Generative Adversarial Networks and Their Applications

Team 6

Altamash Rafiq, Die (Christy) Hu, Ronald Nhondova, Srishti Saha, Tien-Yu (Aaron) Liu

19 April, 2020

Abstract

In 2014, Goodfellow et al. introduced the Generative Adversarial Network (GAN), a class of algorithms that quickly revolutionized the practice and field of generative modeling. Their algorithm sets up two deep neural networks to train concurrently, one learning to generate fake data and the other learning to discriminate the fake data from the real data, until an equilibrium is reached. In the process, the generator learns to produce strong replicas and novel forms of the real data. Since 2014, this algorithm has been widely adopted and modified in many ways for applications in image generation and translation, style transfer, music/acoustics, and much more. In this tutorial paper, we present the mathematics behind the basic GAN, conditional GAN (pix2pix) and Self-Attention GANs, and provide their implementation details on Fashion-MNIST, CIFAR-10, and Facades datasets.

Introduction

Generative Adversarial Networks (GANs) are neural network-based generative models geared at producing highly realistic replicas of the data used to train them. In the notorious case of deep fakes, for example, GANs have been trained on public image and video data to produce realistic videos of public figures or fake nude images of celebrities. On the other hand, GANs have also found many positive applications eg, training on paintings from famous artists to produce style transfer algorithms that can render an image in the desired artist's style.

Presented by Goodfellow, et al. in 2014, the original algorithm pitted two multi-layer-perceptrons, the *generator* and *discriminator*, against one another in a simultaneously adversarial and collaborative relationship. Training together, the generator produces fake data and the discriminator tries to differentiate it from real data. As the iterations proceed, the discriminator differentiates between fake and real data more accurately while the generator produces fake data that is harder to distinguish than before. Eventually, an equilibrium is reached wherein learning plateaus for both networks and the generator can now be used to produce samples from a probability distribution that approximates the underlying probability distribution of the real data.

This framework has been adapted for use in not only image/video generation but also for music/acoustics, text generation and more. Here, we present an introduction to the mathematics behind GANs. To demonstrate the strengths and weaknesses of different GANs and to compare their performances, we explore three popular GAN implementations: Deep Convolutional GANs (DCGAN), Pix2Pix, and Self-Attention GANs (SAGAN) implemented on the publicly available Fashion-MNIST, CIFAR-10, and Facades datasets. We also provide a tutorial of our process here.

Background

Traditional Deep Generative models had limited impact due to the difficulty of approximating many intractable probabilistic computations that arise in strategies like maximum-likelihood estimation. These models often required the specification of an explicit sampling distribution/density (left side of Figure 1) and attempted to generate data by approximating it. GANs take a different approach: instead of an explicit density, they rely on a neural network to produce an implicit density that approximates the true density (right side of Figure 1). This approach has allowed GANs to find tremendous success.

GANs, however, require large amounts of training data. Furthermore, several issues can contribute to unstable training progress eg. mode collapsing (generator fails to produce varieties of samples), diminished gradient (discriminator overpowers generator, leading to no improvement in the generator), non-convergence (model parameters oscillate without convergence), and sensitive hyperparameter tuning. Over the past few years, a variety of techniques have been implemented to address these difficulties either through constructing new network architectures, using different objective functions, applying empirical heuristic tricks, or incorporating regularization methods.

Radford et al. (2016) introduced DCGAN by using convolutional layers in GANs. This design contributes to GAN's performance on data with spatial relationships, such as complex images and videos. Regarding using different learning objectives, Arjovsky et al. (2017) introduced Wasserstein GAN by replacing cross-entropy loss with Wasserstein loss which correlated with both the generator's convergence and generated image quality,

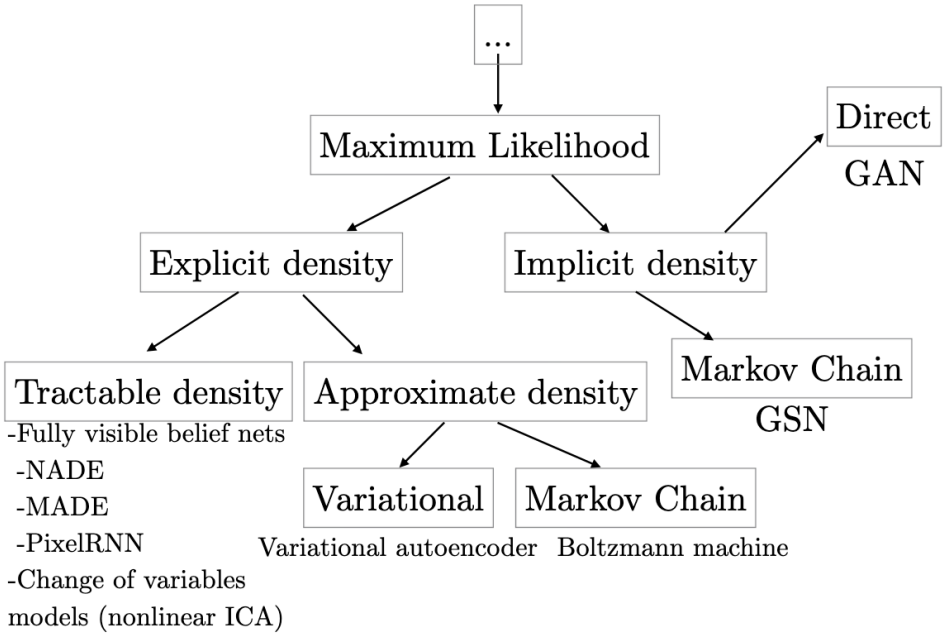

Figure 1: Deep Generative Models Taxonomy (Goodfellow, 2016)

making it possible to more closely track optimization of GANs. To further improve GAN training behavior, Zhang et al. (2019) proposed SAGAN which applied spectral normalization to both the generator and the discriminator while also reducing computational cost.

Methods

General GAN Algorithm

The training strategy of generative adversarial networks is based on Game Theory principles applied to collective learning between two players, the generator and the discriminator. The two players are each represented by a function which is differentiable both with respect to the function's inputs and parameters. Collectively, these functions form a probabilistic model, containing latent variables **z** and observed variables **x** (Goodfellow et al.,2014). The latent variables here refer to some underlying structure in the data that the generator needs to approximate. When z is sampled from some simple prior distribution, often random noise, the generator function, $G(z, \theta^{(G)})$, produces a sample of **x** drawn from its current model of the probability distribution of the data. Typically, a deep neural network is used for both $G(z, \theta^{(G)})$ and the discriminator, $D(x, \theta^{(D)})$ (Figure 2).

The cost function for each player is defined in terms of both players' parameters, but each player can only modify its own model parameters and not those of the other player. This is mathematically specified below:

- Discriminator: minimize $J^{(D)}(\theta^{(D)}, \theta^{(G)})$
- Generator: minimize $J^{(G)}(\theta^{(D)}, \theta^{(G)})$

As a result of this specification of cost functions, the players' interaction becomes a game (in the Game Theory sense) instead of an optimization problem. The objectives of each player shift from finding the local minima for their own networks to finding the local differential Nash equilibrium (Ratliff et al.,2013). A description of the general training process is included in appendix 1b. In most GAN implementations, the same cost function $J^{(D)}(\theta^{(D)}, \theta^{(G)})$, is used for the discriminator. Implementations differ primarily in terms of the cost function used for the generator. The most common discriminator cost function is as follows:

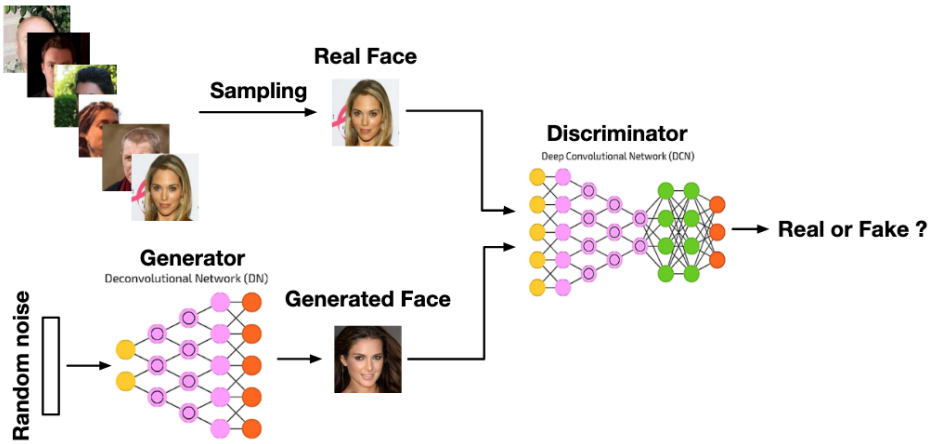

Figure 2: Architecture of a GAN.(Wang et al, 2020)

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\left[\mathop{\mathbb{E}}_{x\sim p_{data}(x)}[log\ D(x, \theta^{(D)})] + \mathop{\mathbb{E}}_{z\sim p_z(z)}[log(1\ - D(G(z, \theta^{(G)}), \theta^{(D)}))]\right] \tag{1}$$

Here, p_z is the density function for the prior and p_{data} is the density function for the source of the observed data. If both models have sufficient capacity, then the Nash equilibrium of this game corresponds to $G(z, \theta^{(G)})$ being drawn from a distribution that the discriminator interprets as identical to the true distribution as the training data. At this point, $D(x, \theta^{(D)}) = \frac{1}{2}$ (see appendix 1a for derivation) for all $\mathbf{x}$, real or fake. This feedback system from Discriminator to Generator is often missing in other generative models. This introduces another perspective on GANs - their structure is also cooperative rather than just adversarial; the discriminator estimates the ratio of densities and then shares this information with the generator, hence instructing the generator. The game is set up as what is referred to as minimax in game theory. This means that the optimal parameters for the generator are:

$$\theta^{(G)*} = \min_{\theta^{(\mathbf{G})}} \max_{\theta^{(\mathbf{D})}} \left[-J^{(D)}(\theta^{(D)}, \theta^{(G)})\right] \tag{2}$$

As the discriminator is minimizing whilst the generator is trying to maximize the same cross-entropy function, this makes the generator susceptible to vanishing gradients. As such, the most common cost function for the generator is:

$$J^{(G)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathop{\mathbb{E}}_{z\sim p_z(z)} logD(G(z, \theta^{(G)}), \theta^{(D)}) \tag{3}$$

DCGAN

DCGANs extends the work done by Goodfellow et al. (2014) by adding multiple convolution layers to the multi-layer perceptrons in both the generator and the discriminator. Figure 3 presents Radford et al's implementation of DCGAN where De-Convolution layers are used instead of convolution layers. In either case, the generator comprises a number of convolution filters that learn to mix and match features to produce data that resembles the real data.

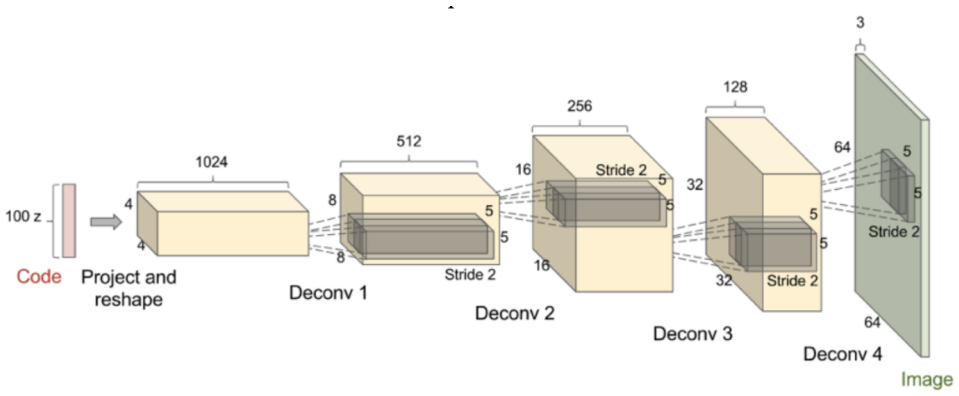

Figure 3: The generator network used by a DCGAN (Radford et al., 2015)

SAGAN

Traditional DCGANs have trouble with image classes that are more distinguishable by geometry than texture in multiclass datasets. SAGAN, proposed by Zhang et al. (2019) seeks to answer this problem by utilizing the self-attention mechanism as a supplement to convolutions and achieving a reasonable balance between computational efficiency and long-range dependencies (Zhang et al., 2019). The self-attention mechanism is developed based on the concept of *Scaled Dot-Product Attention* (Figure 4) proposed by Vaswani et al. (2017). Here the attention is defined by obtaining the weights of V (value) after performing a matrix multiplication of Q (query) and K (key). When incorporated into the GAN framework, this layer allows both the generator and discriminator to model highly detailed features of one location in relation to other distant portions of the image with a relatively small computational cost (Zhang et al., 2019).

The specific math for the process is shown below:

$$\begin{aligned} \beta_{j,i} &= \frac{\exp(s_{ij})}{\sum_{i=1}^{N} \exp(s_{ij})}, \text{ where } s_{ij} = f(x_i)^T g(x_j) \\ o_j &= v\left(\sum_{i=1}^{N} \beta_{j,i} h(x_i)\right), h(x_i) = W_h x_i, v(x_i) = W_v x_i \end{aligned} \tag{4}$$

The final output of the convolutional layer is denoted as below, where γ represents a learnable scale parameter that is initialized as 0 for the network to first explore the local neighborhood for spatial cues before learning to refine with self-attention. The self-attention layer is used in both the generator and the discriminator.

$$y_i = \gamma o_i + x_i \tag{5}$$

SAGAN uses the hinge version of the adversarial loss. The generator and discriminator are trained to minimize this loss in an alternating fashion. (Zhang et al., 2019)

$$\begin{aligned} L_D &= -\mathbb{E}_{(x,y)\sim p_{\text{data}}}[\min(0, -1 + D(x,y))] - \mathbb{E}_{z\sim p_z, y\sim p_{\text{data}}}[\min(0, -1 - D(G(z),y))] \\ L_G &= -\mathbb{E}_{z\sim p_x, y\sim p_{\text{data}}} D(G(z),y) \end{aligned} \tag{6}$$

SAGAN uses a two-timescale update rule by applying different learning rates to the discriminator and the generator and also adds spectral normalization to control the gradients. Both techniques have helped SAGAN achieve more stable training behavior while reducing its computational cost. (Heusel et al., 2017; Zhang et al., 2019)

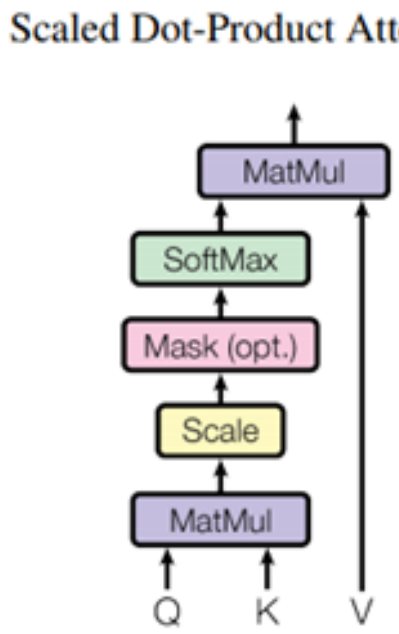

Figure 4: Source: Vaswani et al. (2017)

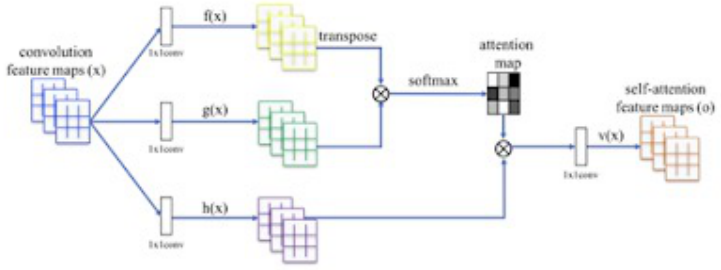

Figure 5: The self-attention layer. The generated feature maps are being passed through three 1x1 convolutions and the filters are named as f, g, and h. Self-attention mechanism: image outputs through f, g, and h are then being vectorized, multiplied (with their transpose), and softmaxed (on rows) to obtain the attention map. The final output o is generated through merging $h(x)$ with the attention map.

CGAN (pix2pix)

Conditional Generative Adversarial Networks (Mirza et al., 2014) extend the concept of traditional GANs to a conditional model by conditioning both the generator and discriminator on auxiliary information fed in as an additional input-layer to both functions(eqn 8).

$$\min_G \max_D V(D,G) \propto \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}\ (\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{7}$$

CGAN implementations like pix2pix (Isola et al.,2016) learn to map an input image to an output image while also learning a loss function to train this mapping (Figure 6). Therefore, it does not require a specified loss function. This method has several applications like synthesizing photos from label maps, reconstructing objects from edge maps, and colorizing images.

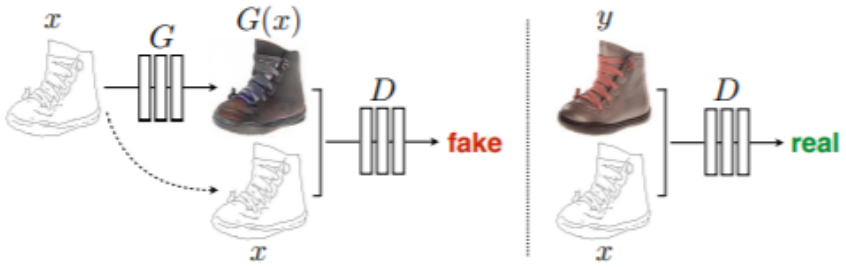

Figure 6: Training a conditional GAN to map edges to a real photo. Both the generator and discriminator access the input edge map (x) that makes it different from a conventional GAN (Isola P. et al.)

In pix2pix, the generator is a convolutional neural network that follows a modified U-Net architecture (Ronneberger et al.,2015). The U-Net model improves a conventional encoder-decoder deep learning model as it allows the bottleneck stage to be circumvented by forming links between layers of the same size in the encoder and the decoder. The discriminator is a PatchGAN (Isola et al.,2016) model which is a deep convolutional neural network designed to classify patches of an input image as real/fake, rather than the entire image. After running convolutionally over the entire image, the final output is an averaged score for the entire image. This method is useful because a fixed-size patch discriminator can be applied to arbitrarily large images to obtain scalable image translation.

The method for training the model is described in Figure 7.(source: here).

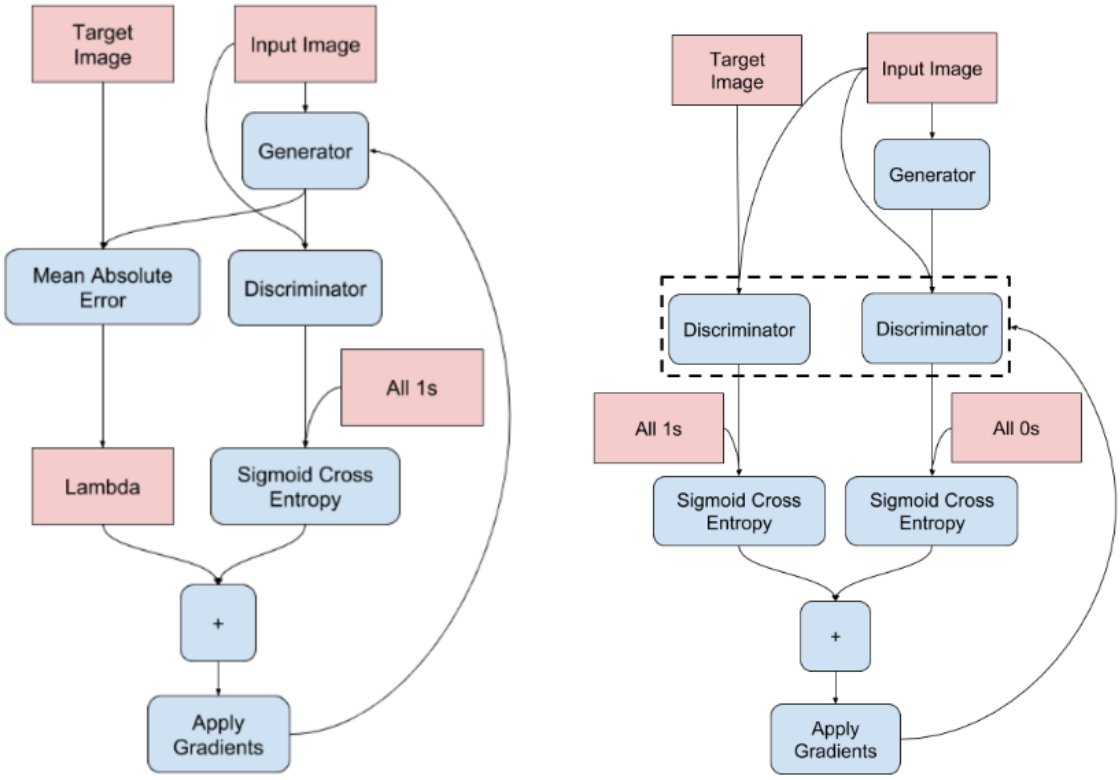

Figure 7: Training Process for generator under the pix2pix implementation (Left). The loss function for the generator learning process includes the generator GAN loss (sigmoid cross-entropy loss of the generated images and an array of ones or real images) and generator L1 loss (Mean Absolute Error between the generated image and the target image). Lambda defines the weightage assigned to the L1 loss so that it does not overpower the complete loss function. For our application, lambda=100; Training Process for discriminator (Right). The loss function for the discriminator learning process includes the sigmoid cross entropy loss of real images (sigmoid cross-entropy loss of the real images and an array of ones) and the generated images (sigmoid cross-entropy loss of the generated images and an array of zeros or generated images).

Examples of the Technique in Practice

DCGAN

We implemented a relatively shallow DCGAN on the Fashion-MNIST and CIFAR-10 datasets, training it for 10,000 epochs on each dataset. For Fashion-MNIST, we implemented the algorithm on the full data set while for CIFAR-10, we implemented the algorithm class-by-class on the following classes: horses, automobiles, ships, and birds.

On the Fashion-MNIST dataset, the DCGAN quickly started producing believable images of clothes. On the more complex and colored CIFAR-10 dataset, however, our generator did not produce strong examples despite significant training. However, if we restrict selection to the fake images that the discriminator incorrectly assigns a probability of being true as 0.99 or higher, we obtain the highly believable fake images shown below. Here, in the case of automobiles for example, not only did the generator recognize and start producing the major features of automobiles including wheels, windscreens, and bonnets, it also learnt their correct spatial reference to each other. On top of this, it also learnt how to situate the automobiles in believable positions and environments.

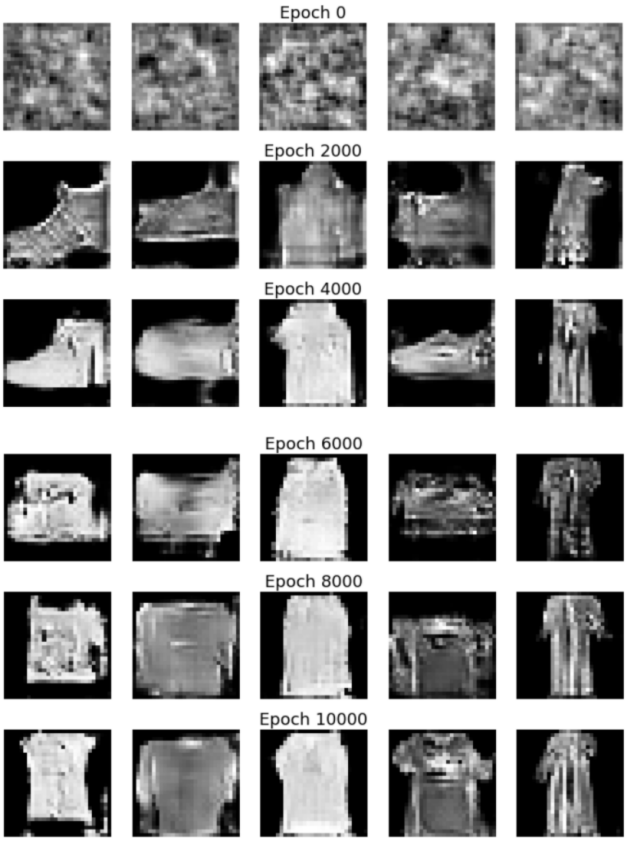

Figure 8: Fashion-MNIST output for DCGAN. Here, we show how a fixed set of noise is periodically transformed into more and more realistic images by the generator.

Figure 9: CIFAR-10 output for DCGAN

SAGAN

We applied SAGAN to the horse class in the CIFAR-10 dataset. The architecture of both the generator and discriminator closely follows this PyTorch implementation. Our generator and discriminator networks are trained by using Adam (Kingma et al., 2015) with imbalanced learning rates (generator: 0.0001, discriminator: 0.0004) and 1:1 updates (Heusel et al., 2017) for 10,000 steps. The loss is the Hinge version of the adversarial loss (Lim & Ye, 2017; Tran et al., 2017; Miyato et al., 2018), and the batch size is 64. Every 1,000 steps during the training phase, 5,000 images are generated from the fixed-length input noise (256), and further evaluated by the metric of Frechet Inception Distance (FID, Heusel et al., 2018).

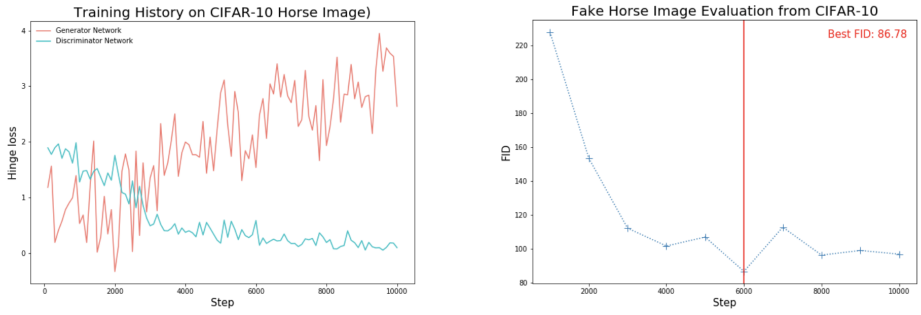

Figure 10: FID Generator Evaluation

Figure 10 displays the training history of no equilibrium established. With increasing iterations, the generator loss increases while that of the discriminator decreases, possibly indicating that the discriminator is overpowering the generator. This could be attributed to causes including the incorrect length of the latent vector, an insufficient amount of training images (5k images), and bad initialization. Despite that, the generator is able to produce images (Figure 11) of satisfactory quality (lowest FID 86.78). However, careful hyperparameter tuning appears necessary to successfully train a SAGAN.

FAKE REAL

Figure 11: Comparing Fake Images to Real Images - SAGAN

CGAN (pix2pix)

The model implementation used the framework provided in the Tensorflow colab notebook. The training was done on 400 images and the test set has 106 images. After training the model for 150 epochs, the generated are shown in Figure 12.

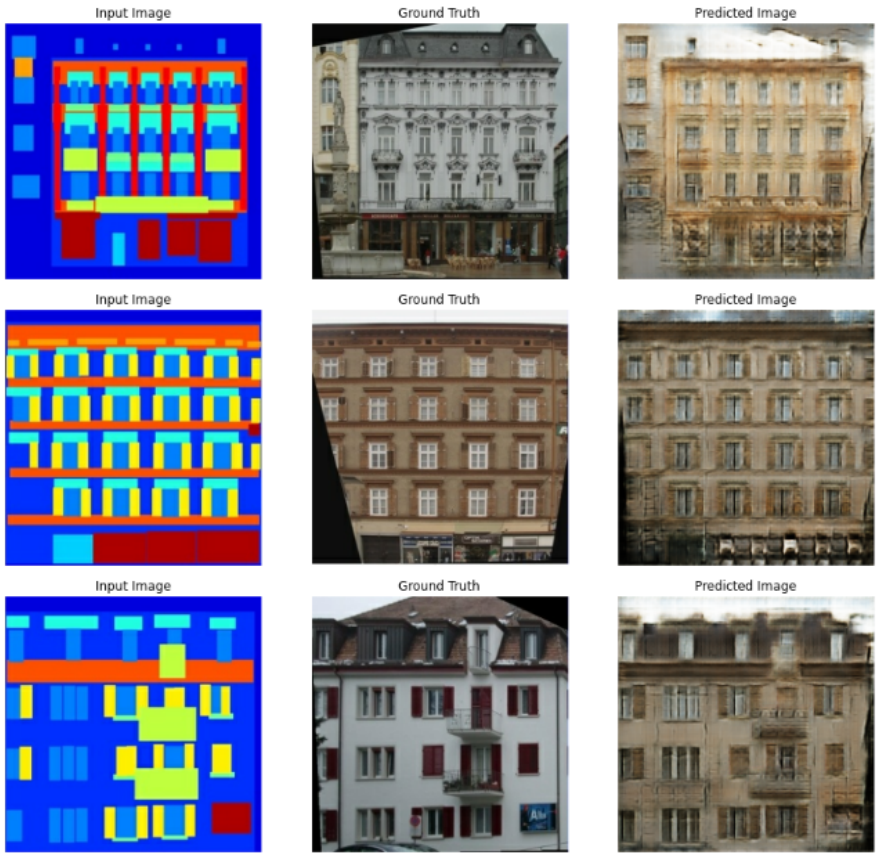

Figure 12: Input images, ground truth images, and generated images after 150 epochs (from left to right).

As neither the generator GAN loss nor the discriminator loss dropped suddenly (Figure 13), domination of one model over the other was not an issue and balanced learning occurred. To assess the average uncertainty in the discriminators predictions, we choose a threshold of $log(2) = 0.69$ due to the presence of 2 output options. A discriminator loss of under 0.69 means the discriminator is performing better than random at identifying the real images from the generated images. A generator GAN loss value below 0.69 implies that the generator is doing better than random at fooling the discriminator with fake images. As expected, the L1-loss decreases as training progresses.

Although the above model has performed in an optimal manner, the predicted image quality on the test data is not crisp. With higher epochs we expect an increase in the quality of output images.

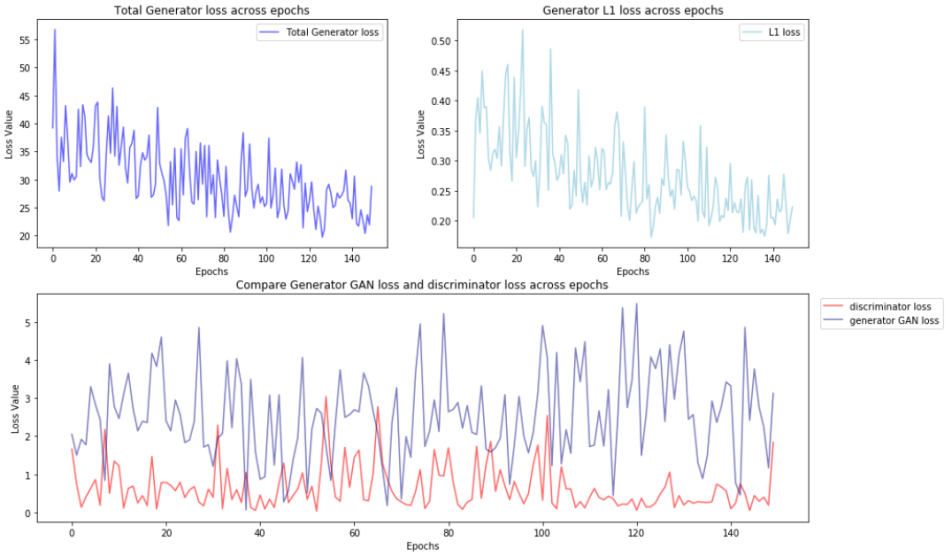

Figure 13: Training history of pix2pix for generator and discriminator.

Comparison

SAGAN produces higher quality images on the horse class in the CIFAR-10 dataset as compared to DCGAN. This might be due to more stable training of SAGAN. Traditional DCGANs are found to have difficulty in modeling certain image classes that are more discernible by geometry than texture with less structural constraints when trained on multi-class datasets such as ImagetNet (Zhang et al., 2019). In a convolution operation, although filters are great at examining spatial locality information, the receptive fields are determined by the spatial size of the kernel and therefore may not be large enough to accommodate complex structures. This problem can be alleviated by increasing the kernel size or the network depth, at the expense of computational efficiency.

Although GANs can generate a random image from the available data, they cannot capture the complex relationship between the latent space and the generated images. CGANs overcome this problem by conditioning both the generator and the discriminator models on additional information like the class label. Thus, the standalone trained generator model can be used to generate images mapped to a specific image-domain in the dataset.

Summary

To summarize, Generative Adversarial Networks function through the uniquely adversarial but collaborative relationship of the generator and discriminator neural networks, with the former trying to create fake data based on the training data while the latter discriminates this fake data from the real data. As both models learn and improve through each iteration, they eventually reach an equilibrium where the generated data approximates the probability distribution of the real data. This concept has found use in a plethora of applications such as image and video generation (eg. deep fakes), music/acoustics, text generation, and architecture.

The advantage of GANs compared to other techniques is that they do not require the specification of an explicit sampling density and can instead map an implicit probability density. Moreover, they are able to map much more complex densities than traditional methods can. However, they are still susceptible to overfitting or underfitting as well as vanishing gradients as the discriminator's confidence increases. Furthermore, they often require prohibitively large amounts of training data. With the development of newer methods, these difficulties are being overcome, opening the possibility for more powerful implementations of GANs in the future.

Roles

Altamash Rafiq - Altamash was one of the primary authors of the project proposal as well as one of the primary producers of the final project video. In this report, he wrote the Abstract, Introduction and DCGAN sections. He implemented the DCGAN algorithm on the Fashion-MNIST and CIFAR-10 datasets to produce the tutorials and analyses referenced in the DCGAN section. Finally, he was primarily responsible for editing the final report and made contributions to most of the sections in terms of either content writing, content trimming, or general editing.

Die (Christy) Hu - Christy collaborated with Aaron on the research and implementation of SAGAN on CIFAR-10, Fashion-MNIST, and pokemon images. The result on CIFAR-10 (bird) was used in the presentation video. Christy was also responsible for writing the Background, Summary and SAGAN sections in the report and worked with Aaron on video production but was not selected to represent our group in the class showcase.

Ronald Nhondova - Ronald was responsible for reading the literature relating to GANs. He wrote the method section including the mathematical foundations and derivation of proofs of the original GAN paper. He also contributed to the writing of the background and research into how other variations of GANs have stemmed from the original paper.

Srishti Saha - Srishti was primarily responsible for the research of the Conditional GANs (pix2pix) model implementation for image-to-image translation. She implemented the model on the facades dataset and the maps dataset but included only the results from the facades dataset due to the quality of outputs obtained in the available computation time. She was responsible for writing the report sections dedicated to pix2pix. She was also one of the primary producers of the final project video which was showcased in the class.

Tien-Yu (Aaron) Liu - Aaron collaborated with Christy to train SAGAN on CIFAR-10, Fashion-MNIST, and pokemon images. The result on CIFAR-10 (bird) was used in the presentation video. In addition, Aaron was responsible for writing the Background, Summary and SAGAN sections in the report. Aaron also worked with Christy on video production but was not selected to represent our group in the class showcase.

References

Cheng, J., Dong, L. & Lapata, M. (2016). Long Short-Term Memory-Networks for Machine Reading.. In J. Su, X. Carreras & K. Duh (eds.), EMNLP (p./pp. 551-561), : The Association for Computational Linguistics. ISBN: 978-1-945626-25-8.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014). Generative Adversarial Networks.

Goodfellow, I.J. (2016). NIPS 2016 Tutorial: Generative Adversarial Networks. ArXiv, abs/1701.00160.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. & Courville, A. C. (2017). Improved Training of Wasserstein GANs.. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan & R. Garnett (eds.), NIPS (p./pp. 5767-5777).

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. & Courville, A. (2017). Improved Training of Wasserstein GANs (cite arxiv:1704.00028Comment: New CIFAR-10 and LSUN image generation experiments).

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B. & Hochreiter, S. (2017). GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan & R. Garnett (eds.), NIPS (p./pp. 6626-6637).

Isola, P., Zhu, J.-Y., Zhou, T. & Efros, A. A. (2016). Image-to-Image Translation with Conditional Adversarial Networks (cite arxiv:1611.07004Comment: Website: https://phillipi.github.io/pix2pix/, CVPR 2017)

Mirza, M. & Osindero, S. (2014). Conditional Generative Adversarial Nets (cite arxiv:1411.1784)

Miyato, T., Kataoka, T., Koyama, M. & Yoshida, Y. (2018). Spectral Normalization for Generative Adversarial Networks.. ICLR, : OpenReview.net.

Odena, A., Buckman, J., Olsson, C., Brown, T. B., Olah, C., Raffel, C. & Goodfellow, I. J. (2018). Is Generator Conditioning Causally Related to GAN Performance?. In J. G. Dy & A. Krause (eds.), ICML (p./pp. 3846-3855).

Radford, A., Metz, L. & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks (cite arxiv:1511.06434Comment: Under review as a conference paper at ICLR 2016).

Ratliff, L. J., Burden, S. A., & Sastry, S. S. (2013). Characterization and computation of local Nash equilibria in continuous games. 2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton). doi: 10.1109/allerton.2013.6736623

Ronneberger, O., Fischer, P. & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W. M. Wells & A. F. Frangi (eds.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015 (p./pp. 234–241), Cham: Springer International Publishing. ISBN: 978-3-319-24574-4

Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A. & Chen, X. (2016). Improved Techniques for Training GANs. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon & R. Garnett (eds.), NIPS (p./pp. 2226-2234).

Salimans, T. & Kingma, D. P. (2016). Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon & R. Garnett (eds.), NIPS (p./pp. 901).

Wang, Z., She, Q., & Ward, T.E. (2019). Generative Adversarial Networks: A Survey and Taxonomy. ArXiv, abs/1906.01529.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017). Attention is All you Need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan & R. Garnett (eds.), NIPS(p./pp. 5998-6008).

Zhang, H., Goodfellow, I., Metaxas, D. & Odena, A.. (2019). Self-Attention Generative Adversarial Networks. Proceedings of the 36th International Conference on Machine Learning, in PMLR 97:7354-7363.

Appendix

1a - Mathematical justification of equilibrium discriminator probability assignment

At global optimality we desire

$$p_{model} = p_{data} \tag{8}$$

. For Fixed G at each epoch, the discriminator also aims to minimize its cost function:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\left[\int p_{data}(x)\ log(D(x))\ dx + \int p_z(z) log(1\ - D(G(z)))\ dz\right]$$

As such, assuming that the generator becomes capable of sampling from our best approximate of the true distribution of the data, the above equation becomes:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\left[\int p_{data}(x)\ log(D(x)) + p_{model}(x) log(1\ - D(x))\ dx\right] \tag{9}$$

This cost function is minimized over [0,1] at:

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)} = \frac{1}{2} \tag{10}$$

This ratio is implicit in the cost function for the generator so as the loss changes, the model learns to favor certain parameters and not others.

1b - Description of General GAN training process

The GAN training process consists of simultaneous stochastic gradient descent (SGD). At each step, two mini-batches are sampled: a minibatch of $\mathbf{x}$ values from the dataset and a minibatch of $\mathbf{z}$ values drawn from the model's prior over latent variables. Then two gradient steps are made simultaneously: one updating $\theta^{(D)}$ to reduce $J^{(D)}$ and the other updating $\theta^{(G)}$ to reduce $J^{(G)}$. In both cases, it is possible to use the gradient-based optimization algorithm of one's choice; Adam (Kingma and Ba, 2014) is a popular choice.