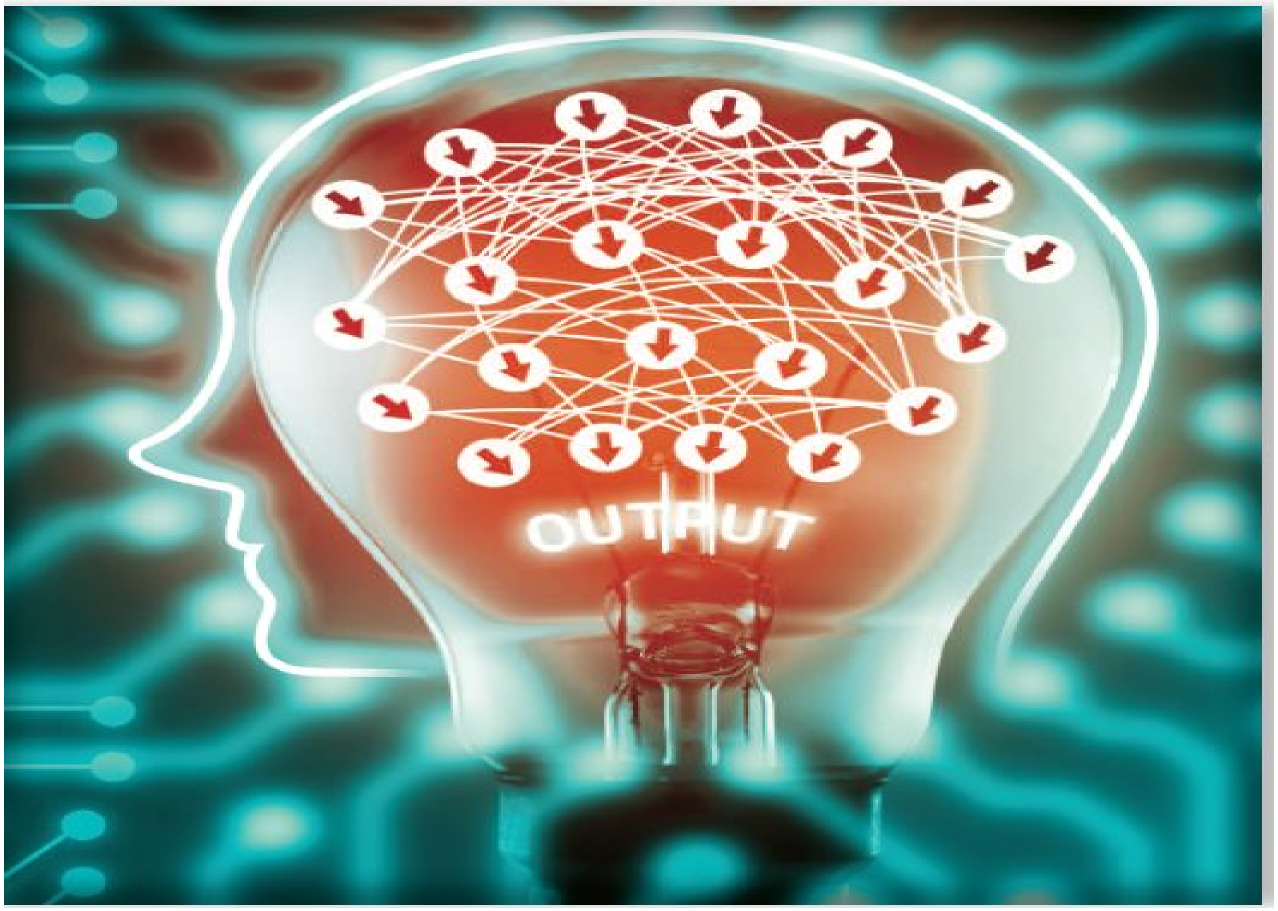
DSP Project Report

Generative Adversarial Networks



Aditya Dubey (1610110007)

Ganesh Rohit Basam (1610110127)

Group no. 7

Introduction:

Deep learning has changed the way we work, compute and has made our lives a lot easier. As Andrej Karpathy mentioned it is indeed the software 2.0, as we have taught machines to figure things out themselves. There are many existing deep learning techniques which can be ascribed to its prolific success. But no major impact has been created by deep generative models, which is due to their inability to approximate intractable probabilistic computations. Ian Goodfellow was able to find a solution that could sidestep these difficulties faced by generative models and created a new ingenious model called Generative Adversarial Networks. I believe it is astonishing when you look at the capabilities of a GAN. Before moving on to an introduction on GAN, let us look at some examples to understand what a GAN and its variants are capable of.

GAN is about creating, like drawing a portrait or composing a symphony. This is hard compared to other deep learning fields. For instance, it is much easier to identify a Monet painting than painting one. But it brings us closer to understand intelligence. Its importance leads us to thousands of GAN research papers written in recent years. In developing games, we hire many production artists to create animation. Some of the tasks are routine. By applying automation with GAN, we may one day focus ourselves to the creating sides rather than repeating routine tasks daily.

Nowadays, most of the applications of GANs are in the field of computer vision. Some of the applications include training semi-supervised classifiers, and generating high resolution images from low resolution counterparts.

Research Paper:

Deep learning for visual understanding: part 2
Generative Adversarial Networks
An overview

Volume: 35 Issue: 1

Antonia Creswell, Tom White, Vincent Dumoulin,

Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath

Generative adversarial networks (GANs) provide a way to learn deep representations without extensively annotated training data. They achieve this by deriving backpropagation signals through a competitive process involving a pair of networks. The representations that can be learned by GANs may be used in a variety of applications, including image synthesis, semantic image editing, style transfer, image super resolution, and classification. The aim of this review article is to provide an overview of GANs for the signal processing community, drawing on familiar analogies and concepts where possible. In addition to identifying different methods for training and constructing GANs, we also point to remaining challenges in their theory and application.

Introduction

GANs are an emerging technique for both semi supervised and unsupervised learning. They achieve this through implicitly modeling high-dimensional distributions of data. Proposed in 2014 [1], they can be characterized by training a pair of networks in competition with each other. A common analogy, apt for visual data, is to think of one network as an art forger and the other as an art expert. The forger, known in the GAN literature as the generator, G, creates forgeries, with the aim of making realistic images. The expert, known as the discriminator, D, receives both forgeries and real (authentic) images, and aims to tell them apart (see Figure 1). Both are trained simultaneously, and in competition with each other.

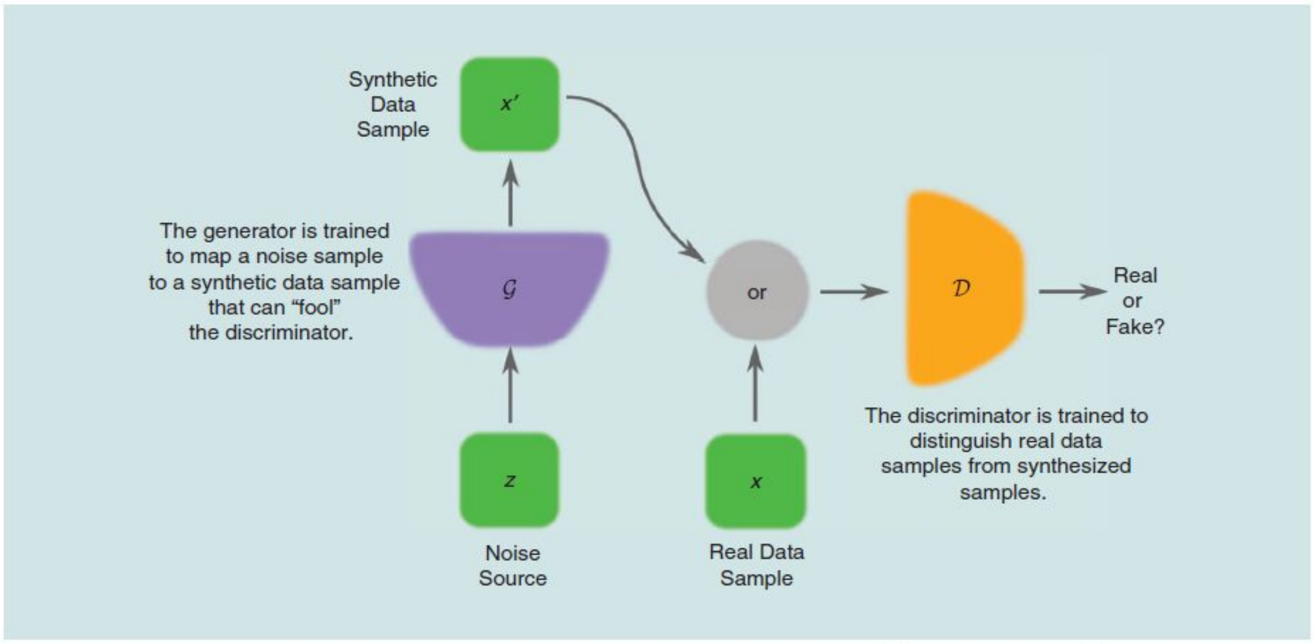

Synthetic Data Sample
x'
The generator is trained to map a noise sample to a synthetic data sample that can "fool" the discriminator.
$\mathcal{G}$
or
$\mathcal{D}$
Real or Fake?
The discriminator is trained to distinguish real data samples from synthesized samples.
z
Noise Source
x
Real Data Sample

FIGURE 1. The two models that are learned during the training process for a GAN are the discriminator ($\mathcal{D}$) and the generator ($\mathcal{G}$). These are typically implemented with neural networks, but they could be implemented by any form of differentiable system that maps data from one space to another; see article text for details.

Crucially, the generator has no direct access to real images—the only way it learns is through its interaction with the discriminator. The discriminator has access to both the synthetic samples and samples drawn from the stack of real images. The error signal to the discriminator is provided through the simple ground truth of knowing whether the image came from the real stack or from the generator. The same error signal, via the discriminator, can be used to train the generator, leading it toward being able to produce forgeries of better quality.

The networks that represent the generator and discriminator are typically implemented by multilayer networks consisting of convolutional and/or fully connected layers. The generator and discriminator networks must be differentiable, though it is not necessary for them to be directly invertible. If one considers the generator network as mapping from some representation space, called a latent space, to the space of the data (we shall focus on images), then we may express this more formally as G G : (z) " R x , where z ! R| | z is a sample from the latent space, x ! R| | x is an image and |·| denotes the number of dimensions.

In a basic GAN, the discriminator network, D, may be similarly characterized as a function that maps from image data to a probability that the image is from the real data distribution, rather than the generator distribution: D: (D x) (" 0 1 ,). For a fixed generator, G, the discriminator, D, may be trained to classify images as either being from the training data (real, close to one) or from a fixed generator (fake, close to zero). When the discriminator is optimal, it may be frozen, and the generator, G, may continue to be trained so as to lower the accuracy of the discriminator. If the generator distribution is able to match the real data distribution perfectly, then the discriminator will be maximally confused, predicting 0.5 for all inputs. In practice, the discriminator might not be trained until it is optimal; we explore the training process in more depth in the section "Training GANs."

On top of the interesting academic problems related to

training and constructing GANs, the motivations behind training GANs may not necessarily be the generator or the discriminator per se: the representations embodied by either of the pair of networks can be used in a variety of subsequent tasks. We explore the applications of these representations in the section "Application of GANs."

Preliminaries

Terminology

Generative models learn to capture the statistical distribution of training data, allowing us to synthesize samples from the learned distribution. On top of synthesizing novel data samples, which may be used for downstream tasks such as semantic image editing [2], data augmentation [3], and style transfer [4], we are also interested in using the representations that such models learn for tasks such as classification [5] and image retrieval [6].

We occasionally refer to fully connected and convolutional layers of deep networks; these are generalizations of perceptrons or spatial filter banks with nonlinear postprocessing. In all cases, the network weights are learned through backpropagation [7].

Notation

The GAN literature generally deals with multidimensional vectors and often represents vectors in a probability space by italics (e.g., latent space is z). In the field of signal processing, it is common to represent vectors by bold, lowercase symbols, and we adopt this convention to emphasize the multidimensional nature of variables. Accordingly, we will commonly refer to pdata(x) as representing the probability density function over a random vector x that lies in R| | x .We will use pg( ) x to denote the distribution of the vectors produced by the generator network of the GAN. We use the calligraphic symbols G and D to denote the generator and discriminator networks, respectively. Both networks have sets of parameters (weights), HD and HG, that are learned through optimization, during training.

As with all deep-learning systems, training requires that we have some clear objective function. Following the usualnotation, we use JG(HG; HD) and JD(HD; HG) to refer to the objective functions of the generator and discriminator, respectively. The choice of notation reminds us that the two objective functions are, in a sense, codependent on the evolving parameter sets HG and HD of the networks as they are iteratively updated. We shall explore this further in the section “Training GANs.” Finally, note that multidimensional gradients are used in the updates; we use G dH to denote the gradient operator with respect to the weights of the generator parameters and D dH to denote the gradient operator with respect to the weights of the discriminator. The expected gradients are indicated by the notation Ed• .

Capturing data distributions

A central problem of signal processing and statistics is that of density estimation: obtaining a representation—implicit or explicit, parametric or nonparametric—of data in the real world. This is the key motivation behind GANs. In the GAN literature, the term data generating distribution is often used to refer to the underlying probability density or probability mass function of observation data. GANs learn through implicitly computing some sort of similarity between the distribution of a candidate model and the distribution corresponding to real data (see Figure 2).

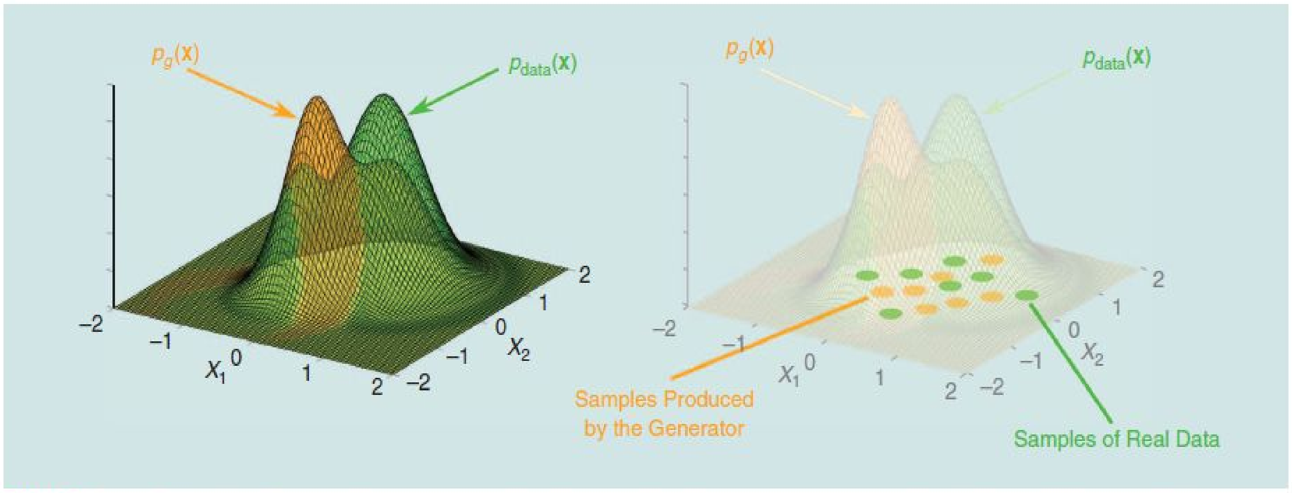


FIGURE 2. During GAN training, the generator is encouraged to produce a distribution of samples, $p_g(\mathbf{x})$ to match that of real data, $p_{\text{data}}(\mathbf{x})$. For an appropriately parameterized and trained GAN, these distributions will be nearly identical. The representations embodied by GANs are captured in the learned parameters (weights) of the generator and discriminator networks.

Why bother with density estimation at all? The answer lies at the heart of—arguably—many problems of visual inference, including image categorization, visual object detection and recognition, object tracking, and object registration. In principle, through Bayes' theorem, all inference problems of computer vision can be addressed through estimating conditional density functions, possibly indirectly in the form of a model that learns the joint distribution of variables of interest and the observed data. The difficulty we face is that likelihood functions for high-dimensional, real-world image data are difficult to construct. While GANs don't explicitly provide a way of evaluating density functions, for a generator-discriminator pair of suitable capacity, the generator implicitly captures the distribution of the data.

Related work

One may view the principles of generative models by making comparisons with standard techniques in signal processing and data analysis. For example, signal processing makes wide use of the idea of representing a signal as the weighted combination of basis functions. Fixed basis functions underlie standard techniques such as Fourier-based and wavelet representations. Data-driven approaches to constructing basis functions

can be traced back to the Hotelling [8] transform, rooted in Pearson's observation that principal components minimize a reconstruction error according to a minimum squared error criterion. Despite its wide use, standard principal component analysis (PCA) does not have an overt statistical model for the observed data, though it has been shown that the bases of PCA

may be derived as a maximum likelihood parameter estimation problem.

Despite wide adoption, PCA is limited—the basis functions emerge as the eigenvectors of the covariance matrix over observations of the input data, and the mapping from the representation space back to signal or image space is linear. So, we have both a shallow and a linear mapping, limiting the complexity of the model and, hence, of the data, that can be represented.

Independent component analysis (ICA) provides another level up in sophistication, in which the signal components no longer need to be orthogonal; the mixing coefficients used to blend components together to construct examples of data are merely considered to be statistically independent. ICA has various formulations that differ in their objective functions used during estimating signal components or in the generative model that expresses how signals or images are generated from those components. A recent innovation explored through ICA is noise contrastive estimation (NCE); this may be seen as approaching the spirit of GANs [9]: the objective function for learning independent components compares a statistic applied to noise with that produced by a candidate generative model [10]. The original NCE approach did not include updates to the generator.

What other comparisons can be made between GANs and the standard tools of signal processing? For PCA, ICA, Fourier, and wavelet representations, the latent space of GANs is, by analogy, the coefficient space of what we commonly refer to as transform space. What sets GANs apart from these standard tools of signal processing is the level of complexity of the models that map vectors from latent space to image space. Because the generator networks contain nonlinearities, and can be of almost arbitrary depth, this mapping—as with many other deep-learning approaches—can be extraordinarily complex.

With regard to deep image-based models, modern approaches to generative image modeling can be grouped into explicit and implicit density models. Explicit density models are either tractable (change of variables models, autoregressive models) or intractable (directed models trained with variational inference, undirected models trained using Markov chains). Implicit density models capture the statistical distribution of the data through a generative process that makes use of either ancestral sampling [11] or Markov chain-based sampling. GANs fall into the directed implicit model category. A more detailed overview and relevant papers can be found in [12].

GAN architectures

Fully connected GANs

The first GAN architectures used fully connected neural networks for both the generator and discriminator [1]. This type of architecture was applied to relatively simple image datasets: MNIST (handwritten digits), CIFAR-10 (natural images), and the Toronto Face Dataset (TFD).

Convolutional GANs

Going from fully connected to convolutional neural networks (CNNs) is a natural extension, given that CNNs are extremely well suited to image data. Early experiments conducted on CIFAR-10 suggested that it was more difficult to train generator and discriminator networks using CNNs with the same level of capacity and representational power as those used for supervised learning.

The Laplacian pyramid of adversarial networks (LAPGAN) [13] offered one solution to this problem, by decomposing the generation process using multiple scales: a ground-truth image is itself decomposed into a Laplacian pyramid and a conditional, convolutional GAN is trained to produce each layer given the one above.

Additionally, Radford et al. [5] proposed a family of network architectures called deep convolutional GAN (DCGAN), which allows training a pair of deep convolutional generator and discriminator networks. DCGANs make use of strided and fractionally strided convolutions, which allow the spatial downsampling and upsampling operators to be learned during training. These operators handle the change in sampling rates and locations, a key requirement in mapping from image space to possibly lower dimensional latent space, and from image space to a discriminator. Further details of the DCGAN architecture and training are presented in the section "Training Tricks."

As an extension to synthesizing images in two dimensions, Wu et al.[14] presented GANs that were able to synthesize three-dimensional (3-D) data samples using volumetric convolutions. Wu et al. [14] synthesized novel objects including chairs, a table, and cars; in addition, they also presented a method to map from two-dimensional (2-D) images to 3-D versions of objects portrayed in those images.

Conditional GANs

Mirza et al. [15] extended the (2-D) GAN framework to the conditional setting by making both the generator and the discriminator networks class-conditional (Figure 3). Conditional GANs have the advantage of being able to provide better representations for multimodal data generation. A parallel can be drawn between conditional GANs and InfoGAN [16], which decomposes the noise source into an incompressible source and a "latent code," attempting to discover latent factors of variation by maximizing the mutual information between the latent code and the generator's output. This latent code can be used to discover object classes in a purely unsupervised fashion, although it is not strictly necessary that the latent code be categorical. The representations learned by InfoGAN appear to be semantically meaningful, dealing with complex inter tangled factors in image appearance, including variations in pose, lighting, and emotional content of facial images [16].

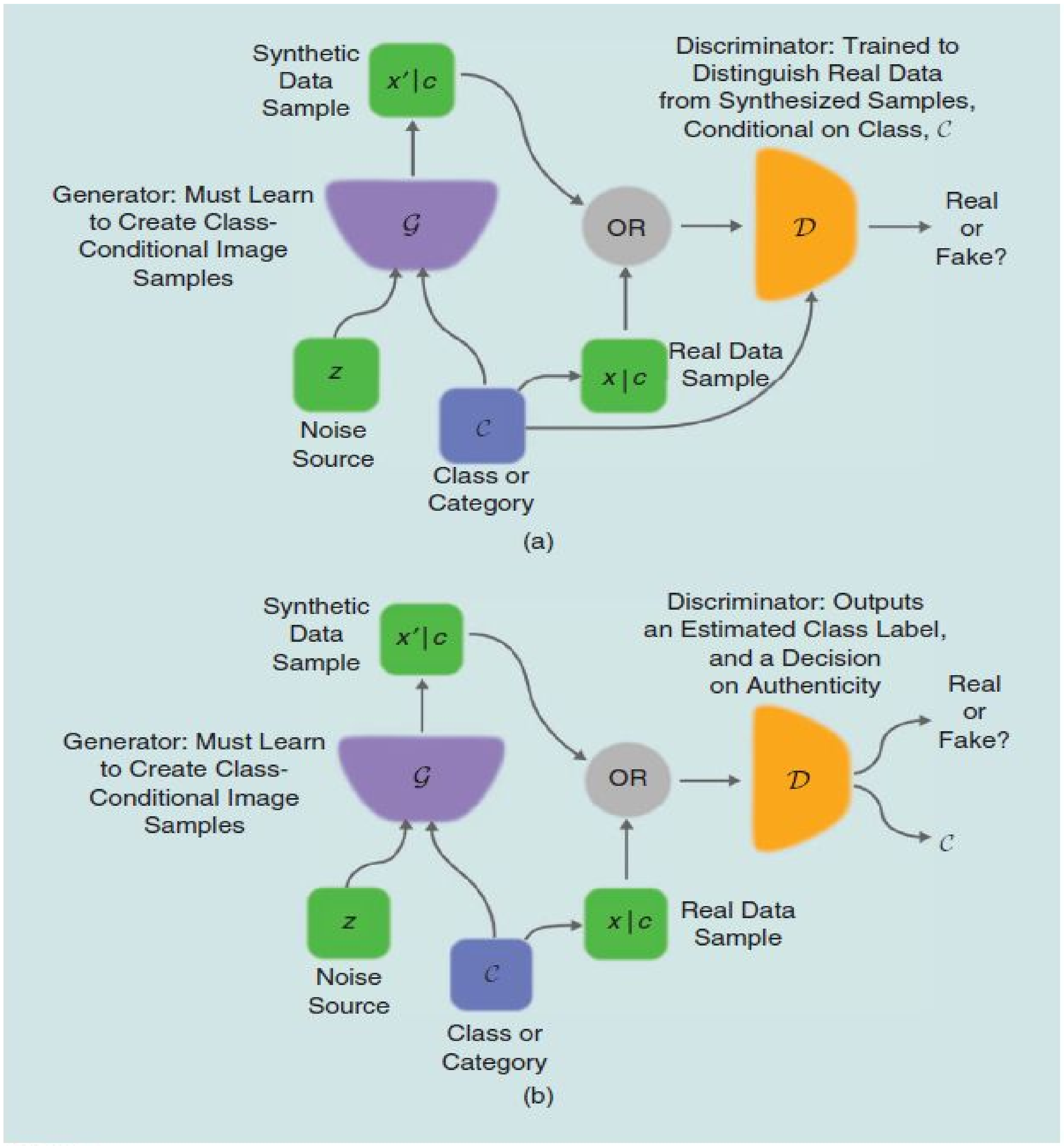

Synthetic Data Sample
$x'|c$
Discriminator: Trained to Distinguish Real Data from Synthesized Samples, Conditional on Class, $\mathcal{C}$
Generator: Must Learn to Create Class-Conditional Image Samples
$\mathcal{G}$
OR
$\mathcal{D}$
Real or Fake?
z
Noise Source
$\mathcal{C}$
Class or Category
$x|c$
Real Data Sample
(a)
Synthetic Data Sample
$x'|c$
Discriminator: Outputs an Estimated Class Label, and a Decision on Authenticity
Real or Fake?
Generator: Must Learn to Create Class-Conditional Image Samples
$\mathcal{G}$
OR
$\mathcal{D}$
$\mathcal{C}$
z
Noise Source
$\mathcal{C}$
Class or Category
$x|c$
Real Data Sample
(b)

FIGURE 3. (a) The conditional GAN, proposed by Mirza et al. [15] performs class-conditional image synthesis; the discriminator performs class-conditional discrimination of real from fake images. (b) The InfoGAN [16], on the other hand, has a discriminator network that also estimates the class label.

GANs with inference models

In their original formulation, GANs lacked a way to map a given observation, x, to a vector in latent space—in the GAN literature, this is often referred to as an inference mechanism.Several techniques have been proposed to invert the generator of pretrained GANs [17], [18]. The independently proposed adversarially learned inference (ALI) [19] and bidirectional GANs (BiGANs) [20] provide simple but effective extensions, introducing an inference network in which the discriminators examine joint (data, latent) pairs.

In this formulation, the generator consists of two networks: the "encoder" (inference network) and the "decoder." They are jointly trained to fool the discriminator. The discriminator itself receives pairs of (x, z) vectors (see Figure 4), and has to determine which pair constitutes a genuine tuple consisting of real image sample and its encoding, or a fake image sample and the corresponding latent-space input to the generator.

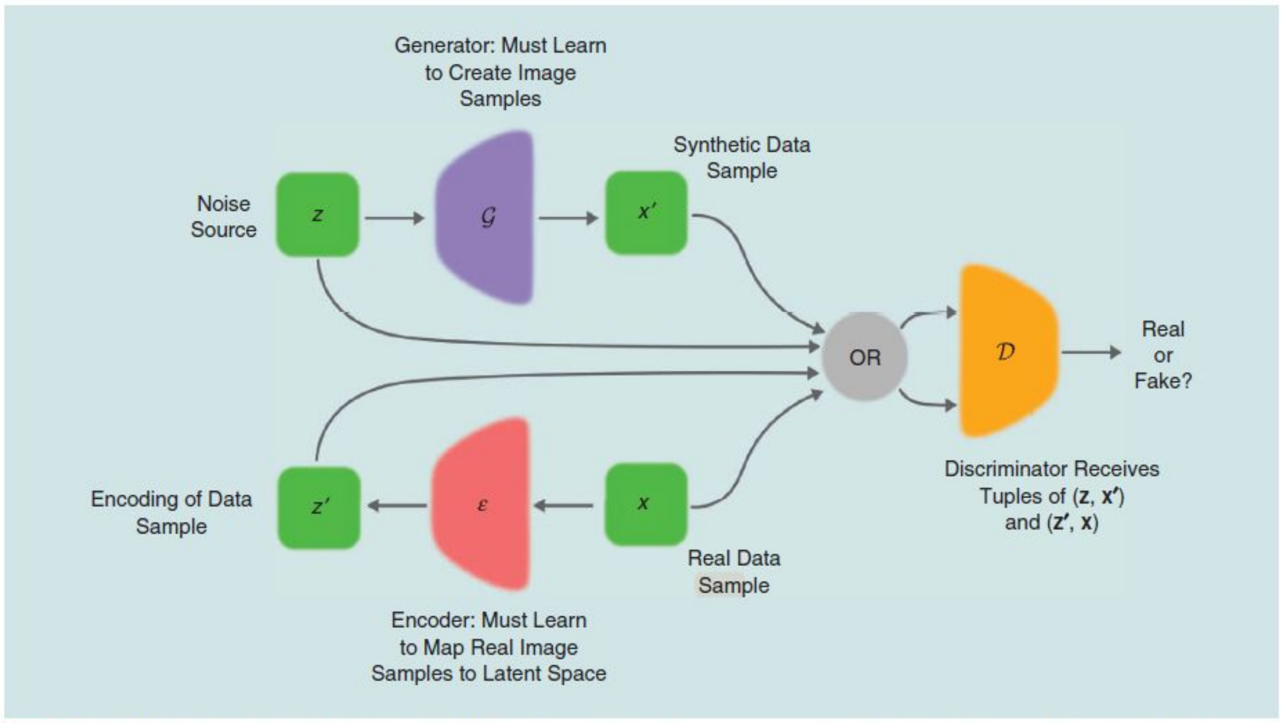

Generator: Must Learn to Create Image Samples
Synthetic Data Sample
Noise Source
z
$\mathcal{G}$
x′
OR
$\mathcal{D}$
Real or Fake?
Encoding of Data Sample
z′
ε
x
Real Data Sample
Discriminator Receives Tuples of (z, x′) and (z′, x)
Encoder: Must Learn to Map Real Image Samples to Latent Space

FIGURE 4. The ALI/BiGAN structure [19], [20] consists of three networks. One of these serves as a discriminator, another maps the noise vectors from latent space to image space (decoder, depicted as a generator $\mathcal{G}$ in the figure), with the final network (encoder, depicted as $\mathcal{E}$) mapping from image space to latent space.

Ideally, in an encoding-decoding model, the output, referred to as a reconstruction, should be similar to the input. Typically, the fidelity of reconstructed data samples synthesized using an ALI/BiGAN are poor. The fidelity of samples may be improved with an additional adversarial cost on the distribution of data samples and their reconstructions [21].

Adversarial autoencoders

Autoencoders are networks, composed of an encoder and decoder, which learn to map data to an internal latent representation and out again. That is, they learn a deterministic mapping (via the encoder) from a data space, e.g., images, into a latent or representation space, and a mapping (via the decoder) from the latent space back to data space. The composition of these two mappings results in a reconstruction, and the two mappings are trained such that a reconstructed image is as close as possible to the original.

Autoencoders are reminiscent of the perfect-reconstruction filter banks that are widely used in image and signal processing. However, autoencoders generally learn nonlinear mappings in both directions. Further, when implemented with deep networks, the possible architectures that can be used to implement autoencoders are remarkably flexible. Training can be unsupervised, with backpropagation being applied between the reconstructed image and the original to learn the parameters of both the encoder and the decoder.

As suggested previously, one often wants the latent space to have a useful organization. Additionally, one may want to perform feed-forward, ancestral sampling [11] from an autoencoder. Adversarial training provides a route to achieve these two goals. Specifically, adversarial training may be applied between the latent space and a desired prior distribution on the latent space (latent-space GAN). This results in a combined loss function [22] that reflects both the reconstruction error and a measure of how different the distribution of the prior is from that produced by a candidate encoding network. This approach is akin to a variational autoencoder (VAE) [23] for which the latent-space GAN plays the role of the Kullback–Leibler (KL)-divergence term of the loss function.

Mescheder et al. [24] unified VAEs with adversarial training in the form of the adversarial variational Bayes (AVB) framework. Similar ideas were presented in [12]. AVB tries to optimize the same criterion as that of VAEs, but uses an adversarial training objective rather than the KL divergence.

Training GANs

Introduction

The training of GANs involves both finding the parameters of a discriminator that maximize its classification accuracy and finding the parameters of a generator that maximally confuse the discriminator. This training process is summarized in Figure 5.

The cost of training is evaluated using a value function, V^G,Dh that depends on both the generator and the discriminator. The training involves solving

$$\max_{\mathcal{D}} \min_{\mathcal{G}} V(\mathcal{G}, \mathcal{D}),$$

where

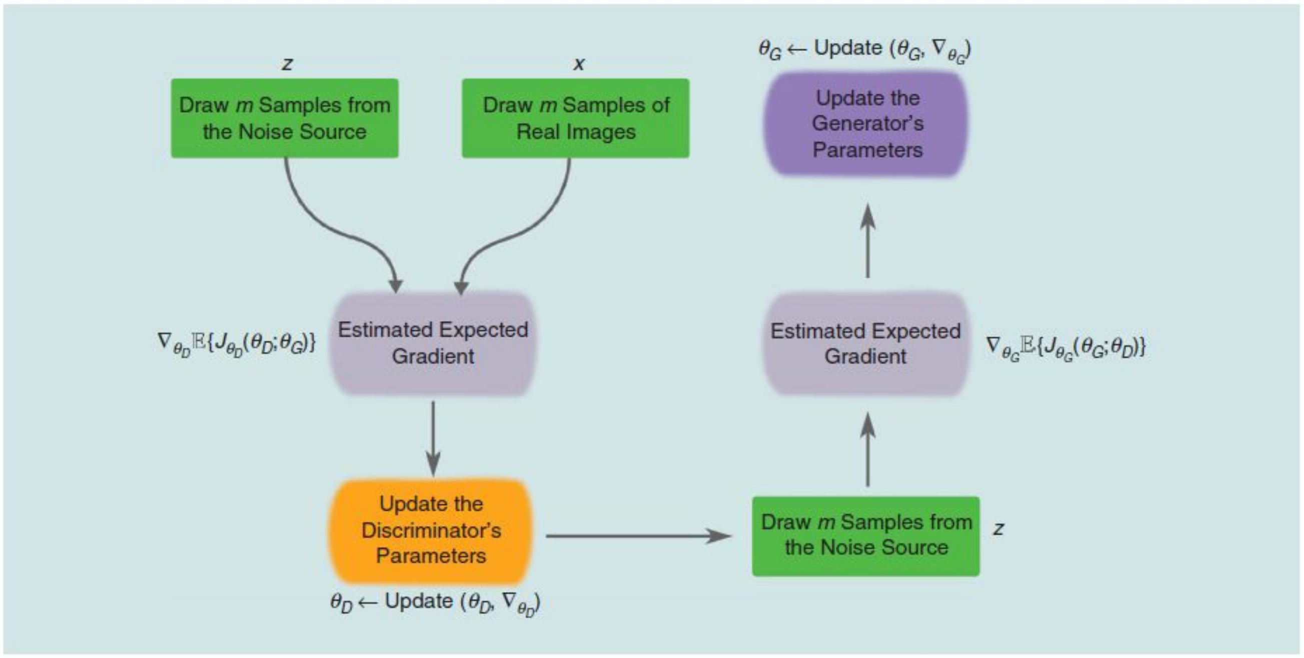
$$V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \log \mathcal{D}(\mathbf{x}) + \mathbb{E}_{p_g(\mathbf{x})} \log (1 - \mathcal{D}(\mathbf{x})).$$

During training, the parameters of one model are updated, while the parameters of the other are fixed. Goodfellow et al. [1] show that, for a fixed generator, there is a unique optimal discriminator, D* (x) p (x) p (x) p (x) .

= data ^ data + g h

They also show that the generator, G, is optimal when pg (x) =pdata (x), which is equivalent to the optimal discriminator predicting 0.5 for all samples drawn from x. In other words, the generator is optimal when the discriminator, D, is maximally confused and cannot distinguish real samples from ones that are fake.

Ideally, the discriminator is trained until optimal with respect to the current generator; then the generator is again updated. However in practice, the discriminator might not be trained until optimal but rather may only be trained for a small number of iterations, and the generator is updated simultaneously with the discriminator. Further, an alternate, non saturating training criterion is typically used for the generator, using maxGlogD(G(z)) rather than minGlog(1 -D(G(z))).



z
Draw m Samples from the Noise Source
x
Draw m Samples of Real Images
$\theta_G \leftarrow \text{Update}(\theta_G, \nabla_{\theta_G})$
Update the Generator's Parameters
$\nabla_{\theta_D}\mathbb{E}\{J_{\theta_D}(\theta_D;\theta_G)\}$
Estimated Expected Gradient
Estimated Expected Gradient
$\nabla_{\theta_G}\mathbb{E}\{J_{\theta_G}(\theta_G;\theta_D)\}$
Update the Discriminator's Parameters
$\theta_D \leftarrow \text{Update}(\theta_D, \nabla_{\theta_D})$
Draw m Samples from the Noise Source
z

FIGURE 5. The main loop of GAN training. Novel data samples, $\mathbf{x}'$, may be drawn by passing random samples, $\mathbf{z}$, through the generator network. The gradient of the discriminator may be updated k times before updating the generator.

Despite the theoretical existence of unique solutions, GAN training is challenging and often unstable for several reasons[5], [25], [26]. One approach to improving GAN training is to asses the empirical "symptoms" that might be experienced during training. These symptoms include:

■■ difficulties in getting the pair of models to converge [5]

■■ the generative model "collapsing" to generate very similar samples for different inputs [25]

■■ the discriminator loss converging quickly to zero [26], providing no reliable path for gradient updates to the generator.

Several authors suggested heuristic approaches to address these issues [1], [25]; these are discussed in the next section. Early attempts to explain why GAN training is unstable were proposed by Goodfellow and Salimans et al. [1], [25], who observed that gradient descent methods typically used for updating both the parameters of the generator and discriminator are inappropriate when the solution to the optimization problem posed by GAN training actually constitutes a saddle point. Salimans et al. provided a simple example that shows this [25]. However, stochastic gradient descent is often used to update neural networks and there are well-developed machine-learning programming environments that make it easy to construct and update networks using stochastic gradient descent.

Although an early theoretical treatment [1] showed that the generator is optimal when pg (x) =pdata (x), a very neat result with a strong underlying intuition, the real data samples reside on a manifold that sits in a high-dimensional space of possible representations. For instance, if color image samples are of size N # N # 3 with pixel values [0, R+] 3, the space that may be represented—which we can call X—is of dimensionality 3N2, with each dimension taking values between zero and the maximum measurable pixel intensity. The data samples in the support of pdata, however, constitute the manifold of the real data associated with some particular problem, typically occupying a very small part of the total space, X. Similarly, the samples produced by the generator should also occupy only a small portion of X.

Arjovsky et al. [26] showed that the support pg (x) and pdata (x) lie in a lower-dimensional space than that corresponding to X. The consequence of this is that pg (x) and pdata (x) may have no overlap, and so there exists a nearly trivial discriminator that is capable of distinguishing real samples, x~pdata (x) from fake samples, x~pg (x) with 100% accuracy. In this case, the discriminator error quickly converges to zero. Parameters of the generator may only be updated via the discriminator, so when this happens, the gradients used for updating parameters of the generator also converge to zero and may no longer be useful for updates to the generator. Arjovsky et al.'s explanations account for several of the symptoms related to GAN training [26].

Goodfellow et al. [1] also showed that when D is optimal, training G is equivalent to minimizing the Jensen–Shannon (JS) divergence between pg (x) and pdata (x) . If D is not optimal, the update may be less meaningful or inaccurate. This theoretical insight has motivated research into cost functions based on alternative distances. Several of these are explored in the section "Alternative Formulations."

Training tricks

One of the first major improvements in the training of GANs for generating images were the DCGAN architectures proposed by Radford et al. [5]. This work was the result of an extensive exploration of CNN architectures previously used in computer vision, and it resulted in a set of guidelines for constructing and training both the generator and discriminator. In the section "Convolutional GANs," we alluded to the importance of strided and fractionally strided convolutions [27], which are key components of the architectural design. This allows both the generator and the discriminator to learn good upsampling and downsampling operations, which may contribute to improvements in the quality of image synthesis. More specifically to training, batch normalization [28] was recommended for use in both networks to stabilize training in deeper models. Another suggestion was to minimize the number of fully connected layers used to increase the feasibility of training deeper models. Finally, Radford et al. [5] showed that using leaky rectifying linear units (ReLUs) activation functions between the intermediate layers of the discriminator gave superior performance over using regular ReLUs.

Later, Salimans et al. [25] proposed further heuristic approaches for stabilizing the training of GANs. The first, feature matching, changes the objective of the generator slightly to increase the amount of information available. Specifically, the discriminator is still trained to distinguish between real and fake samples, but the generator is now trained to match the discriminator's expected intermediate activations (features) of its fake samples with the expected intermediate activations of the real samples. The second, minibatch discrimination, adds an extra input to the discriminator, which is a feature that encodes the distance between a given sample in a minibatch and the other samples. This is intended to prevent mode collapse, as the discriminator can easily tell if the generator is producing the same outputs.

A third trick, heuristic averaging, penalizes the network parameters if they deviate from a running average of previous values, which can help convergence to an equilibrium. The fourth, virtual batch normalization, reduces the dependency of one sample on the other samples in the minibatch by calculating the batch statistics for normalization with the sample placed within a reference minibatch that is fixed at the beginning of training.

Finally, one-sided label smoothing makes the target for the discriminator 0.9 instead of one, smoothing the discriminator's classification boundary, hence preventing an overly confident discriminator that would provide weak gradients for the generator. Sønderby et al. [29] advanced the idea of challenging the discriminator by adding noise to the samples before feeding them into the discriminator. Sønderby et al. [29] argued that one-sided label smoothing biases the optimal discriminator, while their technique, instance noise, moves the manifolds of the real and fake samples closer together, at the same time preventing the discriminator easily finding a discrimination boundary that completely separates the real and fake samples. In practice, this can be implemented by adding Gaussian noise to both the synthesized and real images, annealing the standard deviation over time. The same process was independently proposed by Arjovsky et al. [26].

Alternative formulations

The first part of this section considers other information-theoretic interpretations and generalizations of GANs. The second part looks at alternative cost functions that aim to directly address the problem of vanishing gradients.

Generalizations of the GAN cost function

Nowozin et al. [30] showed that GAN training may be generalized to minimize not only the JS divergence, but an estimate of f-divergences; these are referred to as f-GANs. The f-divergences include well-known divergence measures such as the KL-divergence. Nowozin et al. showed that the f-divergence may be approximated by applying the Fenchel conjugates of the desired f-divergence to samples drawn from the distribution of generated samples, after passing those samples through a discriminator[30]. They provide a list of Fenchel conjugates for commonly used f-divergences, as well as activation functions that may be used in the final layer of the generator network, depending on the choice of f-divergence. Having derived the generalized cost functions for training the generator and discriminator of an f-GAN, Nowozin et al. [30] observe that, in its raw form, maximizing the generator objective is likely to lead to weak gradients, especially at the start of training, and proposed an alternative cost function for updating the generator, which is less likely to saturate at the beginning of training. Nowozin et al. proposed that when the discriminator is trained, the derivative of the f-divergence on the ratio of the real and fake data distributions is estimated, while when the generator is trained only an estimate of the f-divergence is minimized. Uehara et al. [31] extend the f-GAN further, where in the discriminator step the ratio of the distributions of real and fake data are predicted, and in the generator step the f-divergence is directly minimized. Alternatives to the JS-divergence are also covered by Goodfellow [12].

Alternative cost functions to prevent vanishing gradients

Arjovsky et al. [32] proposed the Wasserstein GAN (WGAN), a GAN with an alternative cost function that is derived from an approximation of the Wasserstein distance. Unlike the original GAN cost function, the WGAN is more likely to provide gradients that are useful for updating the generator. The cost function derived for the WGAN relies on the discriminator, which they refer to as the critic, being a k-Lipschitz continuous function; practically, this may be implemented by simply clipping the parameters of the discriminator. However, more recent research [33] suggested that weight clipping adversely reduces the capacity of the discriminator model, forcing it to learn simpler functions. Gulrajani et al. [33] proposed an improved method for training the discriminator for a WGAN, by penalizing the norm of discriminator gradients with respect to data samples during training, rather than performing parameter clipping.

A brief comparison of GAN variants

GANs allow us to synthesize novel data samples from random noise, but they are considered difficult to train due partially to vanishing gradients. All GAN models that we have discussed in this article require careful hyperparameter tuning and model selection for training. However, perhaps the easier models to train are the adversarial autoencoder (AAE) and the WGAN. The AAE is relatively easy to train because the adversarial loss is applied to a fairly simple distribution in lower dimensions (than the image data). The WGAN [33], is designed to be easier to train, using a different formulation of the training objective that does not suffer from the vanishing gradient problem. The WGAN may also be trained successfully even without batch normalization; it is also less sensitive to the choice of nonlinearities used between convolutional layers.

Samples synthesized using a GAN or WGAN may belong to any class present in the training data. Conditional GANs provide an approach to synthesizing samples with user specified content.

It is evident from various visualization techniques (Figure6) that the organization of the latent space harbors some meaning, but vanilla GANs do not provide an inference model to allow data samples to be mapped to latent representations. Both BiGANs and ALI provide a mechanism to map image data to a latent space (inference), however, reconstruction quality suggests that they do not necessarily faithfully encode and decode samples. A very recent development shows that ALI may recover encoded data samples faithfully [21]. However, this model shares a lot in common with the AVB and AAE. These are autoencoders, similar to VAEs, where the latent space is regularized using adversarial training rather than a KL-divergence between encoded samples and a prior.

The structure of latent space

GANs build their own representations of the data they are trained on, and in doing so produce structured geometric vector spaces for different domains. This is a quality shared with other neural network models, including VAEs [23], as well as linguistic models such as word2vec [34]. In general, the domain of the data to be modeled is mapped to a vector space, which has fewer dimensions than the data space, forcing the model to discover interesting structure in the data and represent it efficiently. This latent space is at the “originating” end of the generator network, and the data at this level of representation (the latent space) can be highly structured and may support high-level semantic operations [5]. Examples include the rotation of faces from trajectories through latent space, as well as image analogies that have the effect of adding visual attributes such as eyeglasses onto a “bare” face.

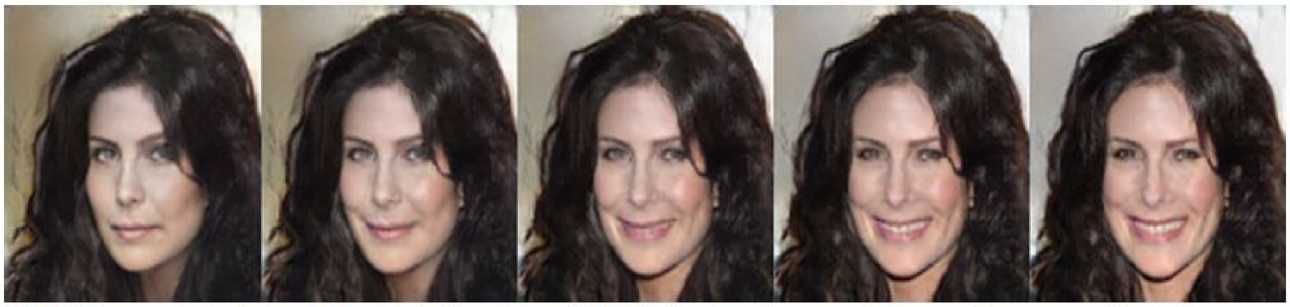


FIGURE 6. An example of applying a “smile vector” with an ALI model [19]. The first image is an example of an unsmiling woman and the last is an example of a woman smiling. A $\mathbf{z}$ value for the first image is inferred, $\mathbf{z}_1$ and for the last, $\mathbf{z}_2$. Interpolating along a vector that connects $\mathbf{z}_1$ and $\mathbf{z}_2$, gives $\mathbf{z}$ values that may be passed through a generator to synthesize novel samples. Note the implication: a displacement vector in latent space traverses smile “intensity” in image space. (Figure used courtesy of Tom White.)

All (vanilla) GAN models have a generator that maps data from the latent space into the space to be modeled, but many GAN models have an encoder that additionally supports the inverse mapping [19], [20]. This becomes a powerful method for exploring and using the structured latent space of the GAN network. With an encoder, collections of labeled images can be mapped into latent spaces and analyzed to discover “concept vectors” that represent high-level attributes such as “smiling” or “wearing a hat.” These vectors can be applied at scaled offsets in latent space to influence the behavior of the generator (Figure 6). Similar to using an encoding process to model the distribution of latent samples, Gurumurthy et al. [35] propose modeling the latent space as a mixture of Gaussians and learning the mixture components that maximize the likelihood of generated data samples under the data generating distribution.

Applications of GANs

Discovering new applications for adversarial training of deep networks is an active area of research. We examine a few computer vision applications that have appeared in the literature and been subsequently refined. These applications were chosen to highlight some different approaches to using GAN based representations for image manipulation, analysis, or characterization and do not fully reflect the potential breadth of application of GANs.

Using GANs for image classification places them within the broader context of machine learning and provides a useful quantitative assessment of the features extracted in unsupervised learning. Image synthesis remains a core GAN capability and is especially useful when the generated image can be subject to pre-existing constraints. Superresolution [36]–[38] offers an example of how an existing approach can be supplemented with an adversarial loss component to achieve higher-quality results. Finally, image-to-image translation demonstrates how GANs offer a general-purpose solution to a family of tasks that require automatically converting an input image into an output image.

Classification and regression

After GAN training is complete, the neural network can be reused for other downstream tasks. For example, outputs of the convolutional layers of the discriminator can be used as a feature extractor, with simple linear models fitted on top of these features using a modest quantity of (image, label) pairs[5], [25]. The quality of the unsupervised representations within a DCGAN network have been assessed by applying a regularized L2-SVM classifier to a feature vector extracted from the (trained) discriminator [5]. Good classification scores were achieved using this approach on both supervised and semi supervised data sets, even those that were disjoint from the original training data.

The quality of the data representation may be improved when adversarial training includes jointly learning an inference mechanism such as with ALI [19]. A representation vector was built using last three hidden layers of the ALI encoder, a similar L2-SVM classifier, yet achieved a misclassification rate significantly lower than the DCGAN [19].

Additionally, ALI has achieved state-of-the art classification results when label information is incorporated into the training routine. When labeled training data is in limited supply, adversarial training may also be used to synthesize more training samples. Shrivastava et al. [39] use GANs to refine synthetic images while maintaining their annotation information. By training models only on GAN-refined synthetic images (i.e., no real training data) Shrivastava et al. [39] achieved state-of-the-art performance on pose- and gaze-estimation tasks. Similarly, good results were obtained for gaze estimation and prediction using a spatiotemporal GAN architecture [40]. In some cases, models trained on synthetic data do not generalize well when applied to real data [3]. Bousmalis et al. [3] propose to address this problem by adapting synthetic samples from a source domain to match a target domain using adversarial training. Additionally, Liu et al. [41] propose using multiple GANs—one per domain—with tied weights to synthesize pairs of corresponding images samples from different domains. Because the quality of generated samples is hard to quantitatively judge across models, classification tasks are likely to remain an important quantitative tool for performance assessment of GANs, even as new and diverse applications in computer vision are explored.

Image synthesis

Much of the recent GAN research focuses on improving the quality and utility of the image-generation capabilities. The LAPGAN model introduced a cascade of convolutional networks within a Laplacian pyramid framework to generate images in a coarse-to-fine fashion [13]. A similar approach is used by Huang et al. [42] with GANs operating on intermediate representations rather than lower-resolution images.

LAPGAN also extended the conditional version of the GAN model where both G and D networks receive additional label information as input; this technique has proved useful and is now a common practice to improve image quality. This idea of GAN conditioning was later extended to incorporate natural language. For example, Reed et al. [43] used a GAN architecture to synthesize images from text descriptions, which one might describe as reverse captioning. For example, given a text caption of a bird such as "white with some black on its head and wings and a long, orange beak," the trained GAN can generate several plausible images that match the description.

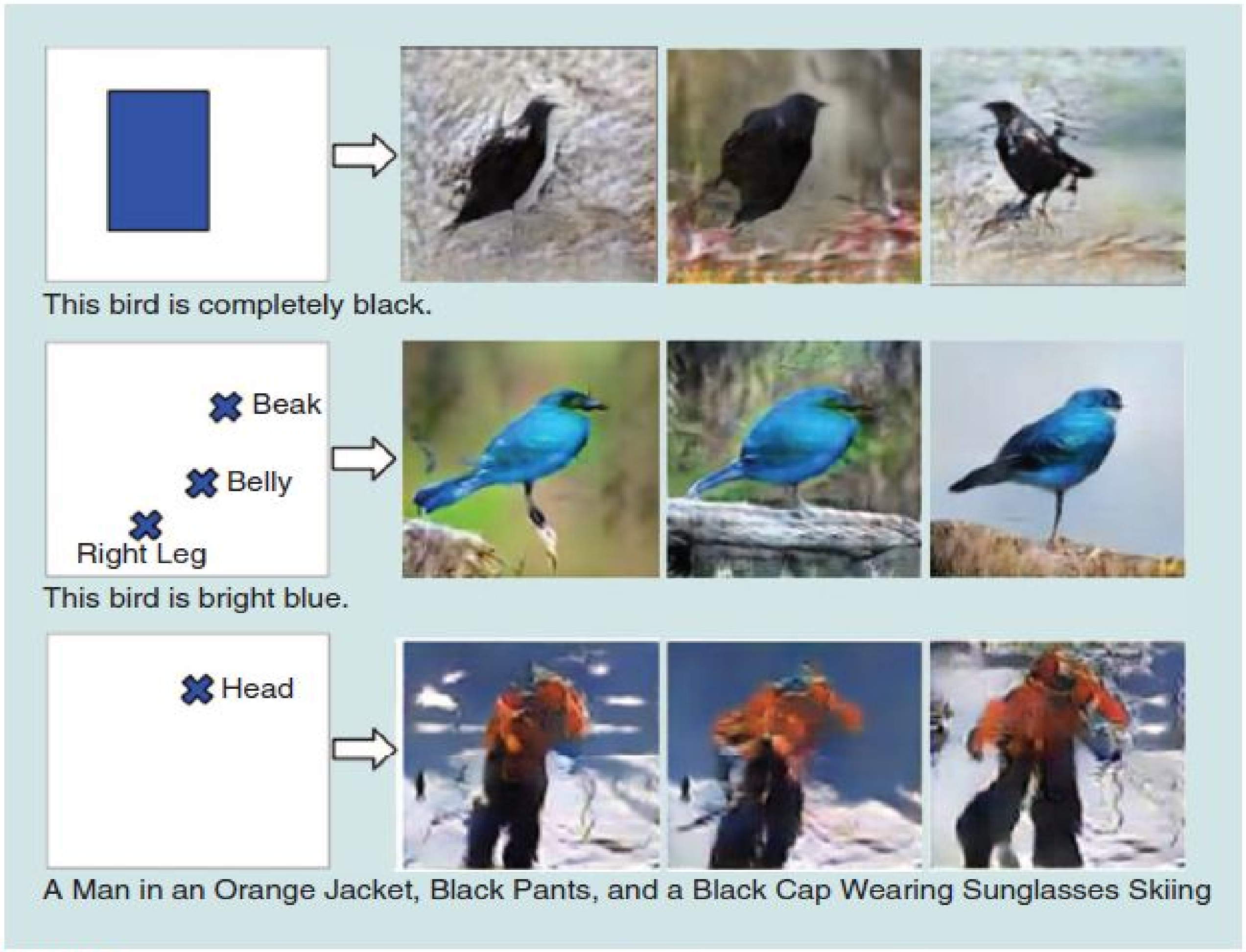

This bird is completely black.
Beak
Belly
Right Leg
This bird is bright blue.
Head
A Man in an Orange Jacket, Black Pants, and a Black Cap Wearing Sunglasses Skiing

FIGURE 7. Examples of image synthesis using the GAWWN. In the GAWWN, images are conditioned on both text descriptions and image location specified as either a keypoint or bounding box. (Figure reproduced from [44] with permission.)

In addition to conditioning on text descriptions, the generative adversarial what-where network (GAWWN) conditions on image location [44]. The GAWWN system supported an interactive interface in which large images could be built up incrementally with textual descriptions of parts and user supplied bounding boxes (Figure 7).

Conditional GANs not only allow us to synthesize novel samples with specific attributes, they also allow us to develop tools for intuitively editing images; e.g., changing the hairstyle of a person in an image, making them wear glasses, or editing the image so they appear younger [35]. Additional applications of GANs to image editing include work by Zhu and Brock et al. [2], [45].

Image-to-image translation

Conditional adversarial networks are well suited for translating an input image into an output image, which is a recurring theme in computer graphics, image processing, and computer vision. The pix2pix model offers a general-purpose solution to this family of problems [46]. In addition to learning the mapping from input image to output image, the pix2pix model also constructs a loss function to train this mapping.This model has demonstrated effective results for different problems of computer vision that had previously required separate machinery, including semantic segmentation, generating maps from aerial photos, and colorization of black and white images. Wang et al. present a similar idea, using GANs to first synthesize surface-normal maps (similar to depth maps) and then map these images to natural scenes.

CycleGAN [4] extends this work by introducing a cycle consistency loss that attempts to preserve the original image after a cycle of translation and reverse translation. In this formulation, matching pairs of images are no longer needed for training. This makes data preparation much simpler, and opens the technique to a larger family of applications. For example, artistic style transfer [47] renders natural images in the style of artists, such as Picasso or Monet, by simply being trained on an unpaired collection of paintings and natural images (Figure 8).

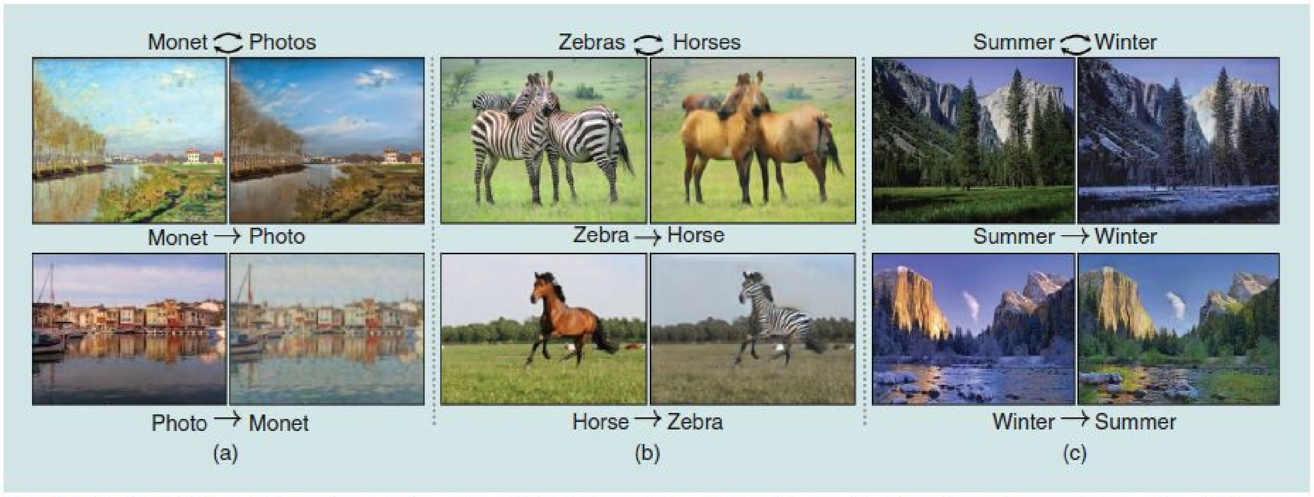


FIGURE 8. The CycleGAN model learns image to image translations between two unordered image collections. Shown here are the examples of bidirectional image mappings: (a) Monet paintings to landscape photos, (b) zebras to horses, and (c) summer to winter photos in Yosemite National Park. (Figure reproduced from [4] with permission.)

Superresolution

Superresolution allows a high-resolution image to be generated from a lower-resolution image, with the trained model inferring photo-realistic details while upsampling. The SRGAN model [36] extends earlier efforts by adding an adversarial loss component, which constrains images to reside on the manifold of natural images.

The SRGAN generator is conditioned on a low-resolution image and infers photo-realistic natural images with 4 # upscaling factors. Unlike most GAN applications, the adversarial loss is one component of a larger loss function, which also includes perceptual loss from a pre trained classifier, and a regularization loss that encourages spatially coherent images. In this context, the adversarial loss constrains the overall solution to the manifold of natural images, producing perceptually more convincing solutions.

Customizing deep-learning applications can often be hampered by the availability of relevant curated training data sets. However, SRGAN is straightforward in customizing to specific domains, as new training image pairs can easily be constructed by downsampling a corpus of high-resolution images. This is an important consideration in practice, since the inferred photo-realistic details that the GAN generates will vary depending on the domain of images used in the training set.

Discussion

Open questions

GANs have attracted considerable attention due to their ability to leverage vast amounts of unlabeled data. While much progress has been made to alleviate some of the challenges related to training and evaluating GANs, there still remain several open challenges.

Mode collapse

As articulated in the section "Training GANs," a common problem of GANs involves the generator collapsing to produce a small family of similar samples (partial collapse) and, in the worst case, producing simply a single sample (complete collapse) [26], [48]. Diversity in the generator can be increased by practical hacks to balance the distribution of samples produced by the discriminator for real and fake batches, or by employing multiple GANs to cover the different modes of the probability distribution [49]. Yet another solution to alleviate mode collapse is to alter the distance measure used to compare statistical distributions.

Arjovsky [32] proposed to compare distributions based on a Wasserstein distance rather than a KL-based divergence (DCGAN [5]) or a total-variation distance (energy-based GAN[50]). Metz et al. [51] proposed unrolling the discriminator for several steps, i.e., letting it calculate its updates on the current generator for several steps, and then using the "unrolled" discriminators to update the generator using the normal minimax objective. As normal, the discriminator only trains on its update from one step, but the generator now has access to how the discriminator

would update itself. With the usual one step generator objective, the discriminator will simply assign a low probability to the generator's previous outputs, forcing the generator to move, resulting either in convergence, or an endless cycle of mode hopping.

However, with the unrolled objective, the generator can prevent the discriminator from focusing on the previous update, and update its own generations with the foresight of how the discriminator would have responded.

Training instability—saddle points

In a GAN, the Hessian of the loss function becomes indefinite. The optimal solution, therefore, lies in finding a saddle point rather than a local minimum. In deep learning, a large number of optimizers depend only on the first derivative of the loss function; converging to a saddle point for GANs requires good initialization. By invoking the stable manifold theorem from nonlinear systems theory, Lee et al. [52] showed that, were we to select the initial points of an optimizer at random, gradient descent would not converge to a saddle with probability one (also see [25] and [53]). Additionally, Mescheder et al. [54] have argued that convergence of a GAN's objective function suffers from the presence of a zero real part of the Jacobian matrix as well as eigenvalues with large imaginary parts. This is disheartening for GAN training; yet, due to the existence of second-order optimizers, not all hope is lost. Unfortunately, Newton-type methods have compute-time complexity that scales cubically or quadratically with the dimension of the parameters. Therefore, another line of questions lies in applying and scaling second order optimizers for adversarial training.

A more fundamental problem is the existence of an equilibrium for a GAN. Using results from Bayesian nonparametrics, Arora et al. [48] connects the existence of the equilibrium to a finite mixture of neural networks—this means that, below a certain capacity, no equilibrium might exist. On a closely related note, it has also been argued that, while GAN training can appear to have converged, the trained distribution could still be far away from the target distribution. To alleviate this issue, Arora et al.[48] propose a new measure called the neural net distance.

Evaluating generative models

How can one gauge the fidelity of samples synthesized by a generative models? Should we use a likelihood estimation? Can aGAN trained using one methodology be compared to another (model comparison)? These are open-ended questions that are not only relevant for GANs but also for probabilistic models, in general. Theis [55] argued that evaluating GANs using different measures can lead conflicting conclusions about the quality of synthesized samples; the decision to select one measure over another depends on the application.

Conclusions

The explosion of interest in GANs is driven not only by their potential to learn deep, highly nonlinear mappings from a latent space into a data space and back but also by their potential to make use of the vast quantities of unlabeled image data that remain closed to deep representation learning. Within the subtleties of GAN training, there are many opportunities for developments in theory and algorithms, and with the power of deep networks, there are vast opportunities for new applications.

My understanding of the research paper and project :

1. Ganesh Rohit (1610110127)

Generative adversarial networks (GANs) are deep neural network architectures comprised of two artificial neural networks. GANs comprise of two deep neural networks, the **generator,** and the **discriminator**.

GANs' have incredible potential, because they can learn to imitate any distribution of data. The main focus for GAN (Generative Adversarial Networks) is to generate data from scratch. The data can include anything from images to music. We use GANS mostly for images.

Unlike the other conventional neural networks, Generative adversarial networks (GANs) take up a game-theoretic approach. The two neural networks are in constant battle with each other during the training process. The two neural networks are the Generator and the Discriminator. They both are trained against each other.

Generator:

It is one of the two neural networks used in GANs. It creates real looking data from given random noise inputs. The generator tries to produce data that come from some probability distribution.

Discriminator:

It is second neural networks used in GANs. The discriminator is fed by both data generated by the generator and the true data samples. The discriminator acts like a judge. It decides if the input comes from the generator or from the true training set. The Discriminator evaluates them for authenticity.

How GANs Work:

A generator is used to generate real-looking images and the discriminator's job is to identify which one is a fake. The two neural networks are always in a constant battle, the generator trying to fool the discriminator by the data it is generating and the discriminator tries not to get fooled by the generator. But a generator alone will just create random noise. Conceptually, the discriminator in GAN provides guidance to the generator on what images to create. To generate the best images you will need a very good generator and a discriminator. This is because if your generator is not good enough, it will never be able to fool the discriminator and the model will never converge. If the discriminator is bad, then images which make no sense will also be classified as real and hence your model never trains and in turn you never produce the desired output.

During the training, the generator is learning to create fake data, and the discriminator is learning to detect the fake data generated by the generator. Both of them are learning and improving. The generator is constantly learning to create better fakes, and the discriminator is constantly getting better at detecting them. The end result being that the Generator is now trained to create ultra-realistic fake data.

The Discriminator's output *D(X)* is the probability that it's input *x* is real, i.e. *P (class of input = real image).* If the input is real data then D(x) = 1, when the input is fake data generated by the generator then D(x) = 0. We want the generator to create images with *D(x) = 1*. So we can train the generator by back propagating this target value all the way back to the generator, i.e. we train the generator to create images that towards what the discriminator thinks it is real.

We train both networks in alternating steps and lock them into a fierce competition to improve them. Eventually, the discriminator identifies the tiny difference between the real and the generated, and the generator creates images that the discriminator cannot tell the difference. The GAN model eventually converges and produces natural look images.

Backpropagation:

To measure the loss or the cost function at the discriminator end, we use the cross-entropy function as in most Deep Learning: *p log (q)*. For real image, *p* (the true label for real images) equals to 1. For generated images, we reverse the label (i.e. one minus label*)*. So the objective function becomes:

$$\min_{\theta_g}\max_{\theta_d}\left[\mathbb{E}_{x\sim p_{data}}\log \underbrace{D_{\theta_d}(x)}_{\substack{\text{Discriminator output}\\ \text{for real data x}}} + \mathbb{E}_{z\sim p(z)}\log(1-\underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\substack{\text{Discriminator output for}\\ \text{generated fake data G(z)}}})\right]$$

The discriminator tries to maximize the objective function, therefore we can perform gradient ascent on the objective function. The generator tries to minimize the objective function; therefore we can perform gradient descent on the objective function.

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Discriminator side objective function

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Generator side objective function

Finally when the GAN model eventually converges and produces natural look images, the discriminator gives an output of *D(x) = 0.5.*

So this is how GANs work.

GAN Problems:

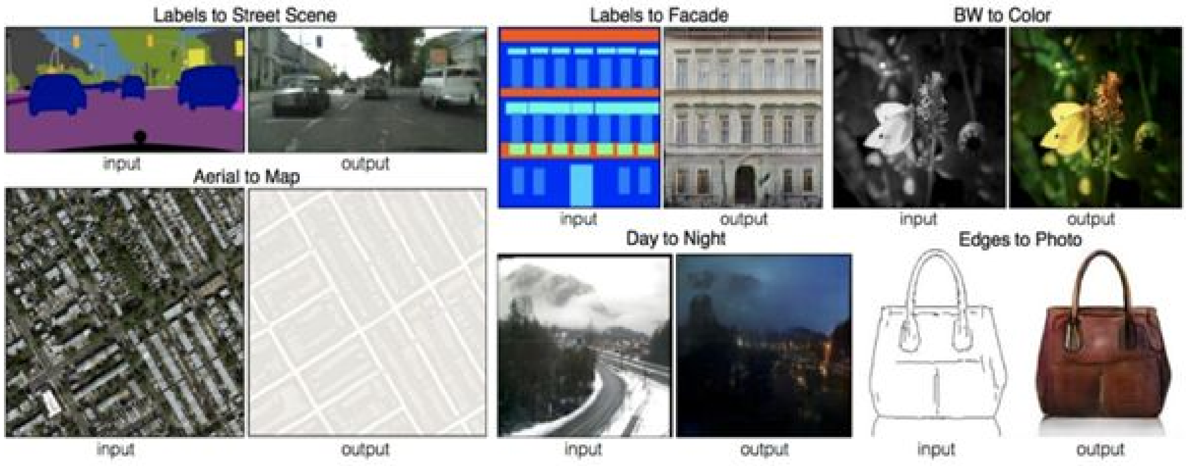
1. Non-convergence: the model parameters oscillate, destabilize and never converge.
2. Diminished gradient: the discriminator gets too successful that the generator gradient vanishes and learns nothing.

Some of the applications of GANs:

Examples

- If we give a segmented image of the road, the network is able to fill in the details with objects such as cars etc. The network is able to convert a black & white image into colour. Given an aerial map, the network is able to find the roads in the image. It is also able to fill in the details of a photo, given the edges.

- Given an image of a face, the network can construct an image which modifies different features of the face. It can widen the smile, it can add a smile, it can change all the other facial features also, and it can even represent how that person could look when they are old.



2. Aditya Dubey (1610110007)

The paper talks about how generative adversarial networks work, what problems were faced while working on, what are the application of the networks and the author tries to find an analogy of this

Technique with general signal processing techniques.

- **Introduction**

  This technique is promising because of the extraordinary amount of data in the world and much of it is easily accessible . Generative models have many short term applications but in long term they have the potential to learn the natural features of a dataset, Whether that’s categories or pixels or audio samples or something else entirely.

· **Working**

The working of the GANs can assumed that there is building and the job of the cops is to restrict the thieves to enter the building but thieves always tries to copy a normal person based on the feedback he gets about the thieves who tried entering the building and rejected due to a reason. The same way deep learning models GANs work there are Two networks which compete with each the first one is the generator and the second one

Is the discriminator , the role of generator is produce fake data and the role of discriminator

is to categories the data between fake and real. The feedback which is given by the discriminator is used to train the generator as well as the discriminator.

Here the generator takes in noise (made up of random numbers) and the noise is shaped into an image is feed forwarded in the generator network by multiplying them with the weights of the generator. The generator input is put into the discriminator and discriminator is trained to classify the fake image as fake and real image as real. The output from the discriminator is used back propagated to the generator and weights of the generator are updated. This keeps continuing until after some time the discriminator is unable to discriminate between fake and real images. The probability of image being fake and real is 0.5. The training of GANs involves both finding the parameters of a discriminator that maximize its classification accuracy and finding the parameters of a generator that maximally confuse the discriminator.

The cost of training is evaluated mathematically by

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

· Problems faced while training :

1. difficulties in getting the pair of models to converge,
2. the generative model “collapsing” to generate very similar samples for different inputs .
3. the discriminator loss converging quickly to zero, providing no reliable path for gradient updates to the generator.

· Application of this network :

1. In image generation
2. for super resolution of images.
3. Image synthesis
4. Image-to-image translation

Implementation and code:

Implementation:

- The code is implemented in python with the help of pytorch library.
- The code is divided into majorly three parts , 1st one where the discriminator network is defined which takes input as 784 numbers(flattened form of 28*28 image) and then it trains itself on fake and real labels. The errors generated while training the discriminator is back propagated to improve the weights of the discriminator . Then comes the generator which generates a 100*1 random numbers and when it passes from the networks it produces 28*28 points. The generator image produced is passed through the discriminator and based on the result the error is back propagated to improve the weights of discriminator. A batch of 100 images are used to train a complete network. In a single epoch complete handwritten mnist data is trained.

Code:

```
import matplotlib
import torchvision
from torch.utils.data import DataLoader as DataLoader
import torch.nn as nn
from PIL import Image
from torchvision.utils import save_image
import numpy
import torch
import os
# written by Aditya Dubey , written in python

epochs = 10
batch_size = 100;

DATA_PATH = '/Users/adityadubey/PycharmProjects/dsp-project'
#/Users/adityadubey/PycharmProjects/dsp-project
MODEL_STORE_PATH = '/Users/adityadubey/PycharmProjects/dsp-project/pytorch_models\\'

trans = torchvision.transforms.Compose([torchvision.transforms.ToTensor(),
torchvision.transforms.Normalize((0.1307,), (0.3081,))])
train_dataset = torchvision.datasets.MNIST(root=DATA_PATH, train=True, transform=trans,
download=True)
test_dataset = torchvision.datasets.MNIST(root=DATA_PATH, train=False, transform=trans)

train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=False)
# divides the training data into batch sizes of 100 and shuffles the data
test_loader =  DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
#

# min min V(D,G)
#data = train
# discrimnator
# image 93 almost 8


class Discrimnator(nn.Module):
   def __init__(self):
       super(Discrimnator, self).__init__()
       self.layer1 = nn.Sequential(
           nn.Linear(784,256),
           nn.LeakyReLU(0.2),
       )
       self.layer2 = nn.Sequential(
           nn.Linear(256,100),
           nn.LeakyReLU(0.2),
           nn.Dropout(0.2)
```

```
        )
        self.layer3 = nn.Sequential(
            nn.Linear(100,1),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.2)
        )
        self.layer4 = nn.Sequential(
            nn.Sigmoid()
        )

    def forward(self,x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        return x

discrimnator = Discrimnator()

class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Linear(100,256),
            nn.LeakyReLU(0.3)
        )
        self.layer2 = nn.Sequential(
            nn.Linear(256,512),
            nn.LeakyReLU(0.3)
        )
        self.layer3 = nn.Sequential(
            nn.Linear(512, 784),
            nn.LeakyReLU(0.3)
        )
        self.layer4 = nn.Sequential(
            nn.Tanh()
        )

    def forward(self,x):
        x = self.layer1(x);
        x = self.layer2(x);
        x = self.layer3(x);
        x = self.layer4(x);
        return x

generator = Generator()

# optimizers and loss
```

```
learning_rate = 0.0002;

def denorm(x):
   out = (x + 1) / 2
   return out.clamp(0, 1)

optimizer_dis = torch.optim.Adam(discrimnator.parameters(), lr=learning_rate)
optimizer_gen = torch.optim.Adam(generator.parameters(), lr=learning_rate)

loss = criterion = nn.BCELoss()

def train_gan(images,fake_labels,real_labels,batch_size):
   # discrimnator

   # train it on real images for discrimnator
   output = discrimnator.forward(images)
   d_loss_real = loss(output,real_labels)
   real_score = output

   # train it on fake images for discrimnator

   output = discrimnator.forward(images)
   d_loss_fake = loss(output,fake_labels)

   # collect the loss and backpropagate
   d_loss = d_loss_fake + d_loss_real
   d_1 = d_loss.item()
   optimizer_dis.zero_grad()
   d_loss.backward()
   optimizer_dis.step()

   # generator
   # input a noise image and produce a fake image

   global fake_image

   global img

   img = torch.rand(batch_size,100)
   global fake_images
   fake_image = generator.forward(img)
   # test it in discrimnator

   error_dis = discrimnator.forward(fake_image.reshape(batch_size, -1))
   error = loss(error_dis,real_labels)
   value = error.item()
   # backpropagate

   optimizer_gen.zero_grad()
   error.backward()
   optimizer_gen.step()

   return value,d_1

def denorm(x):
   out = (x + 1) / 2
   return out.clamp(0, 1)

# train discrimnator and generator
num_epochs = 100;

Error = []

DATA_PATH_1 = '/Users/adityadubey/PycharmProjects/dsp-project/images'


for epoch in range(num_epochs):
   for i, (images, _) in enumerate(train_loader):
       error1 = []
       error2 = []
       batch_size = 100
       images = images.reshape(batch_size, -1)
       real_labels = torch.ones(batch_size, 1)
       fake_labels = torch.zeros(batch_size, 1)

       #batch_size = 100

       error_gen = train_gan(images,real_labels,fake_labels,batch_size)

       error1.append(error_gen[0])
       error2.append(error_gen[1])

   error1 = numpy.array(error1)
   error2 = numpy.array(error2)
   error1.mean()
   error2.mean()

   print("epoch - no ---> {} generator-error --> {}  discrimnator --> {}
".format(epoch,error1,error2))

   # save the image
   # Save real images
   try :
       images = images.reshape(img.size(0), 1, 28, 28)
       save_image(denorm(images), os.path.join(DATA_PATH_1,
'real_images-{}.png'.format(epoch)))
       imag2 = fake_image.reshape(fake_image.size(0), 1, 28, 28)
       save_image(denorm(imag2), os.path.join(DATA_PATH_1,
'fake_images-{}.png'.format(epoch)))

   except Exception as e:
       print(e)


   # Save sampled images
   #fake_images = fake_image.reshape(fake_image.size(0), 1, 28, 28)
   #save_image(denorm(fake_images), os.path.join(DATA_PATH,
'fake_images-{}.png'.format(epoch + 1)))
```

Results:

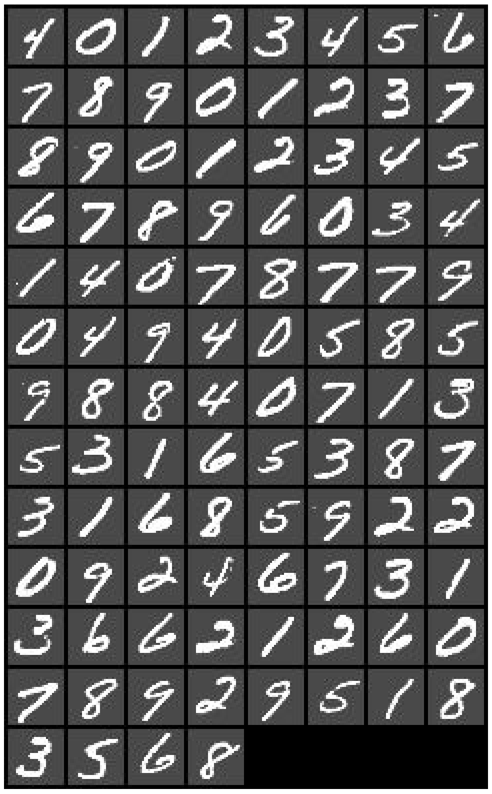
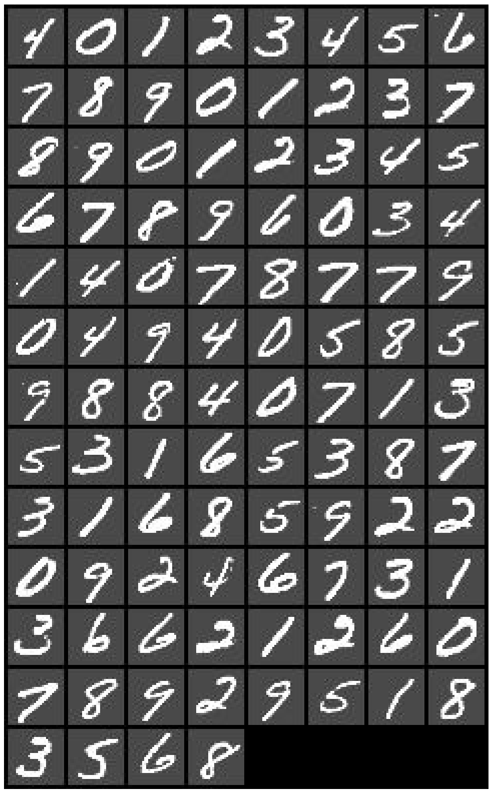
We’re going to have the Generator create new images like those found in the MNIST dataset, which is taken from the real world. The goal of the Discriminator, when shown an instance from the true MNIST dataset, is to recognize them as authentic.

Meanwhile, the Generator is creating new images that it passes to the Discriminator. It does so in the hopes that they, too, will be judged authentic, even though they are fake.

The goal of the Generator is to generate passable handwritten digits, to lie without being caught. The goal of the Discriminator is to classify images coming from the Generator as fake.

We fed random noise matrix to the generator, in hope of the generator producing the fake images which look like the images from the MNIST database.

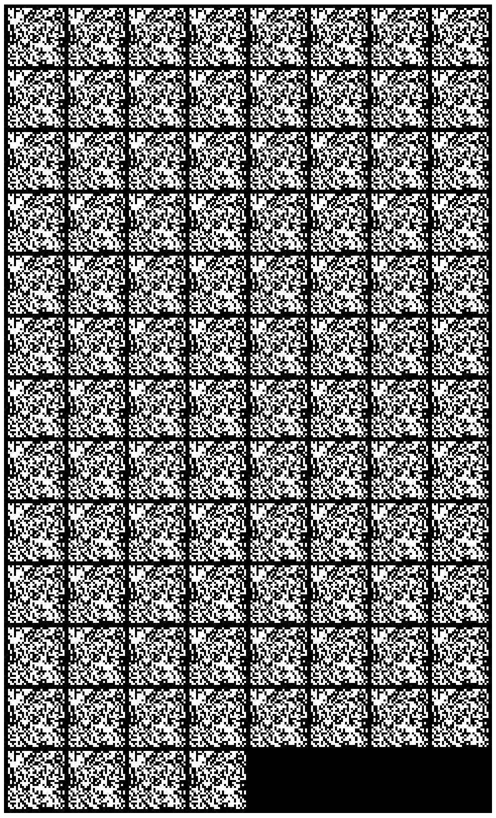
The real images from the the MNIST database used for training are:

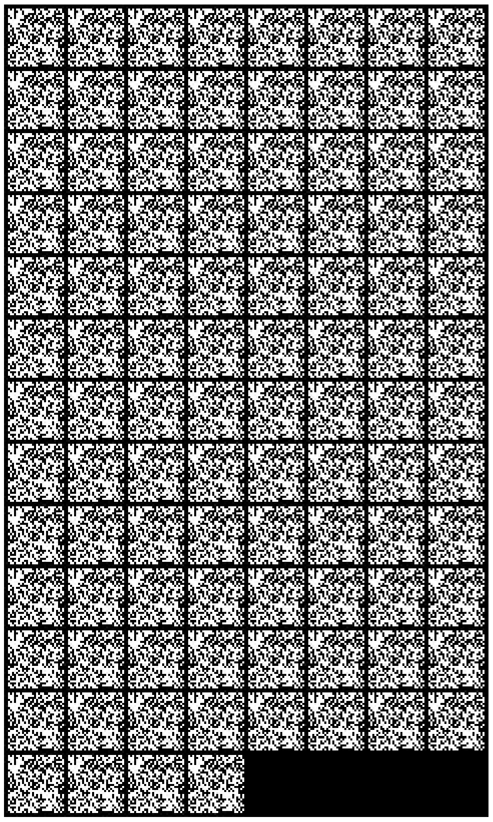



After running the code we observe:

The generator produces the following images:

Before any training: N = 0 After one round of training N = 1





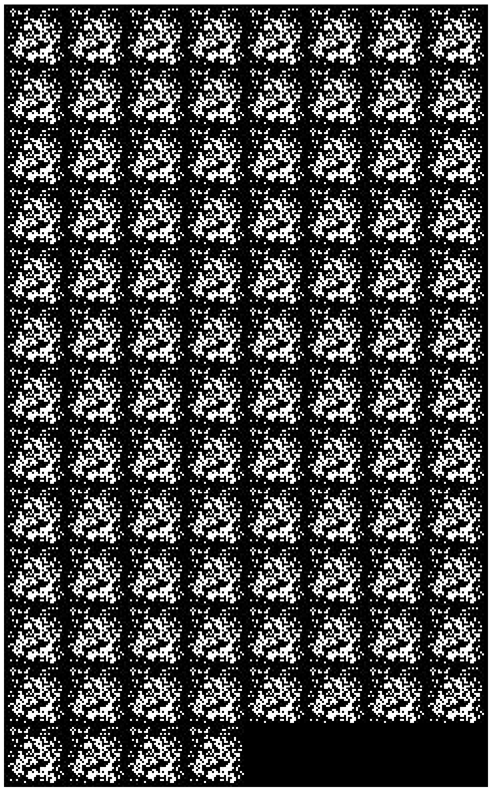
There isn’t any significant change in the images produced by the generator just after 1 round of training.

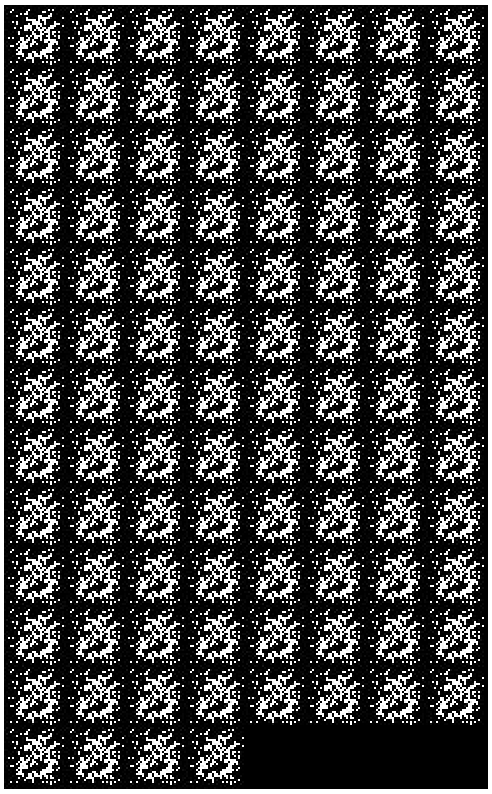
After 50 rounds of training

N = 50

After 93 rounds of training,

N = 93

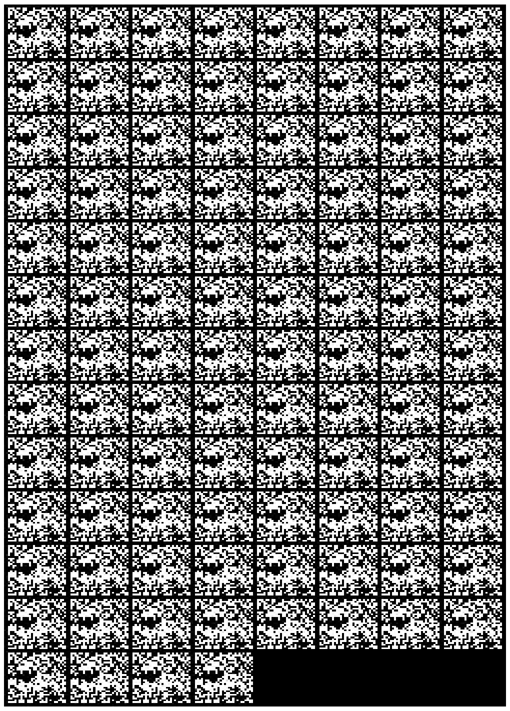


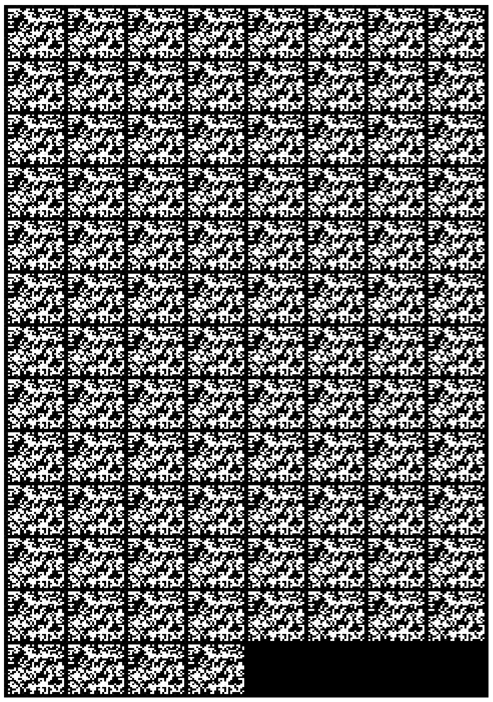


We observe some patterns after 93 rounds of training in the images produced by the generator.

After 95 rounds of training, N = 95

After 100 rounds of training, N = 100





After 95 rounds of training, there is a significant change in the image produced by the generator.

After 100 rounds of training this is the picture generated by the generator. We can clearly see the difference between this and the untrained image.

Conclusion:

Hence, after 100 rounds of training we observe that the images produced by the generator are look drastically different from the images produced by the generator before training. These images after 100 rounds of training look little similar to the real images used.

If we continue this training till 1000 rounds we will observe that the images generated by the generator look somewhat similar to the real images which we have used .

Hence we see that GANs take a long time to train. On a single GPU a GAN might take hours, on a single CPU a GAN might take days. Training the GANs is a very tough task which requires a lot of resources and time. GANs are more unstable to train because you have to train two networks from a single backpropagation. Therefore choosing the right objectives can make a big difference.

Here we tried to implement GANs on a very small scale with limited resources as training GANs on a large scale is a very complex task. While difficult to tune and therefore to use, GANs have stimulated a lot of interesting research and writing. Generative Adversarial Networks are a recent development and have shown huge promises already. It is an active area of research and new variants of GANs are coming up frequently.