edu/~kriz/cifar.html
Caltech 256	It consists of 30,000 images categorized in 256 classes	https://www.kaggle.com/jessicali9530/caltech256
LabelMe	It consists of 190,000 images, 60,000 annotated images, 658,000 labeled objects	http://labelme.csail.mit.edu/Release3.0/browserTools/php/dataset.php
<i>COIL-20</i>	It consists of 100 different toys images. Each toy being photographed in 72 poses. Hence 7200 images for 100 toys are present	https://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php

2.5 Evaluation metrics

This section discusses the various evaluation metrics that are needed to assess the performance of the GAN models.

2.5.1 Precision

Precision (P) refers to the percentage of the relevant results obtained during prediction. It is given by the ratio of true positives and the actual results.

$$P = \frac{TP}{TP + FP}$$

where TP is true positive and FP is false positive.

2.5.2 Recall

Recall (R) refers to the total percentage of the relevant results that are correctly classified by the classifier. It is given by the ratio of true positives and the predicted results.

$$R = \frac{TP}{TP + FN}$$

where TP is true positive and FN is false negative.

2.5.3 F1 score

F1 score is defined as the harmonic mean of both precision and recall. It is given twice the ratio of multiplication of precision and recall to the addition of precision and recall.

$$\text{F1 score} = 2 \left(\frac{P \times R}{P + R} \right)$$

where P is precision and R is recall.

2.5.4 Accuracy

Accuracy refers to how accurately the model predicts the results. It is given by the ratio of true positive and true negative results to the total results obtained.

$$\text{Accuracy} = \frac{TP + TN}{\text{Total}}$$

where TP is true positive and TN is true negative

2.5.5 Fréchet inception distance

Fréchet inception distance (FID) is the metric used to evaluate the quality of the images generated by the GANs. If FID is less, then it means the generator has generated a good

quality image. If FID is more, then it means the generator has generated a lower quality image.

$$\text{FID} = \|\mu_1 - \mu_2\|^2 + \text{Tr}(C_1 + C_2 - 2 \times \sqrt{C_1 \times C_2})$$

where μ_1 and μ_2 are feature-wise mean of real and generated images. C_1 and C_2 are covariance matrices of real and generated image feature vectors. Tr indicates trace linear algebra function.

2.5.6 Inception score

The inception score (IS) is the metric used for measuring both the quality of the generated image and the difference between the generated and the real image. For measuring the quality of the image, the inception network can be used to classify the generated and the real images. The difference between the real and the generated image is computed using KL-divergence.

$$\text{IS} = \exp(E_{g \sim G} D_{\text{KL}}(p(r|g) || p(r)))$$

where g is the generated image, r is the real samples with labels. D_{KL} is the Kullback-Leibler divergence measures the distance between the real and generated image probability distributions.

2.5.7 IoU score

Intersection over union (IoU) otherwise called as Jaccard index is the metric which computes the overlap between the predicted results and the ground truth samples. The score ranges from 0 to 1. 0 indicates no overlap.

$$\text{IoU score} = \frac{TP}{TP + FP + FN}$$

where TP is the true positive results, FP is the false positive results, and FN is the false negative results.

2.5.8 Sensitivity

Sensitivity measures the percentage of the true positives that are correctly computed.

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

where TP is the true positive results and FN is the false negative results.

2.5.9 Specificity

Specificity measures the percentage of the true negatives that are correctly computed.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

where TN is the true negative results and FP is the false positive results.

2.5.10 BELU score

Bilingual evaluation understudy (BELU) score is the metric used for measuring the similarity between the system-generated text and the input reference text.

$$\text{BELU} = \frac{N}{T}$$

where N is the total number of words matching between the system-generated text and the input reference text. T is the total number of system-generated words.

2.5.11 ROUGE score

Recall-oriented understudy for gisting evaluation (ROUGE) score is used for evaluating automatic text summarization. It evaluates by computing ROUGE recall and precision.

$$\text{ROUGE Precision} = \frac{N}{T}, \text{ROUGE Recall} = \frac{N}{R}$$

where N is the total number of words matching between the system-generated text and the input reference text. T is the total number of words in system-generated text. R is the total number of words in the input reference text.

The next section discusses the various languages and tools used for research.

2.6 Tools and languages used for GAN research

This section discusses the various languages that can be used for training the neural networks such as generator and the discriminator.

2.6.1 Python

For training the generator,

- (1) Pandas are used for data manipulation
- (2) Using OS, data path is set
- (3) Train and test data is divided using `pd.DataFrame()` function
- (4) In the infinite loop,
 - `pd.read_csv` is used to read the data from csv file
 - Labels are retrieved from the list using `data.iloc()` function
 - Append it into array using `append()` function
- (5) Inside generator `()`, `batch_size`, `shuffle_data` is used,
 - For setting array list empty `[]` list are initialized
 - `cv2.imread` is used to read images (if there is any)
 - reading array is done using `np.array`

For training the discriminator,

- (1) Keras can be used
- (2) The discriminator is defined using `def define_discriminator(n_inputs=2)`

- (3) Define the model type and the activation functions to be used by,
 - `model = Sequential ()`
 - `model.add (Dense(25, activation='relu', kernel_initializer='he_uniform', input_dim=n_inputs))`
 - `model.add (Dense(1, activation='sigmoid'))`
- (4) Compile the model by specifying the loss function and the optimizer to be used using `model.compile (loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])`
- (5) `return model`

2.6.2 R programming

- (1) Install the neural network packages using `install.packages("neuralnet")`
- (2) Load the neural net packages using `library ("neuralnet")`
- (3) Read the CSV file using `read.csv()`
- (4) Preview the dataset using `View()`
- (5) To view the structure and verify the ID variable `str()` function is used.
- (6) To set the input variables to the same scale, `scale (Any var[1:12])` is used.
- (7) Generate a random seed using `set.seed (200)`
- (8) Split the dataset into 70-30 train and test set using,
 - `ind <- sample(2, nrow (Any var), replace = TRUE, prob = c(0.7, 0.3))`
 - `train.data <- Any var[ind == 1,]`
 - `test.data <- Any var[ind == 2,]`
- (9) Neural network with one hidden layer and two nodes and linear output set to false is given by,


```
nn <- neuralnet(formula = FSTAT ~ AGE + SEX + CPK + SHO + CHF +
MIORD + MITYPE + YEAR + YRGRP + LENSTAY + DSTAT + LENFOL,
data = train.data, hidden = 2, err.fct = "ce", linear.output = FALSE)
```
- (10) Summary can be generated using `summary (nn)`
- (11) Visualize the neural network using `plot (nn)`

2.6.3 MatLab

- (1) Set the dataset path using `fullfile(path)` function
- (2) Load data as `ImageDatastore` using `imageDatastore(path)`
- (3) To view the data from the dataset, `imshow()` function is used
- (4) Divide the dataset into train and test set using `splitEachLabel()` function
- (5) Define the neural network model. E.g., Convolutional neural network by specifying `[imageInputLayer(dimension), convolution2dLayer(dimension), reluLayer, max-Pooling2dLayer (stride dimension), fully ConnectedLayer(10), softmaxLayer classificationLayer]`
- (6) Set the training option settings using the function `trainingOptions(optimization technique, maximum no. of epochs, initial learning rate)`

- (7) Train the model using function `trainNetwork (traindata, layers, optionsset)`
- (8) Prediction can be performed using the function `classify()`
- (9) Compute accuracy

2.6.4 Julia

- (1) Load the train and test data using `dataset_name.traindata()` and `dataset_name.-testdata()`
- (2) Add channel layer by `unsqueeze(traindata, layerno)` and `unsqueeze (testdata, layerno)`
- (3) Encode the labels by using the functions `onehotbatch(traindata, 0:9)` and `onehotbatch(testdata, 0:9)`
- (4) Create complete dataset by using `DataLoader(traindata, batchsize=size)`
- (5) To implement CNN use the function `chain(Conv(dimension), pad=2, stride=2, activation_function)`
- (6) Maxpooling can be implemented by using the function `maxpooling()` and average pooling can be implemented by using `GlobalMeanPool()`
- (7) Binary cross entropy loss is given by `crossentropy(model(x), y)`
- (8) Gradient descent optimizer is given by `Descent(learning rate)` and adam optimizer is given by `adam(learning rate)`
- (9) Train the model using @epochs number of epochs `Flux.train!(loss, b,w, train_data, opt)` where b is bias, w is weight, opt is the optimizer used
- (10) Compute accuracy

The next section discusses the open challenges for future research.

2.7 Open challenges for future research

This section discusses the open challenges of GAN for future research.

- Vanishing gradients is the problem that many GAN architectures suffer from. Firstly, the discriminator will be trained to classify between real and fake images. Then the generator will be trained but initially, the G will be generating the fake image which will be easily classified by D. The value of G will be 0 initially and so the slope will also be close to 0. Hence, the gradient cannot be calculated.
- Mode collapse is that the generator sometimes collapses and will always generate the same or similar fake images of one type, i.e., the generator generates limited varieties of fake samples. GAN architectures have to be designed in such a way that it will not suffer from this problem.
- In many GANs, it is very hard to achieve Nash equilibrium. Many numbers of epochs have to run to achieve Nash equilibrium. The research challenge is to develop a technique which will help G and D to achieve Nash equilibrium easily.

- The challenge is to train G and D simultaneously that they will fail to converge many times. Sometimes the G instead of attaining Nash equilibrium, it might oscillate between specific sample generated.
- How to increase the stability of training?
- What learning rate to set for G and D and also to check what is the effect of changing the learning rate is a challenge.
- Tuning hyperparameter, i.e., deciding on the value to set for hyperparameter is a challenge as it increases the training stability.
- New activation techniques can be proposed for activating the neurons in the network so that the learning can be stable.
- Training G and D is very hard. Optimizing the loss functions is very difficult and it needs many trial and errors. New optimization techniques can be proposed for still better fine-tuning of G and D , if the discriminator is not discriminating properly.
- If one network (either G or D) will not be trained properly, then the entire system performance will degrade.

2.8 Conclusion

This chapter provides an overview of the generative adversarial networks, classification of the GAN models based on learning and their pros and cons. The various applications of GAN in natural language processing, image generation, and translation are discussed. The various natural language processing and image datasets are listed. The evaluation metrics needed for assessing the GAN performance have also been discussed. The tools available for GAN research are also mentioned. Finally, the chapter summarizes the open challenges for the future research.

References

- [1] Y. Puy, Z. Gany, R. Henaoy, X. Yuan, C. Liy, A. Stevens, L. Cariny, Variational autoencoder for deep learning of images, labels and captions, in: 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 2016, pp. 1–9.
- [2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems 27, Montreal, Quebec, Canada, 2014, pp. 2672–2680.
- [3] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, New York, 2016.
- [4] I. Goodfellow, NIPS 2016 tutorial: generative adversarial networks, arXiv: 1701.00160 (2016).
- [5] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, arXiv: 1511.06434 (2015).
- [6] M. Roveri, Learning discrete-time Markov chains under concept drift, IEEE Trans. Neural Netw. Learn. Syst. 30 (9) (2019) 2570–2582.
- [7] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I.J. Goodfellow, A. Bergeron, N. Bouchard, Y. Bengio, Theano: new features and speed improvements, in: Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.