CS7140: Advanced Machine Learning **Spring 2020**

Lecture 13: Generative Adversarial Networks (GAN)

Lecturer: Rose Yu *Scribes: Matin Raayai Ardakani, Sumeet Parekh, Shivam Patel*

13.1 What is a generative Model?

Generative model tries to capture the joint distribution of $P(X, Y)$. We can use conditional decomposition and decompose $P(X, Y)$ and get $P(X|Y)$ and $P(Y)$

It allows to learn a simulator of the data. If we think about images we want to generate images, i.e. we want to model the distribution over pixel values p(x) given the category of the images.

Generative models allow us to estimate the density. Think about a natural language example what is the likelihood that someone is saying happy words given the factor that this is a positive review in the case of restaurant reviews. We want to estimate the conditional density given a particular label of the text and then we can estimate the likelihood of that data happening. This allows us to perform unsupervised learning of data.

13.1.1 Applications of Generative Model

We can use generative models to estimate climate models. we can simulate climate phenomenons use neural nets and sampling which much faster that the older models based on partial differentiation.

Generative models are being used to simulate the dynamics of COVID-19 spreading. It can also be used to estimate the density in the physics experiment of particles appearing in Large Hadron Collider experiments.

13.1.2 Why do we need Generative models?

Image super resolution: We want to take a low resolution image and generate a fine resolution image. The original image. We want to simulate the distribution of pixels at a much higher grid size.

We can use generative models for drug design. we want to generate structures of molecules then will become different drugs so we want to generate certain structure for molecules then will become different drugs so we want to estimate the distribution of different molecules occurring in their composition.

Image compression we want to transform an image from high resolution to a low resolution image. this allows to store important information while limiting the memory consumption.

Generative modeling is hard because when we try to estimate the joint distribution of p(x,y) the dimensionality of x and y can be extremely high and the values can be both continuous and discrete. If you want to plug that into neural networks it is very difficult to design the correct objective function how do we handle high dimensionality. how do we handle the complex correlation dependencies in the generative models.

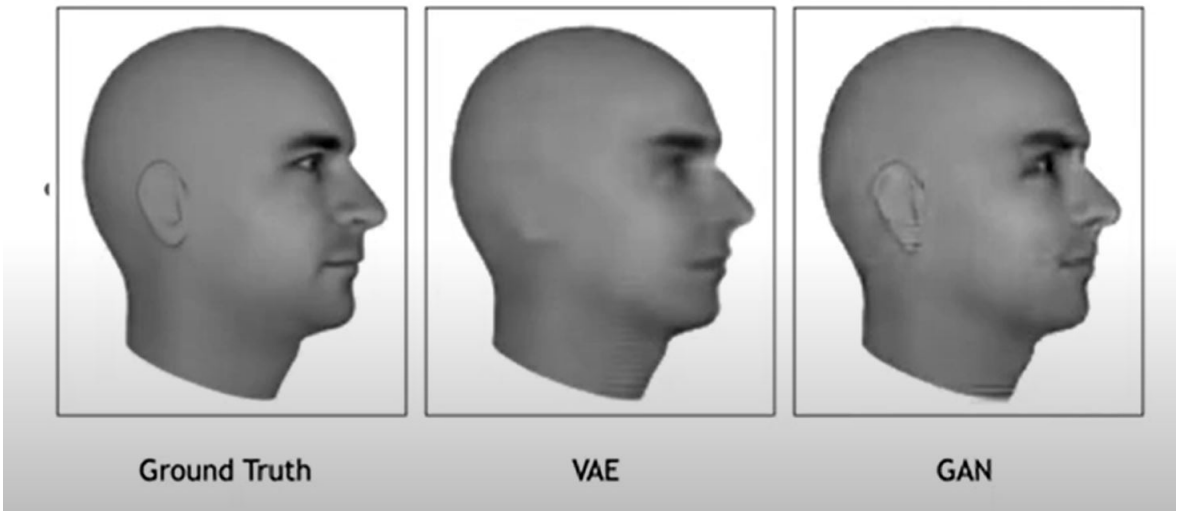


Figure 13.1: Predicting the next frame in the video given the sequence of input frames. Left is ground truth, middle is VAE and right one is from GANs.

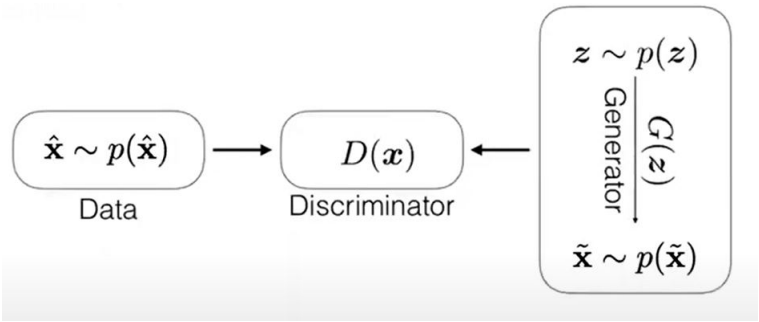


Figure 13.2: General Architecture of Generative Adverserial Networks

13.2 Generative Adversarial Networks

In figure 13.1 we wanted to predict the next frame in the video given the sequence of input frames we wanted to generate the next frame of an image. Left is ground truth, middle is from VAE and right one is from GANs. The reason that VAE gives a blurry image is that VAE is trying to optimize EBLO. It boils down to reconstruction and KL divergence. Reconstruction is computed using square loss which penalize all pixels equally. when looking at the image we see that all pixels cannot be penalized equally because they are geometric states that are unique at different regions of this face region.

There is no way that we can come up with the correct loss function that can give us the right description of complex shapes in image. Since we cannot come up with the correct loss function. we should just let the model learn the loss function. This is why we construct another neural network known as the discriminator. this discriminates between the generated data and the ground truth data.

Looking at the architecture of GAN in figure 13.2 from the left side we have a data distribution which is $\hat{\mathrm{x}}$. So $\hat{\mathrm{x}}$ is a sample from the observed data. So if we look from the right hand side $\widetilde{x}$ is the simulated data or the

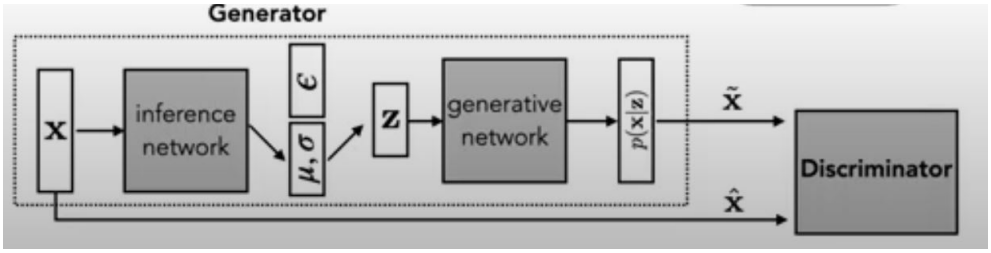


Figure 13.3: General Architecture of Variational Auto Encoders.

samples drawn from the generated model. The way we generate it is from random noise. So we have some random stochasticity z represents our prior knowledge so we transform the random noise z using a generator which in our case can we approximated by a neural network. So from left size and right side we have 2 distributions, one is data distribution and the other one is model distribution which is the data simulated from the generative model. The key difference here is that we are introducing another model which we call a discriminator D(x). D(x) can be approximated with a neural network and it is learning to differentiate between 2 distributions.

For example we can just modify the VAE models slightly so dashed box is the VAE model which we are familiar with. so we have the input data x we feed that through an inference network that gives us an conditional distribution $P(z|x)$ and then we put that into a generative network that gives us $p(x|z)$. So that is a VAE which we trained with ELBO but here we train with another neural network which is the discriminator. So the discriminator takes the output from the generator $\widetilde{x}$ and takes the ground truth data $\hat{x}$. the only modification is changing the loss function from ELBO to another neural network. the way we learn the discriminator is by treating it as a classification problem.

Learn the discriminator:

$$p(\text{data}\,|\mathbf{x}) = D(\mathbf{x}) \quad p(\text{ gen. }|\mathbf{x}) = 1 - D(\mathbf{x})$$

Bernoulli outcome: $y \in \{$ data, gen. $\}$

$$\log p(y|\mathbf{x}) = \log D(\hat{\mathbf{x}}) + \log(1 - D(\tilde{\mathbf{x}}))$$

zero-sum game:

$$\min_G \max_D V(D,G) = \min_G \max_D \mathbb{E}_{p(\hat{\mathbf{x}})}[\log D(\hat{\mathbf{x}})] + \mathbb{E}_{p(\tilde{\mathbf{x}})}[\log(1 - D(\tilde{\mathbf{x}}))]$$

This is a Bernoulli random variable where we are saying that the probability of the input x being data is D(x) and therefore the probability of x coming from generator is 1-D(x). There are 2 outcomes for the discriminator, either it is the data or it is from the generator. We know from classification, if we want to model a Bernoulli random variable then we take the log of that so the conditional distribution of p(y) which is a label given input feature x. What we can do in terms of optimization is we want to take the expected likelihood for every data point that we draw from the data distribution if you want to compute the statistic over the entire data set we need to take the expectation over the probability distribution.

We are trying to optimize the expected log likelihood of $p(\hat{x})$ we are taking the expectation over $\widetilde{x}$ which is generated distribution. there is something that is hidden in the $\widetilde{x}$ because $\widetilde{x}$ is a function of the generator G. We want to find 2 optimizees one is discriminator and the other is a generator and we want to maximise the log likelihood of the discriminator and then we want to minimise the chances where the generator can successfully fool the discriminator.

So the generator minimizes the log probability of the discriminator being correct. If the generator is strong then it will generate realistic data. The discriminator would have difficulty classifying between the generated

data and the ground truth data. If the discriminator is always correct that means the data being generated is quite different from the truth distribution meaning the generator is weak. You can see this competition between the 2 optimizees.

We can optimize this bi-variate objective function using an alternating optimization strategy.

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{p(\hat{\mathbf{x}})}[\log D(\hat{\mathbf{x}})] - \frac{1}{2}\mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z}))$$
$$J^{(G)} = -J^{(D)}$$

Normally when dealing with multi variate optimization problem naturally we optimize 1 variable at a time. Given by fixing all the other variables, which is similar to Gibbs sampling. Here we are dealing with a 2 variable case. Fix the generator and then optimize over the discriminator. Making this the function of the discriminator D giving $J^{(D)}$. next we will fix the value for the discriminator and optimize over the generator.Since we are solving this problem as a gradient descent algorithm then it is easier to formulate everything as a minimization problem. hence the generator equation has a negative sign.

13.3 Training GAN Algorithm

- Repeat
  - Train Discriminator for k steps
    - sample z, x
    - update discriminator $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))$
  - Train Generator
    - sample z
    - update generator $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))$
- Until converge

13.4 Alternative Interpretation: Hypothesis Testing

In this section, we will be discussing the alternative interpretation for Generative Adversarial Networks (GANs). The alternative interpretation can be characterized by Hypothesis Testing. Hypothesis testing is a statistic for interpretation of the GAN objective function.

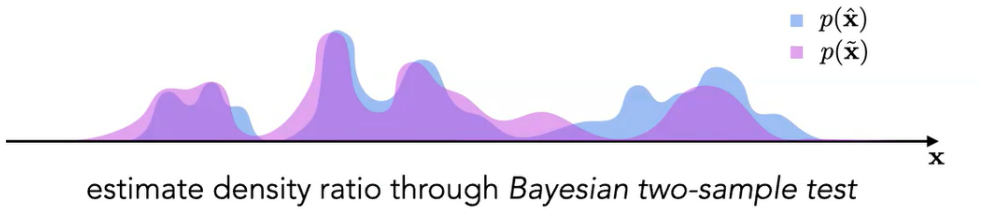


Figure 13.4: Estimate density ratio through Bayesian two-sample test.

In figure 13.4, we are looking at two distributions. The first one is the data distribution represented by $p(\hat{x})$ and the other is one is the general model distribution represented by $p(\tilde{x})$. $p(\tilde{x})$ is the purple curve while $p(\hat{x})$ is the blue curve. If we wanted to estimate the density of the distributions and want to calculate the values of $p(\hat{x})$ and $p(\tilde{x})$, we can use the technique called the Bayesian two-sample test.

Given the data distribution $p(\hat{x})$ and generated distribution $p(\tilde{x})$, we can compute the ratio of the two distributions as follows.

$$\frac{p(\hat{x})}{p(\tilde{x})} = \frac{p(x|data)}{p(x|gen.)} \tag{13.1}$$

Using Baye's rule, we can rewrite the equation as:

$$\frac{p(\hat{x})}{p(\tilde{x})} = \frac{p(data|x)p(x)/p(data)}{p(gen.|x)p(x)/p(gen.)} \tag{13.2}$$

Assuming that the prior distribution of $p(data)$ and $p(gen.)$ to be the same (we assume that the marginal distribution of the blue curve and the purple curve are similar), then the data distribution and general distribution ratio will become:

$$\frac{p(\hat{x})}{p(\tilde{x})} = \frac{p(data|x)}{p(gen.|x)} \tag{13.3}$$

Thus, the density estimation problem now becomes a sample discrimination task.

Under an ideal discriminator, we can prove that the generator minimizes the Jensen-Shannon divergence. Going back to the original problem, we were trying to optimize the objective function over two variables, the discriminator D and generator G.

$$min_G max_D V(D^*, G) = 2D_{JS}(p(\hat{x})||p(\tilde{x})) - 2log2 \tag{13.4}$$

The Jenson Shannon divergence is related to the KL-Divergence. It is the distance between two distributions where the two distributions are $p(\hat{x})$ and $p(\tilde{x})$.

We know that KL-Divergence is not symmetric (KL Divergence between p and q is not the same as the KL Divergence between q and p). In Jenson Shanon divergence, it constructs p as $p(\hat{x})$ and q as the average of the two distributions ($\frac{1}{2}(p(\hat{x}) + p(\tilde{x}))$).

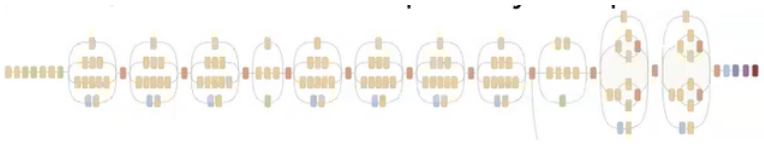


Figure 13.5: Inception v3 Model

$$D_{JS}(p(\hat{x})||p(\tilde{x})) = \frac{1}{2}D_{KL}(p(\hat{x})||\frac{1}{2}(p(\hat{x}) + p(\tilde{x}))) + \frac{1}{2}D_{KL}(p(\tilde{x})||\frac{1}{2}(p(\hat{x}) + p(\tilde{x}))) \tag{13.5}$$

The above equation has two KL Divergence. The first one is the KL divergence between data distribution and the mean of the distribution while the second one is the KL divergence between the generative distribution and the mean of the distributions. If we swap values of $p(\hat{x})$ and $p(\tilde{x})$, the value still remains the same and thus Jenson Shanon divergence is actually symmetric.

13.5 Evaluation

We cannot use simple loss functions like Mean Square Error (MSE) loss or Cross-entropy loss to evaluate generative models as we are not trying to optimize the lower bound of likelihood and hence it would be difficult to quantify the performance of GANs.

One common technique used to evaluate generative models such as GANs is the **inception score**. The idea is to take a very complex discriminative model (in this case the Inception v3 shown in figure 13.5 in the figure above) to quantify class and distribution entropy. This is a pre-trained model. Instead of using humans to discriminate between ground truth and generative outputs, we will be using a separate model to perform the task of discrimination.

$$IS(G) = e^{(E_{p(\tilde{x})}D_{KL}(p(y|\tilde{x})||p(y)))} \tag{13.6}$$

The equation above is used to compute the inception score for the generative model. Here, $p(y)$ is the marginal class distribution and $p(y|\tilde{x})$ is the class distribution for a given image. The inception score is the exponential between the expected KL divergence of the class distribution for a given image and the marginal distribution. The distribution $p(y|\tilde{x})$ should be highly peaked (low entropy) because the inception score is well trained and hence has very high confidence of the correct class. $p(y)$ can be represented as follows:

$$p(y) = \int p(y|\tilde{x})d\tilde{x} \tag{13.7}$$

This distribution is independent from the generated distribution. Ideally, we would want want this distribution to be uniform because we want this discriminator model to be well performing regardless of the generated distribution.

In practice we are going to sample a bunch of images and feed that into the inception train model and take the output of the inception train model (which is the class distribution) and compute the difference between the class distribution and distribution over different classes. This can be represented by:

$$IS(G) \approx e^{(\frac{1}{M}D_{KL}(p(y|x^{(i)})||\tilde{p}(y)))} \tag{13.8}$$

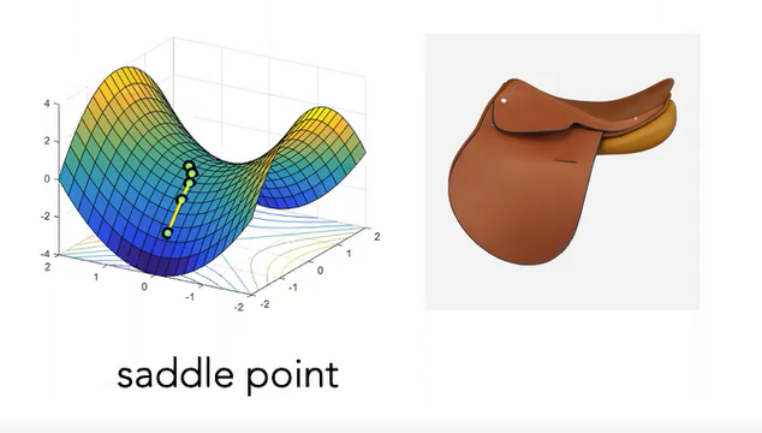


Figure 13.6: This figure shows the saddle point plot on the left and how it looks similar to a saddle shown on the right.

where $\tilde{p}(y)$ is:

$$\tilde{p}(y) = \frac{1}{N}\sum_{i}^{N} p(y|x^{(i)}) \tag{13.9}$$

Here M is different type of generative model. The marginal distribution $\tilde{p}(y)$ is the average of class distribution over all possible images. We can compute the marginal class distribution by feeding in all possible images to the inception model and take the average of the output from the inception train model. The output from the inception train model to compute the KL divergence and that will be averaged over different models and then take the exponent. This will give a reasonable number that can quantify the performance of the generated samples.

13.6 Difficulty of Training GANs

13.6.1 Saddle Optimization Problem

Let us look at the objective function of the Mini-max game. The objective function has two variables D and G. It is a saddle point optimization problem.

$$min_G max_D V(D,G) \tag{13.10}$$

The plot looks like a saddle as represented by the figure 13.6. If we follow the x-axis of the left plot, then this is a minimization problem and if we follow the y-axis then this is a maximization problem. Thus, it is very unstable and is highly non-convex since there is more than one global optima. The generator is optimizing the saddle point. However, the region of saddle point is very small and everything else basically just slips off making it very difficult for the optimization to remain stable.

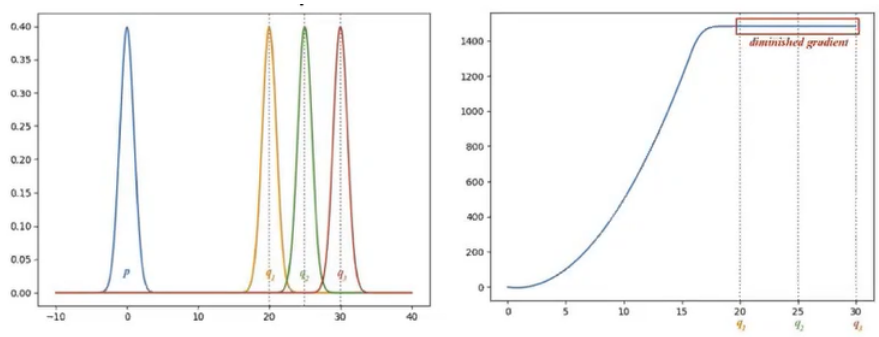


Figure 13.7: The plot on the left shows the distributions p and q and the plot on the right shows the diminishing gradient.

13.6.2 Vanishing Gradient Problem

Another difficulty of training GANs is the vanishing gradient problem in the loss function. GANs tries to optimize the Jensen-Shannon divergence for p and q.

In the figure 13.7, the plot on the right shows the gradient plot for p and q if we try to minimize the Jensen-Shannon divergence between pq. p is represented by the blue curve in the left plot and q is variant and is represented by $q1$, $q2$ and $q3$. Therefore, the mean of distribution q is changing. As you can see from the right plot, the curve becomes flat and thus the gradient of the curve becomes zero. If the gradient of the objective function becomes zero, then there is no way for the objective function to improve anymore and is stuck in the local optima. This corresponds to the diminishing or vanishing gradient problem.

13.6.3 Mode Collapse

When we model data distributions in the real world, most of them are multi-modal. Let us say that we want to jointly model the text distribution of words and the distribution of pixels. Even though they are all high dimensional distributions, the mean and variance of this distribution can be very different. A simple example of multi-modal is Gaussian distribution because every Gaussian distribution has a different mean and maybe different variance.

Let us say we are trying to model MNIST data shown in figure 13.8. It is the distribution over pixels. The figure above shows how the generated distribution is changing for the different number of iterations from 10K steps to 100K steps. The top row is the ground truth while the bottom row are the samples coming out of the generator. As you can see from above, the generator only generates a very uniform homogeneous digit. On the other hand, the ground truth data is highly heterogeneous. The reason this happens is because we are trying to train the two models (generator and discriminator) simultaneously and the generator is generating data independent of the latent variable.

13.6.4 Discontinuity

Consider again, the objective function of GAN training:

$$D_{JS}(p(\hat{x})||p(\tilde{x})) = \frac{1}{2}D_{KL}(p(\hat{x})||\frac{1}{2}(p(\hat{x}) + p(\tilde{x}))) + \frac{1}{2}D_{KL}(p(\tilde{x})||\frac{1}{2}(p(\hat{x}) + p(\tilde{x}))) \tag{13.11}$$

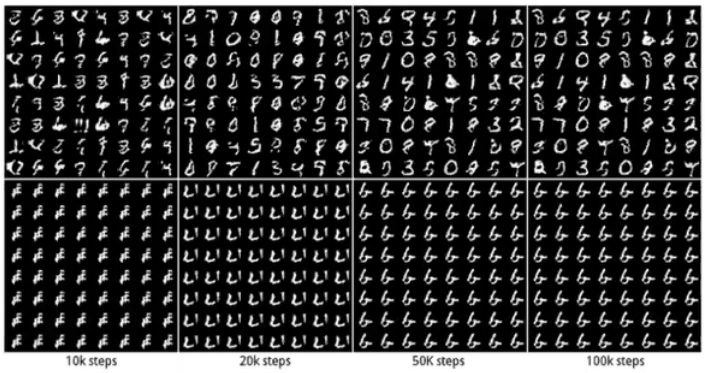


Figure 13.8: The figure displays the MNIST Data with the ground truth values showed in the top row and the generated output on the bottom row.

Remember that with an ideal discriminator, we could prove that the generator minimizes the Jensen-Shannon Divergence of the two distributions and one of its advantages was its symmetricity, meaning its value did not depend on the order of the input data distributions.

However, One disadvantage of Jenssen-Shannon divergence is its discontinuity. For example, consider a uniform random variable $Z \sim U[0,1]$ from which we construct the data distribution and model distribution by uniformly sampling $p(\hat{x}) \sim (0, Z)$ and $p(\tilde{x}) \sim (\theta, Z)$, where θ is the model parameter. Therefore, the JS Divergence of these two distributions with respect to the generative model parameter will be:

$$D_{JS}(p(\hat{x})||p(\tilde{x})) = \begin{cases} \log(2) & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases} \tag{13.12}$$

Jensen-Shannon Divergence and Wasserstein Distance of the data distribution and the model distribution, when $Z \sim U[0,1]$, $p(\hat{x}) \sim (0, Z)$ and $p(\tilde{x}) \sim (\theta, Z)$.

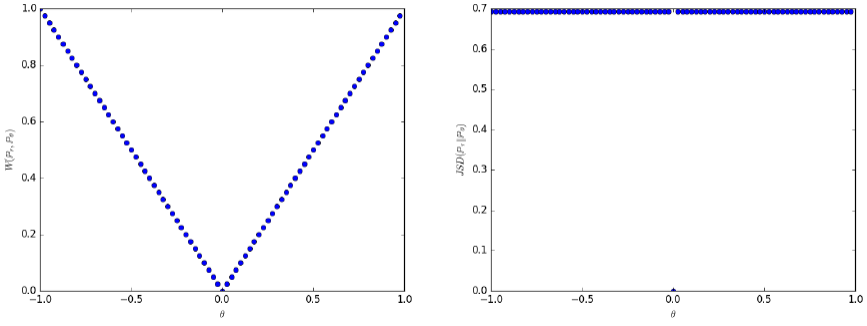
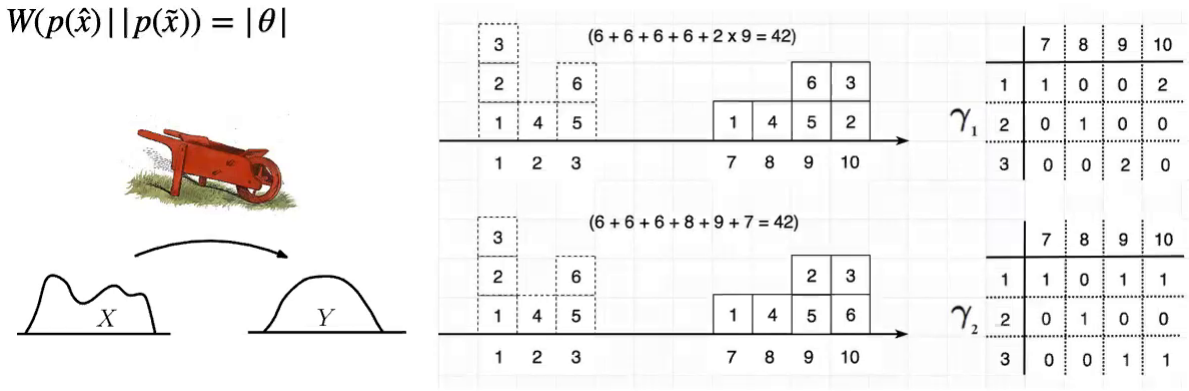


Figure 13.6.4 shows the plot of the two distribution's D_{JS} value with respect to θ. We can see that the D_{JS} is discontinuous when $\theta = 0$. From an optimization perspective, this means that the gradients won't flow normally, which makes training GANs more difficult. This is the motivation for this next line of work, the Wasserstein GANs [Arjovsky et al.(2017)Arjovsky, Chintala, and Bottou].

Figure 13.9: Discrete example of calculating Wasserstein distance between two discrete probability distributions.



13.7 Wasserstein GANs

To address the the problem mentioned previously, Wasserstein GAN uses a different objective function that is continuous and differentiable almost everywhere, called the "Wassserstein's Distance" or the "Earth Mover's Distance":

$$W(p(\hat{x})||p(\tilde{x})) = \inf_{\gamma \in \Pi(p(\hat{x}),p(\tilde{x}))} \mathbb{E}_{(\hat{x},\tilde{x})\sim\gamma}[||\hat{x} - \tilde{x}||] \tag{13.13}$$

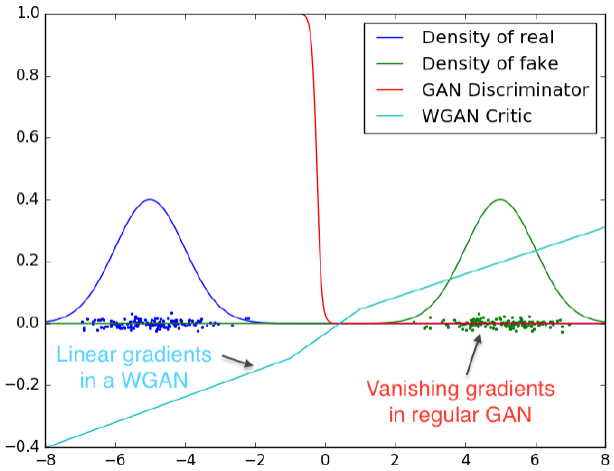
Using the Wasserstein's Distance, the objective function's value in the previous example is now $W(p(\hat{x})||p(\tilde{x})) = |\theta|$, which is continuous everywhere compared to Jenssen-Shannon's Divergence. Figure 13.6.4 shows the Wasserstein's Distance of the two distributions with respect to θ.

Formally speaking, the Wasserstein distance between two distributions is the lower bound ($infimum$) of the expectation of a metric γ, the cost of shifting the densities between the two distributions $p(\hat{x})$ and $p(\tilde{x})$. Therefore, the expectation term of the Wasserstein GAN can be regarded as a weighted summation of the distances between the two distributions. In order to compute it, we draw a sample from the data distribution ($\hat{x}$) and another sample from the model distribution ($\tilde{x}$), calculate the distance between these two data points and multiply each one of them by a weight, which is drawn form the cost function, γ. Finally, to derive the final Wasserstein distance, we select the minimum found cost of transporting points between the two distributions.

To understand this process better, let's take a look at the discrete example in figure 13.9. Assume that we have two discrete distributions X located on the left side of the figure representing $\hat{x}$, and Y located on the right side of the figure representing $\tilde{x}$. To calculate the Earth Mover's distance between the two distribution, our goal is to move points from the left distribution to the right distribution at minimum cost.

The middle plot of figure 13.9 shows two ways of moving points from the histogram of the left distribution to the right one, where the number in each box of the histograms represents a unique point and boxes with the same number represent the same point displaced from both distributions. The right plot of figure 13.9 simply summarizes the middle plot using matrices γ_1 and γ_2, with the x-axis being the destination and the y-axis being the source of the points displaced and the values being the number of points displaced from a particular source to a particular destination. Finally, in order to calculate the distance weight associated with each point, we simply multiply each element in γ by the amount of its displacement on the x axis, which is shown on the equations in the middle of figure 13.9. This can be easily interpolated to the continuous

Figure 13.10: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. The WGAN critic provides very clean gradients on all parts of the space [Arjovsky et al.(2017)Arjovsky, Chintala, and Bottou].



version of calculating the Wasserstein distance.

In general, it's intractable to evaluate the Wasserstein distance since to find the minimum cost of moving points, we have to enumerate over all possible ways of moving points between two distribution, which creates an exponential search space. In [Arjovsky et al.(2017)Arjovsky, Chintala, and Bottou], using the Kantorovich-Rubinstein duality:

$$W(p(\hat{x}), p(\tilde{x})) = \sup_{||f||_L \leq 1} \mathbb{E}[f(\hat{x})] - \mathbb{E}[f(\tilde{x})] \tag{13.14}$$

where the supremum is over all the 1-Lipschitz functions, the objective function is slightly transformed to the following to circumvent this problem:

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{p(\hat{x})}[D(\hat{x})] - \mathbb{E}_{p(\tilde{x})}[D(\tilde{x})] \tag{13.15}$$

Instead of evaluating the minimum cost between moving points, we sample a data point from the data distribution and feed that point into discriminator D, and the second term we sample from the Generator distribution $\hat{f}(\tilde{x})$. We then compute the weighted average for these two distributions separately, and finally, we maximize the discriminator and minimize the generator.

The discriminator here is the set of 1-Lipschitz functions (continuous), which can be enforced through weight clipping or gradient penalty. This means that when training the discriminator in the Wasserstein GAN the weights can be clipped to make the output bounded or the gradient of the discriminator can be penalized by setting the magnitude of the discriminator to a certain threshold to allow the weights of the discriminator to be 1-Lipschitz functions.

Let us observe how these changes gets reflected in the discriminator of the Wasserstein GAN in figure 13.10 for differentiating two Gaussians, one being the density of the real data distribution (represented by the color blue), and the other being the density of the model distribution (represented by the color green). If we use a regular (minmax) GAN discriminator between the two distributions (represented by the red-colored

Figure 13.11: Different GAN architectures trained with different methods. DCGAN is Deep Convolutional GAN and LSGAN is Least Squares GAN.

curve), its line becomes flat on the left side and its gradient vanishes. However, this is not the case if we use the Wasserstein objective function as reflected by the light blue line.

In other words, the main intuition behind Wasserstein GANs outperforming regular (minmax) GANs and addressing some of their shortcomings is that **the gradients of the regular (minmax) GAN saturates which leads to vanishing gradients whereas the Wasserstein GAN has a clean gradient on all points of the space**.

Figure 13.11 shows some example comparisons of Wasserstein and improved Wasserstein GAN performance compared to some other baselines [Gulrajani et al.(2017)Gulrajani, Ahmed, Arjovsky, Dumoulin, and Courville]. It can be seen that WGAN clearly has more diversity (can generate different types of rooms) meaning that it can discover new modes easily than regular GANs since it doesn't suffer from the vanishing gradient problem.

13.8 Applications of GANs

Since their introduction, numerous applications of GANs has been purposed. GANs surely have opened up some very exciting new research paths for the field of Machine Learning. Here, we present some "cool" applications of GANs. Our hope is that the students can think deeply about the technical challenges of GANs and even end up contributing to some of the fundamentals problems in generative models.

13.8.1 Image to Image Translation

In this application, GANs are used to transfer images from one domain or "style" to another. For example, [Isola et al.(2017)Isola, Zhu, Zhou, and Efros] translates the real aerial view of streets to their map view, or

Figure 13.12: Image-to-Image Translation with Conditional Adversarial Networks.

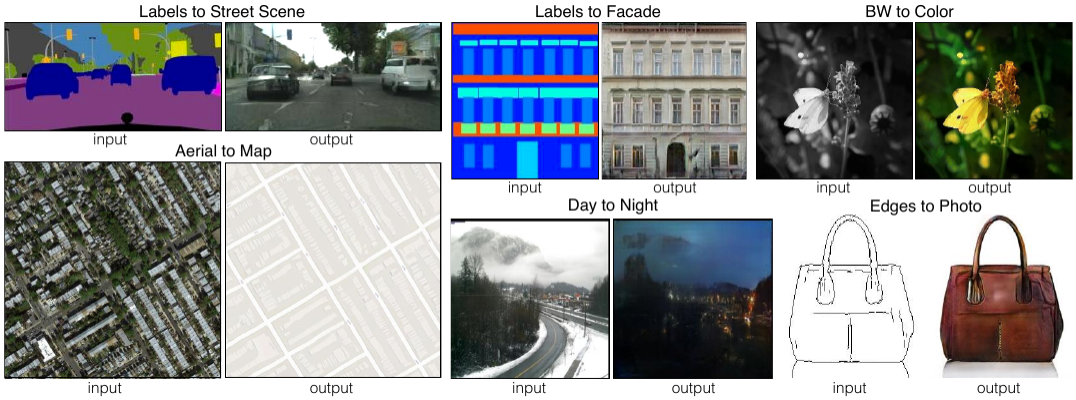


Figure 13.13: Cycle-GAN Translation with Generative Adversarial Networks.

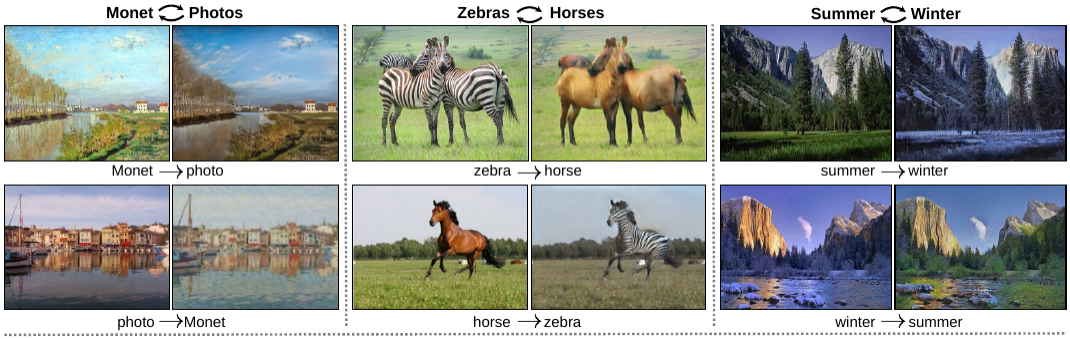


Figure 13.14: A typical jet simulation using GANs.

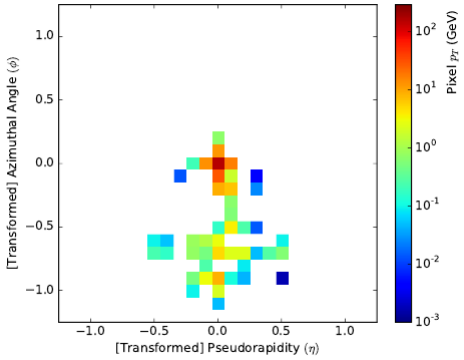
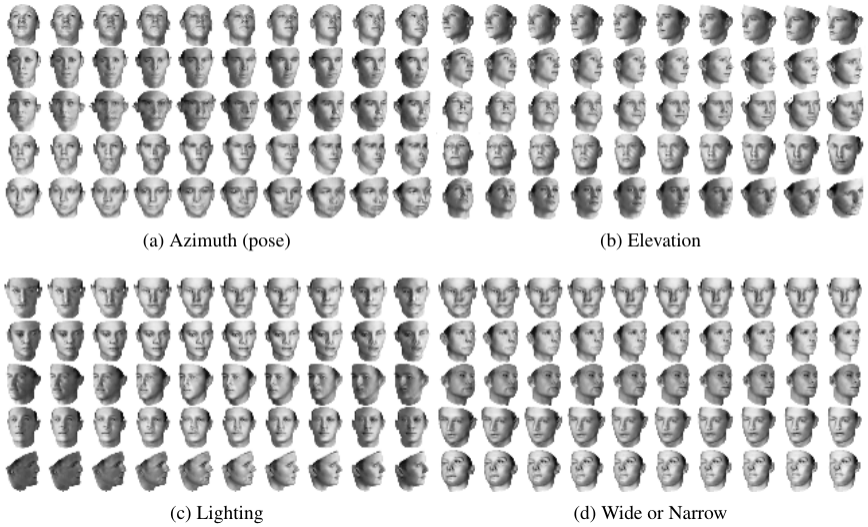


Figure 13.15: The effect of manipulating latent code in InfoGANs.



segmented images to real images, or sketches of bags to rendered "colored" bags as seen in figure 13.12.

Another example is image translation with Cycle GAN, which translates images to other domains while keeping the background fixed [Zhu et al.(2017)Zhu, Park, Isola, and Efros] as seen in figure 13.13, that translates Zebras to horses or changing the season of a landscape photo from winter to summer.

13.8.2 Experimental Simulation

GANs have also been used to generate experimental results on simulation examples in particle Physics. 13.14 shows some physical quantity of angles vs. the transformed version of the particle velocity in an AI based simulator for particle physics [de Oliveira et al.(2017)de Oliveira, Paganini, and Nachman].

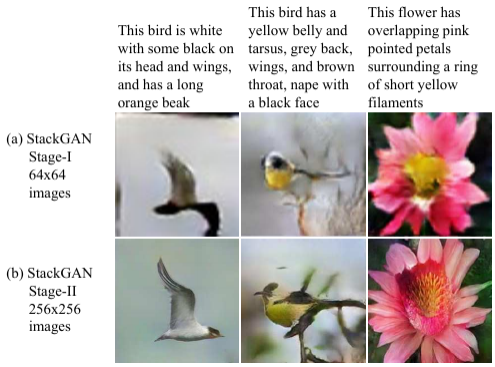
Figure 13.16: 8 Bars of music generated by MIDINet.

(a) MidiNet model 1

(b) MidiNet model 2

(c) MidiNet model 3

Figure 13.17: Some examples of text to image synthesis using StackGAN.



13.8.3 Interpretable Representations

Before, we talked about VAEs generating faces with different poses by manipulating its latent variables. Essentially, we can do the same thing with GANs, except that GANs can generate sharper images due to having an improved loss function and have more control over the latent variables that can be tweaked to generate faces of different poses with different lighting, different elevation, or wide/narrow faces as seen in figure 13.15 [Chen et al.(2016)Chen, Duan, Houthooft, Schulman, Sutskever, and Abbeel].

13.8.4 Music Synthesis

GANs have also been used to synthesis music. As seen in figure 13.16, the GAN is trying to generate notes from symbolic representations of music, opening up path to tapping into a large class of musical creativity [Yang et al.(2017)Yang, Chou, and Yang].

13.8.5 Text to Image Synthesis

GANs have also been used to generate iamges from very simple text descriptions, as seen in figure 13.17 using StackGAN [Zhang et al.(2017)Zhang, Xu, Li, Zhang, Wang, Huang, and Metaxas].

One can help but notice that the publications presented are from years 2016 to 2017, therefore quite outdated in Machine Learning. The students are encouraged to look up some of the more recent applications of GANs, like videos, drug and molecule discovery and even fighting against COVID-19.

References

[Arjovsky et al.(2017)Arjovsky, Chintala, and Bottou] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[Chen et al.(2016)Chen, Duan, Houthooft, Schulman, Sutskever, and Abbeel] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[de Oliveira et al.(2017)de Oliveira, Paganini, and Nachman] Luke de Oliveira, Michela Paganini, and Benjamin Nachman. Learning particle physics by example: location-aware generative adversarial networks for physics synthesis. *Computing and Software for Big Science*, 1(1):4, 2017.

[Gulrajani et al.(2017)Gulrajani, Ahmed, Arjovsky, Dumoulin, and Courville] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[Isola et al.(2017)Isola, Zhu, Zhou, and Efros] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[Yang et al.(2017)Yang, Chou, and Yang] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*, 2017.

[Zhang et al.(2017)Zhang, Xu, Li, Zhang, Wang, Huang, and Metaxas] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 5907–5915, 2017.

[Zhu et al.(2017)Zhu, Park, Isola, and Efros] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.