GAN vs VAE: A Comparative Study

Shashank Bantakal
University of Florida
Gainesville, FL 32611
sbantakal@ufl.edu

Abstract

In the field of unsupervised learning, generative models are enormously popular. They are able to learn the understand the data distribution of a training data and produce novel data with certain distinctions. This paper narrates a comparative analysis of the performance of Generative Adversarial Networks (GAN) & Variational Autoencoder (VAE). The analysis is based on training both the artificial neural network using MNIST handwriting datasets obtained through various sources over the internal and comparing the results. Finally, some of the recent improvement made on these models.

1 Project Overview

The source code affiliated with this paper has been linked below.

`https://github.com/sbantakal/Machine-Learning-Course-Work/`

1.1 Programming Language & Framework

Python has been used as the programming language of choice for this project, primarily due to ease of implementation and understanding and the availability of deep learning frameworks that it offers. Python's Pytorch framework has been used due to its flexibility and support of dynamic computational graphs that allows us to change how a network behaves on the fly.

1.2 The MNIST Database

The acronym stands for Modified National Institute of Standards and Technology[5]. It is a database containing handwritten digits (0 though 9).

MNIST dataset is the subset of the larger dataset available at National Institute of Standard and Technology. It is divided into two datasets: the training dataset has 60,000 samples of hand-written numerals, and the test set has 10,000. All the images are of the same dimensional, with the digits being centered and their size normalized.

This dataset has been used in this project implementation.

2 Generative Adversarial Network (GAN): An Overview

A generative adversarial network [1] is a machine learning framework in which a generative model G is pitted against an adversary, a discriminative model D. The two competing neural networks indulge in a contest to with each other in a minimax two-player game, each one trying to outdo the other.

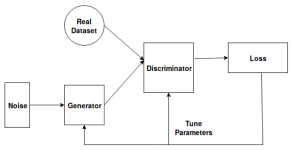

Figure 1: Architecture of Generative Adversarial Network.

The generative model G tries to the captures the data distribution, whilst the discriminative model D estimates the chances of the sample input to it is from the training dataset rather than what is spewed out by the generative model G. The challenge for G is to make its output as identical as possible to the training data in order to maximize the probability of D making a mistake. The generative model is analogous to an art forger trying to come up with a forged work that is as identical as possible to the original painting, while the discriminative model is analogous to a detective trying to the counterfeit art.

Hitherto, the most noteworthy accomplishments in the field of deep learning revolved around discriminative models, primarily those that could map a high-dimensional, plush sensory input to a class label. These successes were mostly based on backpropagation and dropout algorithms, which using piecewise linear units having a largely well-behaved gradient. Generative models on the other hand, have had minimal impact due to the intricacy of approximating many unyielding probabilistic computations that result from maximum likelihood estimation, and other related computational strategies and further due to the difficulty in using the advantages of piecewise linear units in a generative environment. However, the adversarial framework sidesteps these complications.

As mentioned before, the generative neural network generator network learns to map from a latent space to a data distribution of interest, while the discriminator network distinguishes outputs produced by the generator from the true data distribution. The training objective of the generator network is to increase the discriminator networks overall error rate by producing output features that the discriminator thinks are not synthesized but part of the true data distribution.

A recognized dataset (e.g., MNIST dataset as used in this project implementation) acts as the training data for the discriminator. Samples from the training data set are fed into the discriminator until it reaches an acceptable accuracy. The generator is trained subject to whether it is able to trick the discriminator. It is seeded with a random input sampled from a predefined latent space. Subsequently, the outputs produced by the generator are valued by the discriminator. Independent back-propagation techniques are applied to both the networks to update their network parameters to improve the quality of outputs produced by the generator, and to advance the discriminators ability to flag counterfeits produced by the latter.

Therefore, the core idea of GAN is an indirect form of training through the discriminator, i.e. the generator is being trained to minimize distance to a specific solution, enabling the model to learn in an unsupervised manner.

3 Variational AutoEncoder (VAE): An Overview

In neural network lingo, a Variational Auto-Encoder [3] consists of three primary components: an encoder, a decoder and a loss function.

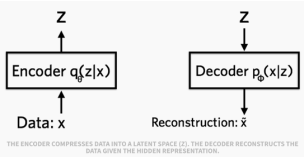

Figure 2: Architecture of Variational AutoEncoder.

The encoder is a neural network, having weights and biases θ. Given an input x the encoder outputs a hidden representation z. In case the input x is a $28 * 28$ pixel MNIST handwriting data, the encoder translates the 784 dimensional data into hidden latent representation space z. The hidden representation space has a smaller dimension compared to the input; this is stated to as the bottleneck as the encoder must evolve to efficiently compress the data into a lower dimensional space. We denote the encoder as $q_\theta(z|x)$. The lower dimensional space is stochastic; and the encoder outputs the parameters to $q_\theta(z|x)$, which is a Gaussian probability density function.

The decoder is also a neural network, having weights and biases ϕ. The input to the decoder is z, while it results the factors to probability distribution of the data. We denote the decoder as $p_\phi(x|z)$. Considering the same handwriting digit example, if the images are black and white representing each pixel as 0 or 1. The probability distribution of each pixel could be denoted using Bernoulli distribution. The decoder is input the latent space depiction of the z and outputs 784 Bernoulli parameters, one each for 784 pixels of the image. The decoder now decodes these parameters of z into 784 real-valued numbers between 0 and 1. Some amount of information is lost between the encoder input and the decoder output, as the decoder the decoder only has access to a gist of the information z. The information loss is measured using log-likelihood $\log p_\phi(x|z)$. which is a measure of how efficiently the decoder has learned to represent the original image x given its latent space z.

Finally, the loss of the variational encoder is the negative log-likelihood with a regularizer. As there is a overall representation which is common to all the datapoints, the loss function is decomposed into terms that depend only on a single datapoint l_i. The total loss is then computed as the summation of losses for each of N datapoints. The loss function l_i. for a single point x_i is given as,

$$l_i = -E_{z,q_\theta(z|x_i)}\left(\log p_\phi(x|z)\right) + KL\left(q_\theta(z|x_i)\right)$$

The first term represents the loss occurred during reconstruction, or the expected negative log likelihood of data point indexed by i. The expectation is taken with respect to the encoder's distribution over the representations. It encourages the decoder to learn to reconstruct the date. If the decoder is not able to reconstruct the data, statistically, it means that it is not able to parameterize a likelihood distribution that does not place much probability mass on true data. For e.g., if the goal is to model black and white images and our model places high probability on there being black spots where there are actually white spots, this will yield the worst possible reconstruction. Poor reconstruction will incur a large cost in this loss function. The second term is a regularizer, this is the Kullback-Leibler divergence between the encoder's distribution $q_\theta(z|x)$ and $p(z)$. It measures how much information is lost when using q to represent p.

The variational autoencoder is trained using gradient descent to optimize the loss with respect to the parameters to the encoder θ and decoder ϕ. For a stochastic gradient descent with step size ρ the encoder and decoder parameter are updated using $\theta \leftarrow \phi \leftarrow \frac{\partial l}{\partial \theta}$.

4 Implementation

4.1 Generative Adversarial Network

The composition of a generative adversarial network consists of two oppositional neural networks, a generator G and a discriminator D.

4.2 Discriminator Network

The discriminator neural network is a standard linear data classifier. At least on hidden layer is needed to make it a universal function approximator. All the hidden layers in the network have one key primary attribute, i.e., they all have a Leaky ReLu activation function applied to their outputs.

4.3 Leaky ReLu

Each neuron in the hidden layers compute a dot product and add a bias value before outputting the value to the neuron in the next layer, this mathematical computation is linear in nature. These linear outputs ensure that the entire neural network to behave linearly. However, this is problematic when the dataset such as MNIST dataset on which the network is being trained on is nonlinear.

Activation functions such as ReLu and Leaky ReLu are used to add non-linearity to network. They are placed directly after the hidden layers, since these mathematical functions are nonlinear in nature, they overall output is nonlinear. One of the most prominent activation functions used in today's neural network is the ReLu activation function, which basically calculates max between 0 and the neuron output. But, firstly since the ReLu function is not continuously differentiable (x=0 being the breaking point) and more significantly, since ReLu sets all negative values to 0, neuron that arrive at large negative values get stuck at 0 (as gradient at 0 is 0), leading to the dying ReLu problem.

In order to sidestep this conundrum, Leaky ReLu has been used in this implementation, which is quite similar to ReLu, however as it permits a small non-zero outputs to pass through it, thus, allowing unimpeded flows of the gradients through the layers in backward direction.The leakage factor was set to 0.2 in the implementation.

4.4 Sigmoid Activation

The To realize a more mathematically stable loss function on the results, it necessitates the discriminator output to be either 0 or 1, representative whether the image is legit or a counterfeit, we make use of BCEWithLogitsLoss function, which chains a sigmoid activation function and binary cross entropy loss into a single function. Therefore, our ultimate output layer will not have any activation function applied to it.

4.5 Generator Network

The generator network is almost exactly similar to the discriminator network, except that tanh function is used for the activation of the hidden layers.

4.6 Tanh

The tanh activation function scales the output between -1 and 1 instead of 0 and 1. The generator network was found to perform at it its best with tanh activation function in the output layer.

The output values should be similar to input pixel values, which are detected as normalized values that are between 0 and 1. Therefore, we need to measure our real input images to have pixel values between -1 and 1 during the training period of the discriminator.

4.7 Calculating the Discriminator and Generator Losses

4.8 Discriminator Loss

The discriminator loss is computed as the total sum of the loss for real and counterfeit images, i.e., $d_{loss} = d_{realloss} + d_{fakeloss}$, since we need require the discriminator to result in 1 for real images and 0 for counterfeits, the computed loss should reflect the same.

The losses will be binary cross entropy loss with logits, attained using BCEWithLogitsLoss. This chains a sigmoid activation function and binary cross entropy loss into a single function. For the real images, we want $D(realimages) = 1$. We need discriminator to categorize real images with a $label = 1$.

In order to assist the discriminator generalize, labels are lowered a bit to 0.9 from 1.0. To accomplish this, the parameter smooth has been used; if True, then smoothen the labels. In PyTorch, this looks like $labels = torch.ones(size) * 0.9$. The discriminator loss for the fake data is comparable, as we require $D(fakeimages) = 0$, where the counterfeit images are the generator results , i.e., $fakeimages = G(z)$.

4.9 Generator Loss

The generator loss looks fairly alike only with reversed labels. As the generator's goal is to get $D(fakeimages) = 1$. Here labels are reversed to representing the generator is trying to trick the discriminator into thinking that its replicas are legit.

4.10 Training

The generator and discriminator variables are required to update separately, two separate Adam optimizers are initialized with a learning rate of 0.002.

The training will involve interchanging between training the discriminator and the generator. The functions for real loss and fake loss are used to help establish the discriminator losses.

The discriminator training comprises of the following steps: -

1. Discriminator loss is calculated for real training images.
2. Produce counterfeit images.
3. Discriminator loss is calculated for fake generated images.
4. The two losses are added together
5. Backpropagation & optimization steps are performed to revise the discriminator's weights.

The generator training comprises of the following steps: -

1. Counterfeit images are generated
2. Calculate the discriminator loss on counterfeit images, using reversed labels.
3. Backpropagation & optimization steps are performed to revise the generator's weights.

4.11 Variational Autoencoder

The variational autoencoder implemented uses a similar setup found in most VAE papers; a multivariate normal distribution for the conditional distribution of the latent vectors given and the input image $q_\theta(z|x_i)$ and a multivariate Bernoulli distribution for the conditional distribution of images given the latent vector $p_{phi}(x|z)$. Using a Bernoulli distribution, the negative log likelihood of a data points the distribution i.e., the reconstruction loss reduces to the pixel wise binary cross entropy.

4.12 VAE Network Implementation

As discussed earlier, a variational auto encoder network consists of two different neural networks namely an encoder neural network and a decoder neural network.

The MNIST images consists of digits 0-9 in a 28*28 grayscale images. We do not center them as we will be using a binary cross entropy loss that treats pixel values as probabilities in [0,1] while creating the training and test data sets.

Both the encoder and decoder neural networks consist of two hidden convolutional layers. Convolution layers have been used as they general give better performance than fully connected versions that have the same number of parameters. Each of the convolutional layers in the encoder and the first convolutional layer in the decoder are using ReLu activations, while the final convolutional layer of the decoder has a sigmoid activation since binary cross entropy for the computation of the reconstruction loss.

During the reconstruction of the image from the latent space, the network general builds them up from low resolution, high-level descriptions, i.e., the network describe a rough image and then fills in the details. This process is generally occurring during the deconvolution process in the decoder. However, the deconvolution process can lead to uneven overlap when the kernel size (output window size) is not visible to the stride of deconvolution filter (spacing between points on the top), causing checkerboard effect in the decoded image. In order to diminish this effect, the kernel size has been set to 4.

4.13 Training

The training process of the VAE network involves the following steps: -

1. An input image is passed through the encoder.
2. The encoder outputs the parameters of the distribution.
3. A latent space z is sampled from the output of the encoder output.
4. The decoder then decodes the latent space to reconstruct the image.
5. Reconstruction error is computed using over various datapoints using binary cross entropy loss and the KL divergence functions.
6. Finally, perform backpropagation and an optimization phase to revise the network parameters of both the decoder and encoder.

Table 1: VAE vs GAN Model Comparison

Criteria	VAE	GAN
Learning type	Semisupervised & unsupervised	Unsupervised
Architecture	Convolutional	Non Linear
Gradient Update	SGD with update to reconstruction and KL loss	SGD update to both Generator and Discriminator
Optimizer	Adam	Adam
Performance Metrics	Log-likelihood and error rate	Accuracy and error rate

5 Comparative Analysis

As seen from table 1, GAN primarily supports only unsupervised learning. On the other hand, VAE is capable of both semi-supervised and unsupervised learning. In both the networks, stochastic gradient descent-based optimization is used for training through Adam optimizer. To calculate performance, in VAE log-likelihood is measured using KL divergence.

Figure 3: Training output images produced by GAN after every 10 epochs

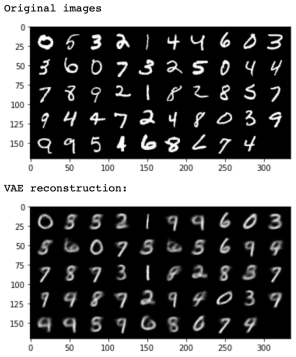

Figure 4: VAE reconstructed images

1. The primary comparison criteria of the two models is through visual inspection of their created samples. As seen above, Variational autoencoder mind to produce blurred images as matched to the GAN. The cause being decoder outputs an average value of all generated images or mean value of distribution. The image blurriness can be mitigated by using L1 loss.

2. GAN outputs nonsensical images during the initial epochs as shown figure 3, as its loss value is high due to sampling from random noise. The loss evaluates how well participant is doing against the competitor. The generators loss increase as we move through epochs, but still image quality of generated images increases.
3. GAN produces great quality images matched to the VAE but limited assortments of samples. This is due to equilibrium not been reached, non-convergence is a difficult problem in GAN.
4. In GAN, both the generator and the discriminator must learn at a similar pace. If the generator becomes too successful paralleled to discriminator for minimizing cost function, than Discriminator cannot learn due to low gradient update, this issue is termed as mode collapsing.

6 Conclusion

A comparative analysis of the two prevalent generative models based on their architecture, objective and performance was discussed in this paper. As seen both models have their pros and cons.

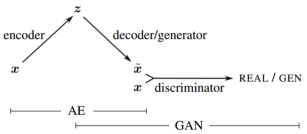

Figure 5: Struncture of VAE-GAN

A recently proposed combination of these network is VAE-GAN [4] (shown in figure 5), in which the decoder is swapped with Generator of GAN and loss function is computed using discriminator. This setup has resulted in improved quality of images compared to VAE, while producing more diverse images than GAN.

References

[1] Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua (2014). Generative Adversarial Networks (PDF). Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 2672–2680.

[2] Ian Goodfellow, M. Mirza, B. Xu, Y. Benjio. Generative Adversarial Network. Department of Computer Science and Research Operationl, University of Montreal(2014).

[3] Diederik P. Kingma, Max Welling. Auto-Encoding Variational Bayes, Machine Learning Group, Universiteit van Amsterdam(2014).

[4] Hugo L.,Ole W.,Anders L.,Soren S. Autoencoding beyond pixels using a learned similarity metric, arXiv:1512.09300v2 [cs.LG](2016).

[5] https://wiki.pathmind.com/mnist