Comparison of GAN and VAE for Fake Image Generation

Syed M. Ali

UFID: 3816-1305

ali.s@ufl.edu

ABSTRACT

In unsupervised learning, generative models are really good. They can learn the distribution of data and then make new data with a bit of variation. In this paper, we discuss how they are different from the regular discriminative models. Here we focus mainly on the Generative Adversarial Network (GAN) and Variational Autoencoder (VAE), how they work along with their architecture. Finally, we will generate images using both based on the MNIST dataset and based on that we will see the pros and cons of both.

Introduction

In machine learning, the two main types of problems are supervised learning and unsupervised learning. This leads to two main types of models that solve these problems. Discriminative models are mainly catered to solve the supervised learning problem because there is some relation between some variables x and y, so the model tries to guess the value of the y if it is given x. This guess work is based on learning the relation between the variables and that is learned by the training data by seeing existing pairs and the guess improves as the number of training data increases.

This in mathematical form can be expressed as:

$$f(x) = \text{argmax}P(y/x)$$

In this form we can see that we are just picking the y that has the most chance of being the class given x. Thus, the model is learning to find the decision boundary between the classes. Some examples of discriminative models are SVM, Decision Trees, Naïve Bayes, and Artificial Neural Network. All these models do not care how the data was generated. This is in sharp contrast to the generative models which do care about this and find how the data is distributed. It does this using the joint probability by using the mathematical form:

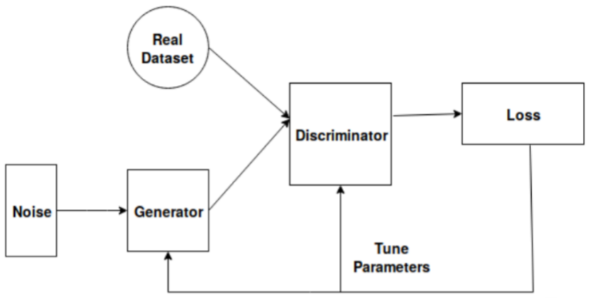
$$f(x) = \text{argmax}P(y/x) * P(y)$$

In this form, we want a score for x and y together and find y for a totally new x based on that. This way the joint probability will be the highest. The algorithms we are comparing, Generative Adversarial Network and Variational Autoencoder are of this type.

The main task of generative models is to first see some training data and try to get a good estimation of the distribution of data. Once it does this, then it goes and generates new samples with a bit of variation. Finally, the maximum likelihood estimation can be used to do the classification. Among the generative models, there are two main types. There are implicit models that learn the distribution without a density function. These types of generative models only generate the data but cannot evaluate the likelihood. The other type is the explicit models which assume a distribution and find the density using some optimization algorithm.

Generative Adversarial Network

The generative adversarial network relies on two competing networks, hence the "Adversarial" in the name. These two networks are: a generator and a discriminator. The generator tries to make new data that is as close to possible to the training data. Then the discriminator receives these samples and tries to find if they are real or not. Overtime, the generator will get better and better at making realistic samples and the discriminator will get better and better at detecting real and fake samples thus training the generator to produce very realistic samples.

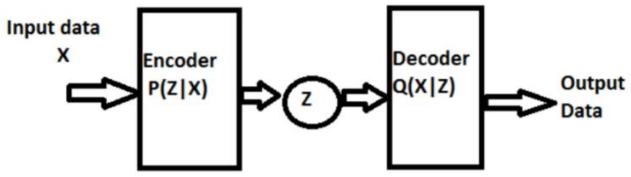


Overview of Generative Adversarial Network

The generator is basically a neural network and it takes some randomly generated data which can be called as noise. It than uses this noise to generate the samples. The discriminator is also a neural network, but it takes samples and determines if the samples were fake or real.

Variational Autoencoder

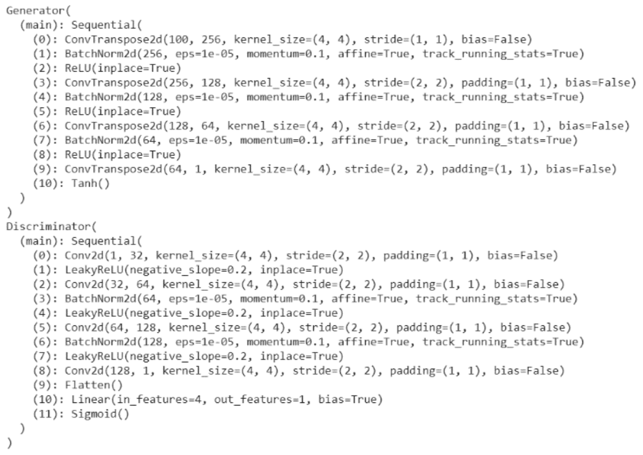
Just like the GAN, there are two networks but here the first network is an encoder and the second is the decoder. The encoder takes the training data as the input and tries to learn the representation of it so that it can convert it into an encoding vector. The vector has each of its dimension representing some feature of the data. The decoder network will take the encoding vector as the input and try to generate back the original data.



Overview of Variational Autoencoder

Experiment Details

For GAN, the model of the generator and discriminator used is show below:

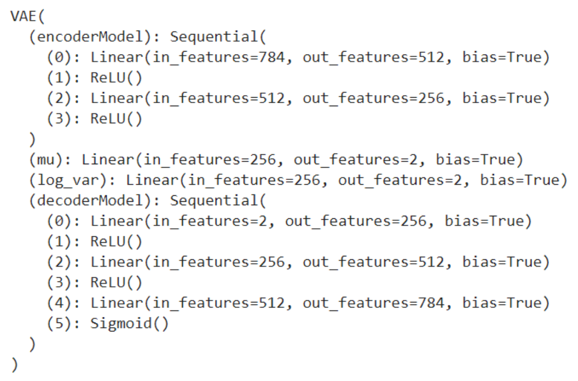


```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 256, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): Tanh()
  )
)
Discriminator(
  (main): Sequential(
    (0): Conv2d(1, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(128, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): Flatten()
    (10): Linear(in_features=4, out_features=1, bias=True)
    (11): Sigmoid()
  )
)
```

The generator starts with its input as noise vectors coming in and each layer reshapes it until it becomes an image vector in the end. Each layer passes it output to a Batch norm and ReLU before feeding to the next layer. The only exception being the output layer which uses Tanh.

At the discriminator, this image is then reshaped again at each layer until it can be decided if this is real or not, this time using leaky ReLU for activation and output layer using sigmoid.

For VAE, the model of the encoder and decoder is shown below:

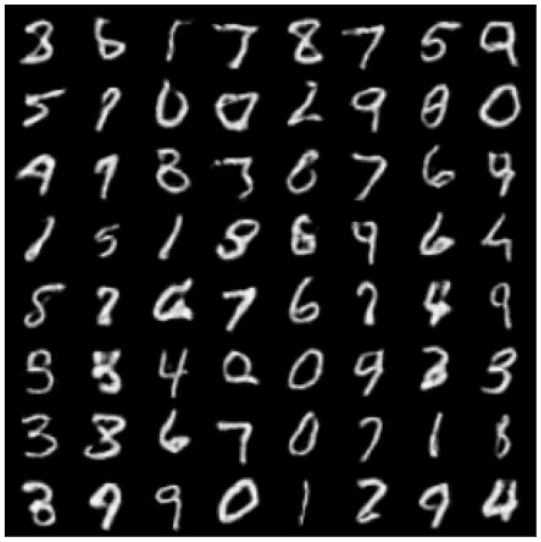

```
VAE(
  (encoderModel): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): ReLU()
  )
  (mu): Linear(in_features=256, out_features=2, bias=True)
  (log_var): Linear(in_features=256, out_features=2, bias=True)
  (decoderModel): Sequential(
    (0): Linear(in_features=2, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=784, bias=True)
    (5): Sigmoid()
  )
)
```

The encoder starts with an image and reduces the dimensions at each layer until a small vector is left. The decoder then reverses this process and makes the image back. ReLU is used for activation in both, except for the output of decoder which uses sigmoid.

The encoder has 2 separate output layers one for the mean and the other for the variance.

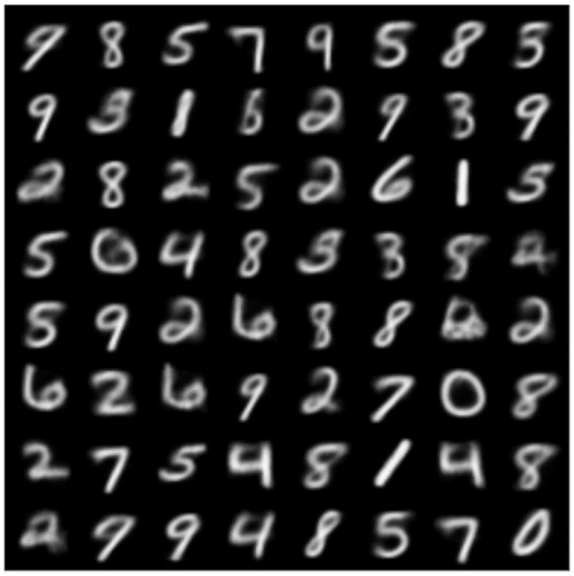
Results

The GAN generated this image after training for 10 Epochs on a subset of MNIST dataset.



Training beyond 10 epochs does not increase the quality of results much, instead it makes the digits fade and the fading increases with more training, so this is why the training is capped at 10 epochs.

The VAE generated this image after training for about 5 epochs on a subset of the MNIST dataset.



Training for more than 5 epochs did not make the results any better so it was capped at 5 epochs.

Analysis

If we visually compare the outputs, we can see that the variational autoencoder tends to produce images that are blurry as compared to the generative adversarial network. This is because when we reduce the images to just a vector, its information gets compressed and the compressed result we get is the mean and variance. When we do this for all the images, we are basically training the VAE to learn the mean and variance of the entire dataset and recreate the images from that. This is similar to averaging out the images which is also known as blurring.

GAN is not without its faults as well. It can produce images that do not make any sense at the start since it is using noise for generation as compared to VAE which starts out with data that is relevant to the images it is trying to produce. This can also sometimes lead to

GAN not converging. Overall, however, the GAN does produce better quality images as compared to the VAE even if at times the variety is a bit limited. So, in conclusion, GAN is a better choice for producing fake images.

CODE

The code can be found at the repository below: https://github.com/ali18997/Fake-Image-Generation

REFERENCES

1. Ian Goodfellow, M. Mirza, B. Xu, Y. Benjio. *Generative Adversarial Network*. Department of Computer Science and Research Operations, University of Montreal (2014).
2. Ian Goodfellow. NIPS 2016 *Tutorial: Generative Adversarial Networks*, from NIPS conference(2016).
3. Diederik P. Kingma, Max Welling. *Auto-Encoding Variational Bayes*, Machine Learning Group, Universiteit van Amsterdam (2014).
4. Jaydeep T. Chauhan. *Comparative Study of GAN and VAE*, International Journal of Computer Applications (0975 - 8887), Volume 182 - No.22, October 2018.