Generative Adversarial Network and Variational Autoencoder Analysis

Tarun Belani
Department of Computer Science
University of Florida
Gainesville, FL 32611
tbelani@ufl.edu

Abstract

Generative Adversarial Networks [1], or GANs, are deep generative networks in which a game is played between a generating network, which aims to generate data not present in a given dataset, and a discriminator network, whose job it is to classify whether the input data came from the training data distribution or the generator distribution. Alternatively, Variational Autoencoders [2], or VAEs, are another form of deep generative models in which an encoder outputs a parameterized distribution given an image, and a decoder samples from this distribution attempting to reconstruct the original image. In this work, we compare both of these approaches on the MNIST [3] dataset to generate images of handwritten digits, and analyze the results of these two approaches.

1 Introduction

GANs [1] are networks which contain a generator network G and an adversary, or discriminator network D which try to beat each other in a minimax game. G attempts to maximize the error of the adversary network by tricking it to believe that data produced by G comes from the training dataset. At the same time, D tries to minimize its own error by correctly distinguishing between data drawn from the training set and data produced by G. The ideal relationship which G and D should have when finished training for the image generator problem is a Nash Equilibrium scenario, where G is able to produce data indistinguishable from the training data, resulting in an increase in D's error, while D is still able to accurately identify data which comes from the training set, but is always fooled by G.

VAEs [2] consists of an encoder network E' and decoder network D'. In the traditional non-variational autoencoder, an encoder E' attempts to compress its input data into a lower dimensional space to more compactly represent the input data with little information loss. In this scenario, the desired behavior of the decoder D' is to rebuild a given latent representation back into the original input which it was encoded from. Similar to the generator G in the GAN, we can use the decoder as a way to map from a lower dimensional latent space to an output similar to the input data which the latent representation encodes.

Though this network has the strong fundamental idea of using D' as a way to reconstruct input data, it fails when we would like to sample our latent representation from, say, a Gaussian. The problem with the autoencoder is that we form a discrete latent space which we decode from, rather than a continuous space. A continuous latent space would allow D' to extend its functionality in being able to decode over latent representations sampled from $\mathcal{N}(\mu, \sigma^2 I)$ in the Gaussian case. We utilize the VAE network in this project to acheive such functionality, as an alternative to the GAN.

Preprint. Under review.

2 Method

2.1 Generative Adversarial Networks

In a GAN, the generator G learns to maps points in a latent space to its learned generated data. Given m samples $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ drawn from some distribution, the network G maximizes the loss function

$$\frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(\mathbf{z}^{(i)}\right)\right)\right) \tag{1}$$

Maximizing this loss results in the behavior of G trying to maximize the discriminator D's probability of classifying the data produced by G as being drawn from the training set distribution, or equivalently maximizing the probability of G fooling D. Conversely, the discriminator network D learns to distinguish data produced from the generator G given m training samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ by maximizing the loss

$$\frac{1}{m} \sum_{i=1}^{m} \left[\log\left(D\left(\mathbf{x}^{(i)}\right)\right) + \log\left(1 - D\left(G\left(\mathbf{z}^{(i)}\right)\right)\right)\right] \tag{2}$$

Equivalently, we maximize D's estimated probability of $\mathbf{x}^{(i)}$ being classified as real, i.e coming from the training distribution, and maximizing D's estimated probability of the generated input $G(\mathbf{z}^{(i)})$ being classified as generated data. This loss function can also be interpreted as the cross entropy between the training and generated distributions, which D wants to maximize as to better distinguish between the underlying training and generated data distributions, and which G wants to minimize to lead to both training and input data distributions being similar.

2.2 Variational Autoencoders

VAEs are able to solve the issue of encoding over a continuous latent space by training a probabilistic encoder E' which, rather than producing a direct latent representation of its input data, gives an output which parameterizes some distribution. The probabilistic decoder network D' should then sample from the distribution defined by the parameters given by the encoder, and use the sampled input to reconstruct the original data. This works since a given input now encodes a subset of the overall continuous latent space as the latent variable input is sampled rather than fixed, enabling the decoder to generalize over the latent space and produce data similar to the training data given the random noise sample.

In the VAE [2], our goal is to use Bayes methods of variational inferencing by representing the results of E' and D' as posteriors $p_\theta(\mathbf{z} \mid \mathbf{x})$ and $p_\theta(\mathbf{x} \mid \mathbf{z})$ respectively, where $\mathbf{z}$ is a noise sample and $\mathbf{x}$ is the input data. Since the true posterior distribution of the latent variable $\mathbf{z}$ is not defined, we may estimate it with learnt parameters ϕ given by E''s output, to give the estimated posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$. When we go to sample from the distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$, rather than directly sampling from the distribution defined by parameter ϕ, we instead use the reparameterization trick, which in the case of parameterizing a normal distribution says $\mathbf{z} = \mu + \sigma\epsilon$, where μ and σ are given by E''s latent parameters, and $\epsilon \sim \mathcal{N}(0, I)$, I being the identity covariance matrix. Using this sampling method, we may train D' to reconstruct the original image from the sampled latent distribution. The VAE loss function [2] we minimize across the network is

$$-D_{\mathrm{KL}}(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(z \mid \mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}))\right] \tag{3}$$

where the KL divergence measures how much the estimated posterior distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$ has deviated from the prior distribution $p_\theta(\mathbf{z}) = \mathcal{N}(0, I)$. The second term is a reconstruction error which we may simplify as

$$\log(p_\theta(\mathbf{x} \mid \mathbf{z})) = \sum_{i=1}^{N} \left[x_i \log(y_i) + (1 - x_i)\log(1 - y_i)\right] \tag{4}$$

where each x_i is a binary value fed to E' and represents the input data components, each y_i is the reconstruction components which is given by the output of D', and N is the dimension of a single input example. This simplification requires the condition that D' consists of a single hidden layer with a **tanh** activation and sigmoidal output. We can observe that this is the binary cross entropy of the reconstructed data from the input data, where minimization of this term leads to minimization of the gap between the reconstructon distribution $p_\theta(\mathbf{x} \mid \mathbf{z})$ and the true input distribution $p_\theta(\mathbf{x})$, a similar effect of minimizing KL divergence in this relationship. In this project's VAE architecture, these conditions apply since the input data, or pixel values, from MNIST are collapsed to 0 or 1, i.e the input data is binary. We may also analytically compute the latent KL divergence of the output latent distribution and its prior [2], under the condition that both $p_\theta(\mathbf{z}) = \mathcal{N}(0, I)$ and $q_\phi(\mathbf{z} \mid \mathbf{x})$ are Gaussians, as

$$-D_{\mathrm{KL}}(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z})) = \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log\left(\sigma_j^2\right) - \mu_j^2 - \sigma_j^2\right) \tag{5}$$

where J is the latent dimension, and μ_j and σ_j are the parameters of the multivariate Gaussians defined by E''s output, or $q_\phi(\mathbf{z} \mid \mathbf{x})$. These analytic equivalencies allow for more feasible computation when training the network, as the reconstruction loss becomes a function of the input data and reconstructed data, and the KL divergence term is analytically a function of the latent parameters output from E'.

2.3 Approach

To compare the performance and results of GANs and VAEs as well as differences between the various architectures, we use the MNIST handwritten digit dataset to train the networks to generate images from their estimated distribution of the dataset. In this project, both MLP and Convolutional Neural Network [3] architectures are used to demonstrate model performance and how performance differs with various architectures. Further view the specific architecture details as well as the code implementation of the methods and behaviors mentioned in 2 is available in the code repository[1].

With both the GAN and VAE approaches, MLP networks can be designed, where each layer is fully connected with the next, to be able to generate images. While this traditional approach may work, it is worth it to also build these networks in their convolutional variants, as we are training these models on an image dataset where it can help to decompose images into local feature indicators in the representational form of latent variables, and generating new images using upsampling methods from the latent features. This convolutional approach may also generalize well to different image datasets with greater feature complexity, such as the ImageNet dataset [4].

One popular variant of the GAN is the Deep Convolutional Generative Adversarial Network [5], or DCGAN. In the DCGAN, an architecture is formed so that the network has no fully connected layers, the ReLU activation is used in all layers of the generator G and discriminator D, excluding the output layer, the leaky ReLU activation [6] is used in all layers of D, and batch normalization [7] is used in both G and D. While D uses convolution layers to build feature maps and classify the input data, G uses transposed convolutions [8] as the method to upsample from the latent noise. The convolutional variant of the VAE simply involves substitution of the fully connected layers with convolutional layers in the encoder and transposed convolutions in the decoder, done similarly in the DCGAN.

Aside from the architectural differences of the various network architectures mentioned, the networks in this project are trained using the Adam optimizer [9] as both a method of performing stochastic gradient descent, or SGD, to optimize the loss functions and tuned to accelerate the training process. The dropout technique [10] is also used, more in the fully connected networks, by probabilistically choosing a subset of neurons to train with in each dropout layer, in turn helping reduce overfitting.

3 Results

3.1 GAN Architecture Variants

After training and tuning the GAN using both the fully connected and convlutional architecture, it was clear that the DCGAN outperformed the GAN in terms of results. Figure 1 shows the results from both these networks. Despite the handwritten digits by the fully connected GAN model being for the most part legible, we see lots of noise in the results, indicating a lack of stabilization in this model. Alternatively, the results produced by the DCGAN indicate a more stable training process, as

[1]Source code available at `https://github.com/tarunb12/GAN-VAE/`

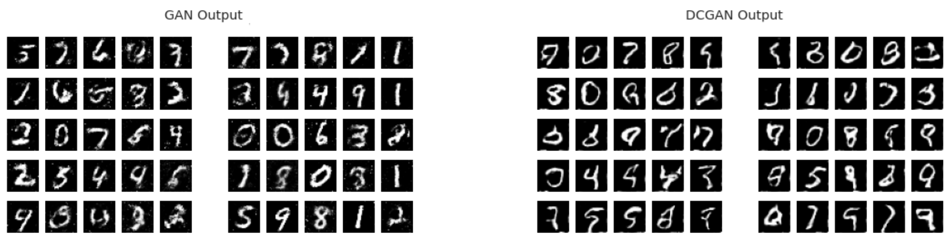

Figure 1: Generated handwritten digits from the GAN (left) and DCGAN (right).

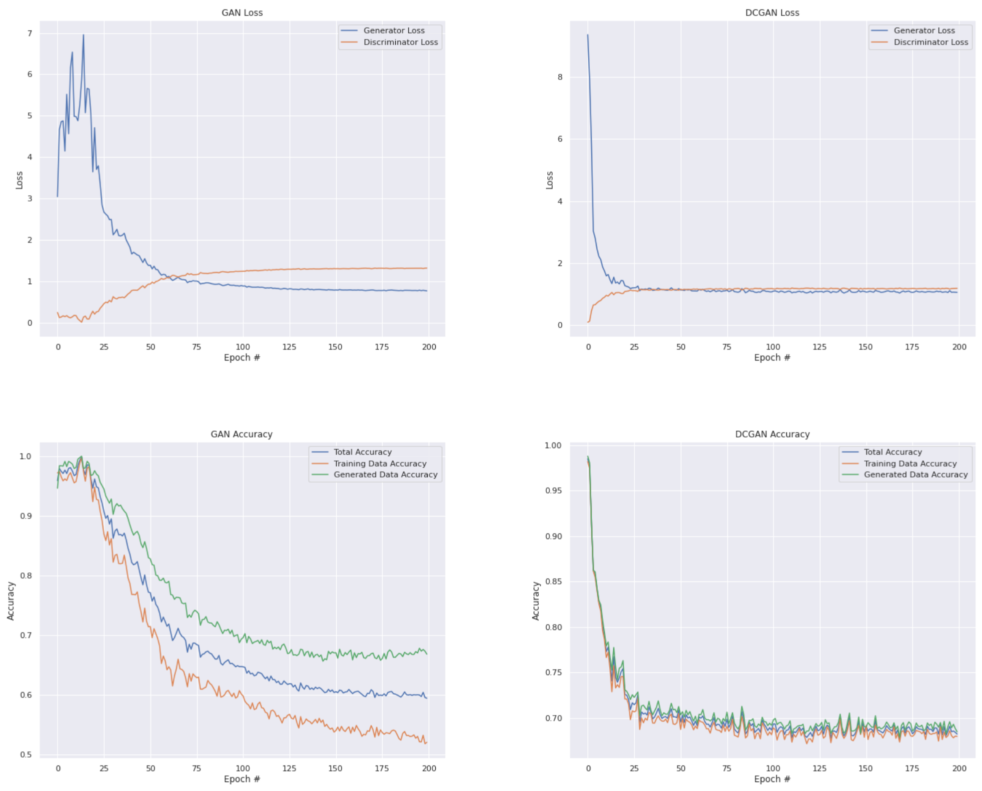

Figure 2: Loss (top) and discriminator accuracy (bottom) plots of the GAN (left) and DCGAN (right).

there is less noise present in comparison with the GAN, as seen in the sampled output in Figure 1 and metrics in Figure 2. Another reason for seeing a difference in these models is that rather than G encoding a certain combination of pixels, it represents a set of features in the image, and decodes these features through the transposed convolution operation [8].

In regards to the difference of metrics between the GAN and DCGAN, we see according to Figure 2 that more stability is present in the DCGAN from the corresponding loss plots, especially in the early stages of training. This can be partially credited to the stability that the batch normalization technique [7] brings to the network by shifting and normalizing the underlying data distribution.

3.2 VAE Architecture Variants

When looking at the performance of the fully connected VAE architecture and its convolutional variant, the CVAE, we can observe the direct mapping from a given input image to its corresponding reconstructed image. In Figure 3, we are able to directly compare ground truth images to the

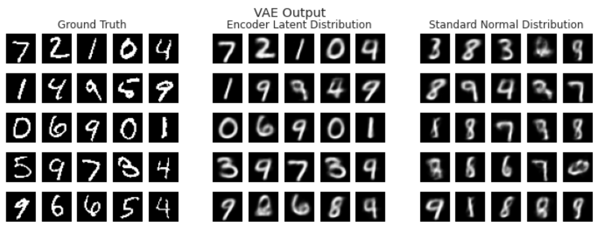

Figure 3: Generated handwritten digits from the VAE.

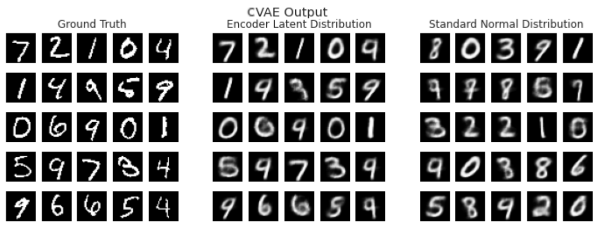

Figure 4: Generated handwritten digits from the CVAE.

reconstructed image of the VAE model. The last set of images are sampled from $\mathcal{N}(0, I)$. Similarly in Figure 4, we compare reconstructed images with their original data, and generate images from the standard normal Gaussian. In a few instances of the VAEs, we see that some of the encoder E''s resulting latent representation of the input data, in this case the ground truth data, is decoded into a different digit with similar features to the expected digit, an implication that the latent variables lie in such a region of the final latent space where encodings representing different digits have some overlap. This result outlines the continuity in the latent space, where this space has a continuous transformation of features when sampling from different points in the space.

Looking at the loss metrics in Figure 5, we do not observe much difference between the two variants besides initial network stability, and a slightly improved final loss of the CVAE. We note that the majority of the total loss comes from the reconstruction loss rather than the latent KL divergence. This leads to a greater emphasis in these networks of how well the decoder is able to reconstruct the image, rather than exactly how close the latent posterior of E' is to $\mathcal{N}(0, I)$. If such a condition were reached where the latent posterior substantially deviates from $\mathcal{N}(0, I)$, resulting in a large KL divergence, the network would then place greater importance in minimizing this difference and adjust E''s parameters accordingly.

4 Conclusions

4.1 Comparison

After analyzing both the output and metrics of both the GAN, VAE, and its convolutional variants, the fully connected GAN seems to be the lag behind the all the other networks. When compared with the DCGAN and the other VAE variants, much more noise is present in the output of the GAN shown in Figure 1, which can also be seen in the instability of the loss and accuracy metrics in Figure 2.

In the DCGAN model, we see that compared to the other networks trained on MNIST that there is less noisy output, which we notice in the form of bluriness in other sampled images, such as the VAE outputs, and sharpness in the DCGAN samples. One reason for this difference may be the technique itself lacks dependence on an explicit density estimation method. The techniques used to train the VAE networks aim to perform density estimation, shown in 2.2. As a result of the posterior, or reconstruction distibution $p_\theta(\mathbf{x} \mid \mathbf{z})$ having strong assumptions on the independence of data, the VAE's decoder may produce noisy data, i.e blurred handwritten digits.

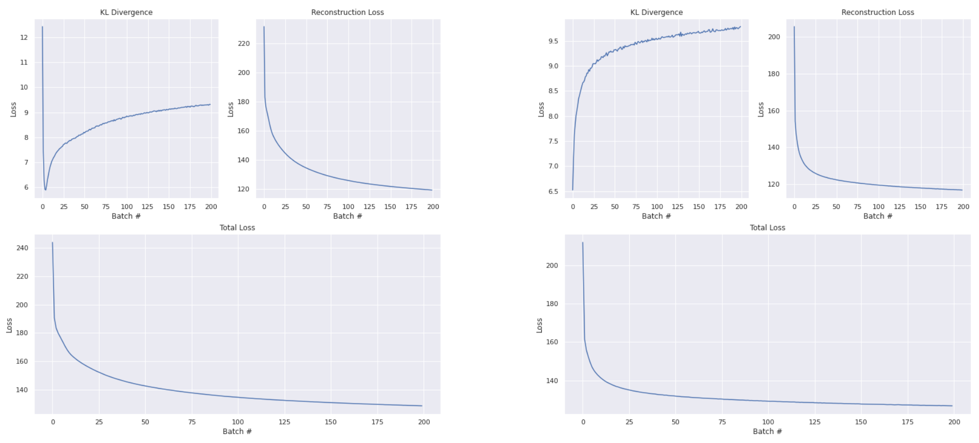

Figure 5: Loss plots of the VAE (left) and CVAE (right).

In the case of generating image data using a more complex set of features, the VAE may not generalize as well due to the noise presence. On the other hand, for relatively simpler examples such as training on MNIST, we see that as a result of optimizing on a reconstruction term in the loss function, that the generated data shown in Figures 3 and 4 show a stronger resemblance to their encoded counterparts. As a result, this produces handwritten digits with greater legibility than both the GAN and DCGAN in Figure 1. Therefore, the trained VAE models, more precisely the CVAE model, appears to perform the best in this project, while in general, the DCGAN would likely have superior performance as a generative model on a more complex image dataset, such as ImageNet [4].

4.2 Challenges

One challenge faced in this project when training the GAN was that the network faced the problem of mode collapse. This problem can result from the generator G finding a single mode which tricks the discriminator D, and in turn the discriminator eventually classifies the single mode and begins to classify correctly. This causes a cycle in which D only classifies this generated mode as being sampled from the generated distribution, in which G can simply change its behavior to a different singular mode, once again fooling D and continuing the cycle. As a result of this problem, there is a lack of variance in the data which the generator produces across a variety of noise samples. The solution to this problem was an architectural change which remedies the overfitting problem, such as adding dropout to fully connected layers.

A related problem was encountered where D was learning too quickly in the early stages of training, leaving G behind in the early stages. At this point, G's objective was to minimize equation (2) while D maximized it. Equivalently, G minimizes

$$\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(\mathbf{z}^{(i)}\right)\right)\right) \tag{6}$$

This is a problem since if D correctly classifies samples from G with high confidence, G falls into the vanishing gradient problem since G minimizes the same loss that D maximizes. This problem is fixed when we introduce a different loss function for G in which minimizing yields the same result as minimizing equation (6). Maximizing equation (1) satisfies this requirement, so that G's gradient does not vanish early on.

Lastly, an issue encountered across each model was failure of convergence using a vanilla SGD optimizer. The model seemed to progress very slowly as not much improvement was seen in both the generator output and metrics. After switching and tuning the Adam optimizer [9], an accelerated SGD method, the model had improved results over each epoch, and sped up the training process. This acceleration led to a more stable and faster convergence rate in each model.

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[2] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[3] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, page 319, 1999.

[4] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[5] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[6] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, page 807–814, 2010.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[8] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[10] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.