Comparison Analysis of Generative Models

Liwei Chang*
Department of Chemistry
University of Florida
liweichang@ufl.edu

Abstract

Deep learning has achieved great success with application in various fields. Among many deep learning models have been developed, **Generative Adversarial Nets** and **Variational Autoencoders** are shown to be promising and improved models based on original architecture are still of interest for researchers in both theoretical and application area. In this work, we describe the underlying principle of these two models and show the comparison based on their performance with **MNIST** dataset. In addition, two extended model **DCGAN** and **InfoGAN** are explored with **MNIST** and **Fashion-MNIST** datasets.

1 Introduction

One goal of unsupervised learning is to learn the pattern of unlabelled samples that is representative of all associated features. Many natural tasks, such as face recognition, are possible to solve with such framework. For example, we can represent a cat picture by its face and color, body and hair type, which we generally use as essential attributes of samples. Once the model "understands" those attributes for a given dataset, we should be able to create a new sample from its learned patterns [1]. All types of generative models aim at learning the true data distribution of the training set so as to generate new data points from this distribution with some variations. The most promising models are Generative Adversarial Nets (GAN) [2] and Variational Autoencoders (VAE) [3].

With the motivation to understand GAN and VAE, we present the basic principles in the first section and then show the results of pytorch-implemented models on MNIST and Fashion-MNIST datasets. Finally, we give a brief review of improved GAN models and test our datasets on two of them: DCGAN and InfoGAN.

2 Background

2.1 Generative Adversarial Nets

Generative Adversarial Nets is a framework for training generative model using a minmax strategy introduced by Goodfellow et al.[2] There are two distinct players, a generator and discriminator. The generator creates samples from a certain distribution and the discriminator is tasked to tell if the generated sample is in training dataset or not. The underlying idea is clear from the following function:

$$\min_G \max_D (\mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))])$$

There are two essential steps: 1) we generate a noise variable $z \sim P_{noise}(z)$ and transform it to the generator $G(z)$; 2) the generator is trained by playing against discriminator D that aims to distinguish

*https://perez.chem.ufl.edu/people/group/

Preprint. Under review.

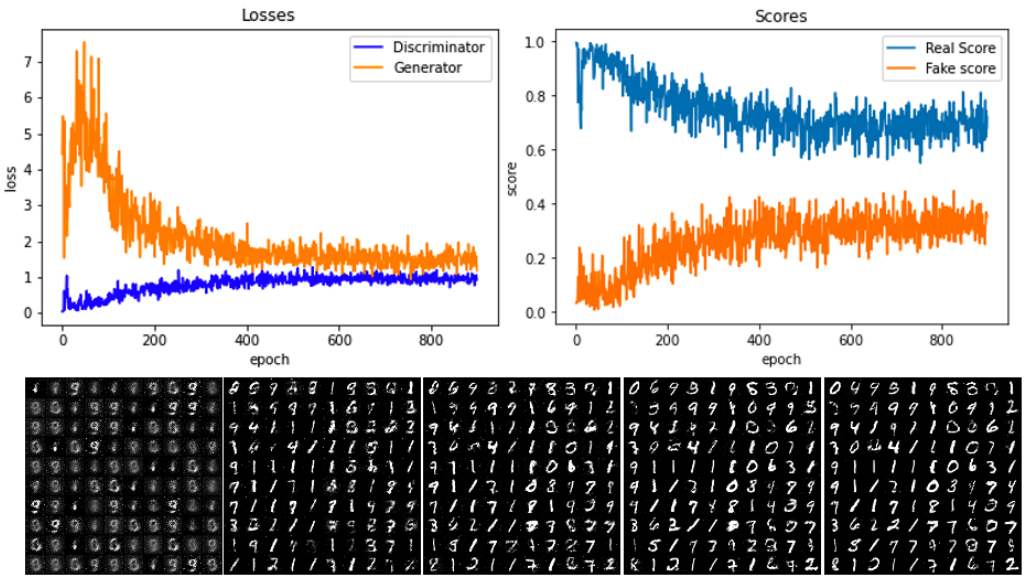

Figure 1: GAN training loss; scores; generated sample after training 1, 100, 200, 400, 800 iterations.

between samples from training samples distribution P_{data} and generated sample distribution P_G. In practice, we use binary cross entropy as loss function and we can see that the optimal $D = \frac{P_{data}}{P_{data}+P_G}$ to achieve Nash-equilibrium.

2.2 Variational Autoencoders

The concept of variational autoencoders was introduced by Diederik P Kingma and Max Welling in their paper **Auto-Encoding Variational Bayes** [3]. Their contribution based on traditional autoencoder is: 1) they showed that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. 2) for i.i.d. datasets with continuous latent variables per sample, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. We assume a generative with a latent variables distributed according to some distribution $p(z_i)$, the observed variable is distributed according to conditional distribution $p(x_i|z_i, \theta)$. We implement $p(x_i|z_i, \theta)$ and $q(z_i|x_i, \phi)$ as neural networks, which can be seen as probabilistic decoder and encoder respectively [4]. In this work, we choose Gaussian distributions for our conditional distribution q, so the output of encoder network are average μ and covariance $\sum$. The optimization strategy is derived from the following: we want to generate samples that have high probability of belonging to the same distribution as the observed data X, i.e. $argmaxP(X|z)$, where $z \sim N(\mu, \sum)$. Notice that with variational parameters ϕ, we have:

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x})\right] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right]$$

The second term in the above equation is the the Kullback-Leibler (KL) divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$, which is non-negative. The first term is the centerpiece called *variational lower bound* or the *evidence lower bound* (ELBO) [4]. In this work, we use binary cross entropy and KL-divergence to perform optimization, which can be seen as reconstruction error and difference from observed data.

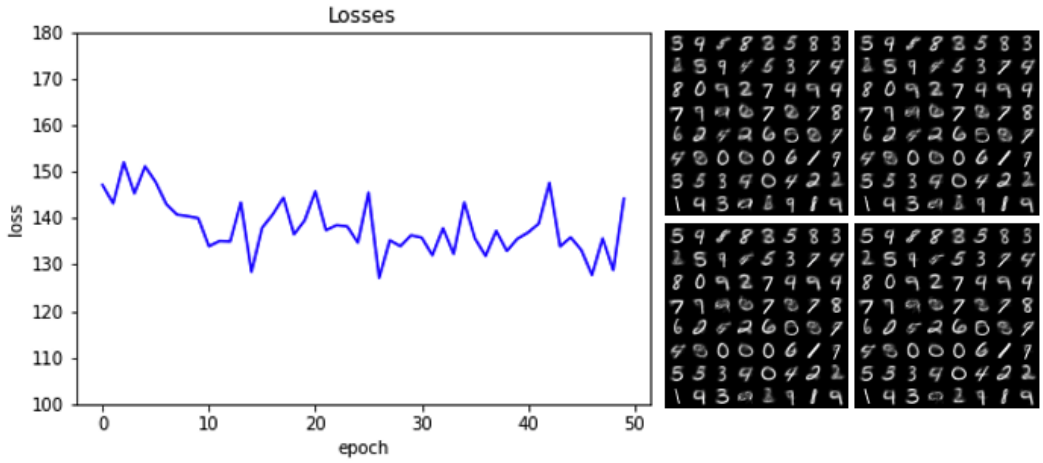

Figure 2: VAE training loss; scores; generated sample after training 1, 10, 20, 40 iterations.

3 Result

3.1 Implementation

The network architecture of vanilla version GAN and VAE are constructed by stacked linear layers, training dataset is MNIST (60000 samples with size 28*28 each). Details are shown below:

Generator		Encoder	
Latent variable	$z \in R^{64}$	MNIST sample	$z \in R^{784}$
1st linear layer	$64 \rightarrow 256+$ ReLU	1st linear layer	$784 \rightarrow 512+$ ReLU
2nd linear layer	$256 \rightarrow 256+$ ReLU	2nd linear layer	$512 \rightarrow 256+$ ReLU
3rd linear layer	$256 \rightarrow 784+$ tanh	3rd linear layer	$256 \rightarrow 2$

Discriminator		Decoder	
MNIST sample	$z \in R^{784}$		
1st linear layer	$784 \rightarrow 256+$ LeakyReLU	1st linear layer	$2 \rightarrow 256+$ ReLU
2nd linear layer	$256 \rightarrow 256+$ LeakyReLU	2nd linear layer	$256 \rightarrow 512+$ ReLU
3rd linear layer	$256 \rightarrow 1+$ Sigmoid	3rd linear layer	$512 \rightarrow 784$ + Sigmoid

$\text{ReLU}(\mathrm{x}) = \max(0, \mathrm{x})$

$\text{LeakyReLU}(\mathrm{x}) = \{^{x \quad x \geq 0}_{0.2x \quad x<0}$

$\text{Sigmoid}(\mathrm{x}) = \frac{1}{1+\exp(-\mathbf{x})}$

Loss: $\ell(\mathrm{x}, \mathrm{y}) = \mathrm{L} = \{\mathrm{l}_1, \ldots, \mathrm{l}_\mathrm{N}\}^\top, \quad \mathrm{l}_\mathrm{n} = -\mathrm{w}_\mathrm{n}\left[\mathrm{y}_\mathrm{n} \cdot \log \mathrm{x}_\mathrm{n} + (1 - \mathrm{y}_\mathrm{n}) \cdot \log\left(1 - \mathrm{x}_\mathrm{n}\right)\right]$, where N is the batch size 100.

3.2 Experiments

The goal of those simple linear networks is to investigate if the representation of samples are effective for training, if the optimization strategy works and the quality of generated sample.

GAN. For discriminator, 1) we expect the discriminator to output 1 if the image was picked from the real MNIST dataset, and 0 if it was generated; 2) we first pass a batch of real images, and compute the loss, setting the target labels to 1; 3) we generate a batch of fake images using the generator, pass them into the discriminator, and compute the loss, setting the target labels to 0; 4) finally we add the two losses and use the overall loss to perform gradient descent to adjust the weights of the discriminator. For generator, 1) we generate a batch of images using the generator, pass them into the discriminator; 2) we calculate the loss by setting the target labels to 1 i.e. real; 3) we use the loss to perform gradient

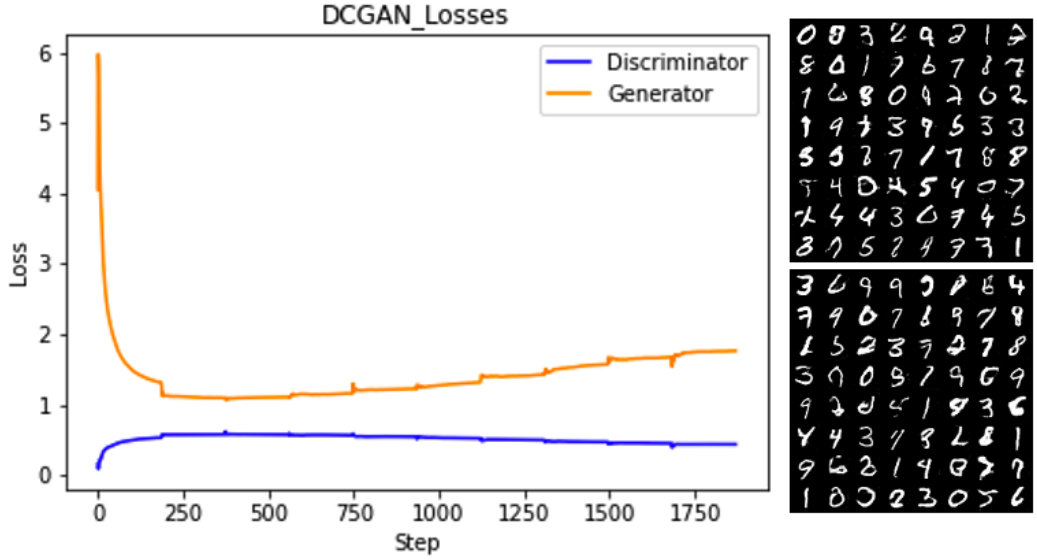

Figure 3: DCGAN training loss; generated sample after training 1500 iterations.

descent i.e. change the weights of the generator, so it gets better at generating real-like images. In Fig 1, the loss for both Discriminator and Generator are expected in that generated samples will become harder and harder for Discriminator to judge if it is from true samples. This trend can also be seen from the score for Discriminator and Generator shown in Fig 2.

VAE. We input samples in batch to calculate the mean and variance from two output layers in encoder. Then we sample from $z \sim N(\mu, \sum)$ and use this as input to the decoder. The loss is calculated from binary cross entropy and KL-divergence for the output of decoder as discussed in 2.2. Our experiment shows that it is much easier to train the VAE model from the convergence of training loss and generated sample quality. Despite of this, we found its inability to generate samples with sharp features and we will discuss this issue in the final section.

4 Exploration

GAN has been known to be unstable to train, often resulting in generators that produce nonsensical outputs as also shown in our results. There are several improvement upon original GAN framework, we show two of them in this section: Deep Convolutional GANs [5] and InfoGAN [6].

4.1 DCGAN

Supervised learning with convolutional networks (CNN) has seen huge application in computer vision. Despite this, applying CNN to unsupervised learning hasn't been very successful until several approaches are developed: 1) replacing deterministic spatial pooling functions (e.g. maxpooling) with strided convolution; 2) using Batch Normalization to stabilizes learning by normalizing the input to each unit to have zero mean and unit variance, which is supposed to solve gradient related issues in deeper networks and training problems due to poor initialization; 3) using ReLU (or LeakyReLU) activation function except for the output layer [5]. The following table shows our implementation.

Generator		Discriminator	
Latent variable	$z \in R^{100}$	MNIST sample	$z \in R^{784}$
fully-connected layer + batchnorm	LeakyReLU	1st conv layer	LeakyReLU
2nd deconv layer + batchnorm	LeakyReLU	2nd conv layer	LeakyReLU
3rd deconv layer	tanh	fully-connected layer	Sigmoid

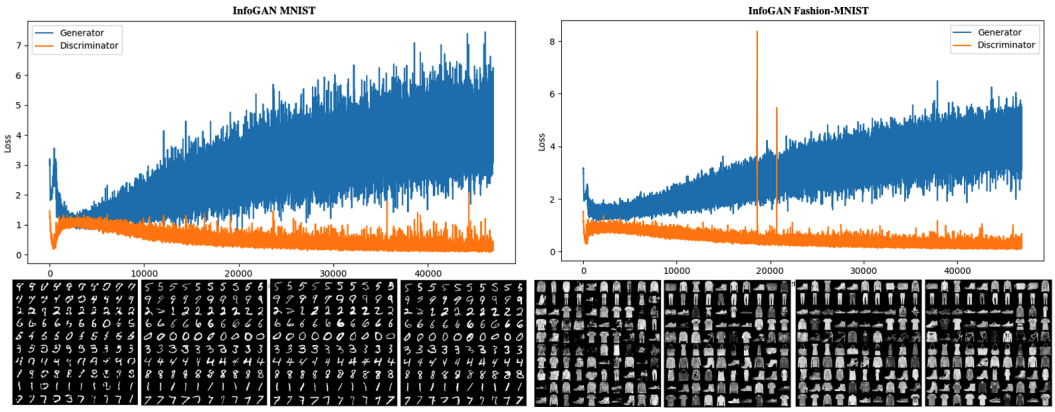

Figure 4: InfoGAN training loss for MNIST and Fashion-MNIST; generated sample after training 500, 5000, 10000, 40000 iterations.

In addition to the above implementations, we use Adam optimizer [7] for training and the learning rate in Generator is 0.0002. We take the same MNIST dataset for training, the loss and generated samples are shown in Fig 3.

CNN in general find areas of correlation within an image. From the above results, we can see the generated images become less blurry compared with vanilla GAN. The total training time (< 10 min) is less than vanilla GAN to make the loss tend to converge, although the loss shows increase with more training epoches, which may reflect the overfitting of the current model. Overall, GAN with convolutional networks work better in generating images.

4.2 InfoGAN

The original GAN framework uses a simple continuous input noise vector z without any restriction on how the Generator may use this vector. As our results indicate, the Generator use the input vector in a highly entangled way, which may be the explanation for the blurriness of generated samples. InfoGAN was proposed following the idea of disentangle representation and maximizing mutual information for inducing latent variables [6]. This method introduces the latent codes c in addition to noise z in Generator. Based on mutual information $I(c; G(z, c))$ between generator distribution $G(z, c)$ and latent codes c, if c and $G(z, c)$ are independent, then $I(c; G(z, c)) = 0$ as knowing latent code c has no effect on noise distribution z. Therefore, the original GAN min-max strategy can be reformulated into:

$$\min_G \max_D V_I(D, G) = V(D, G) - \lambda I(c; G(z, c))$$

where λ is a regularization hyperparameter set to 1 in our implementation. Following general variational approach, we introduce an auxiliary distribution $Q(c|x)$ to approximate posterior distribution $P(c|x)$. The variational lower bound is derived in the paper as the following:

$$L_I(G, Q) = E_{c\sim P(c), x\sim G(z,c)}[\log Q(c \mid x)] + H(c) = \\ E_{x\sim G(z,c)}\left[\mathbb{E}_{c'\sim P(c|x)}\left[\log Q\left(c' \mid x\right)\right]\right] + H(c) \leq I(c; G(z, c))$$

Similar to the implementation in the paper, we use a neural network to represent the distribution Q. There are two type latent codes are introduced: discrete categorical latent code c_i and continuous latent code c_j, which is simply treated as a factored Gaussian. Due to difficulty in training original GAN, we take advantage of convolutional networks in the experiments. The implementation details are shown below.

Generator		Discriminator / Q	
Latent variable	$z \in R^{74}$	MNIST sample	$z \in R^{784}$
1st deconv layer + batchnorm	ReLU	1st conv layer	LeakyReLU
2nd deconv layer + batchnorm	ReLU	2nd conv layer	LeakyReLU
3rd deconv layer + batchnorm	ReLU	2nd conv layer	LeakyReLU
4th deconv layer	Sigmoid	fully-connected layer	Sigmoid

We test the model on two datasets: MNIST and Fashion-MNIST. Output for discrete labels and Gaussian variable $\mu; \sum$ are calculated by batch normalization of Discriminator network.

The loss and generated samples in Fig 4. show that introducing more conditions instead of training solely on random noise is able to generate more similar samples then original GAN. The effects of discrete variables (i.e. supervised labels) and continuous variables are discussed in the paper, which shows that labels work as classifiers that help identify different latent representations and Gaussian variables can be used to control the rotation and width [6].

5 Discussion

This work contains the implementation and results for two vanilla models: GAN and VAE. From the experiments on MNIST dataset, we found that both models are able to generate samples similar to true samples, but there are several drawbacks: 1) simple linear layer are not capable to train GAN as the model converges to the local minima from which it cannot generate samples with correct correlations of all parts in one sample. 2) the blurry image problem can be seen from VAE, which could be explained by following reasons: only simple probability distribution is used, recent work, such as Normalizing Flows [8], shows that improving the flexibility of inference models can help solve this issue by providing a tight bound; the nature of variational approximation used could also explain the blurriness problem. Two improved GAN includes DCGAN and InfoGAN are also investigated, both models show promising performance. It should be pointed out that in training both models, the loss of Generator shows increase after many iterations. This could mean that training is completed as generator cannot be further improved or the overfitting of discriminator.

References

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 35, no. 8, pp. 1798–1828,* 2013

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS, 2014, pp.* 2672–2680.

[3] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *ArXiv preprint arXiv:1312.6114,* 2013.

[4] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends in Machine Learning, vol. 12, no. 4, pp. 307–392,* 2019.

[5] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *ArXiv preprint arXiv:1511.06434,* 2015.

[6] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets." CoRR, abs/1606.03657, 2016.

[7] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ArXiv preprint arXiv:1412.6980,* 2014.

[8] Ivan Kobyzev, Simon Prince, and Marcus Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2020.

[9] Please go to `https://github.com/ccccclw/Notebooks/tree/master/ML_2020Fall/src_gan_vae` for source code.