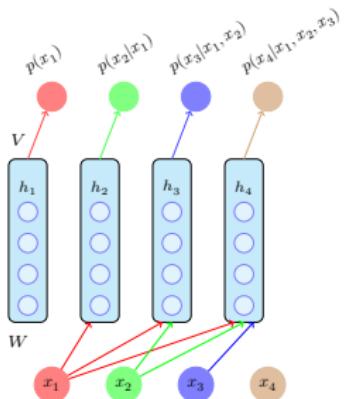
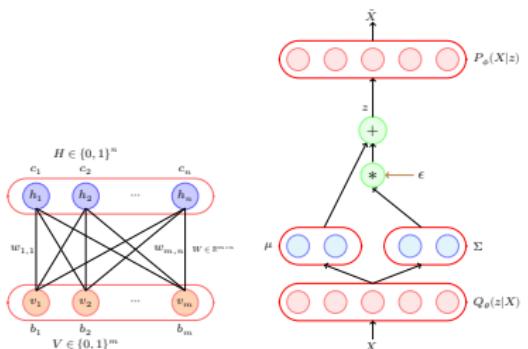


Generative Adversarial Networks (GANs)

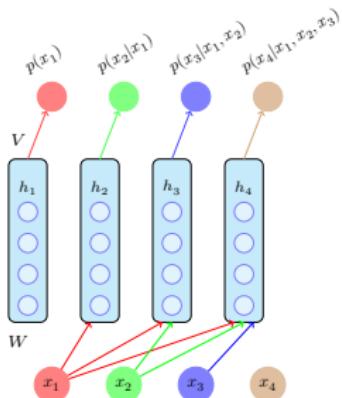
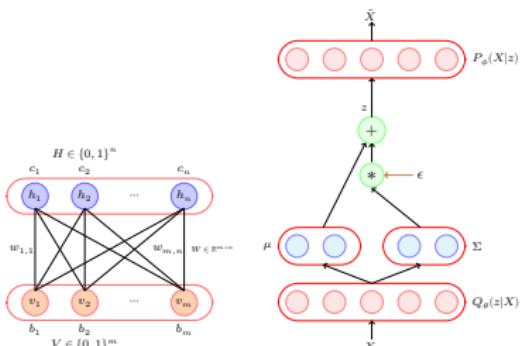
ML Instruction Team, Fall 2022

CE Department
Sharif University of Technology

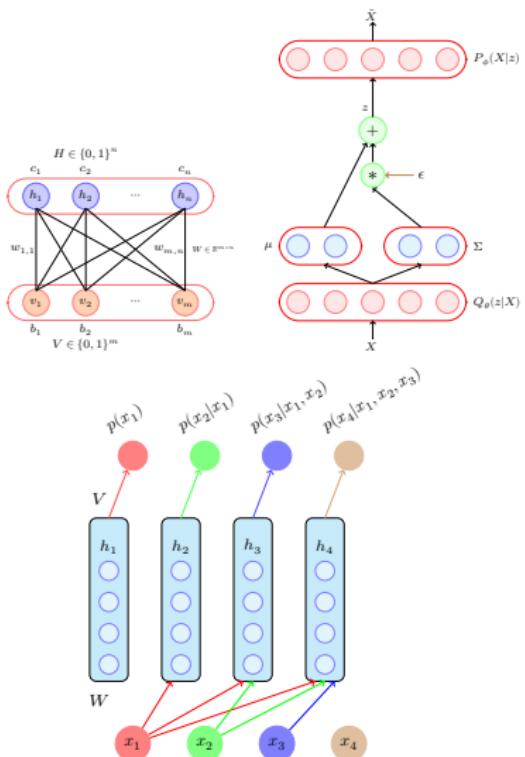
Generative Adversarial Networks - The intuition



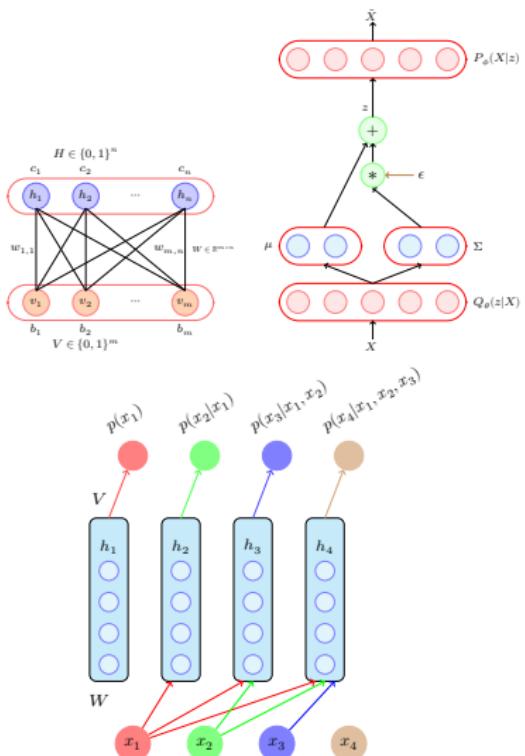
- So far we have looked at generative models which explicitly model the joint probability distribution or conditional probability distribution



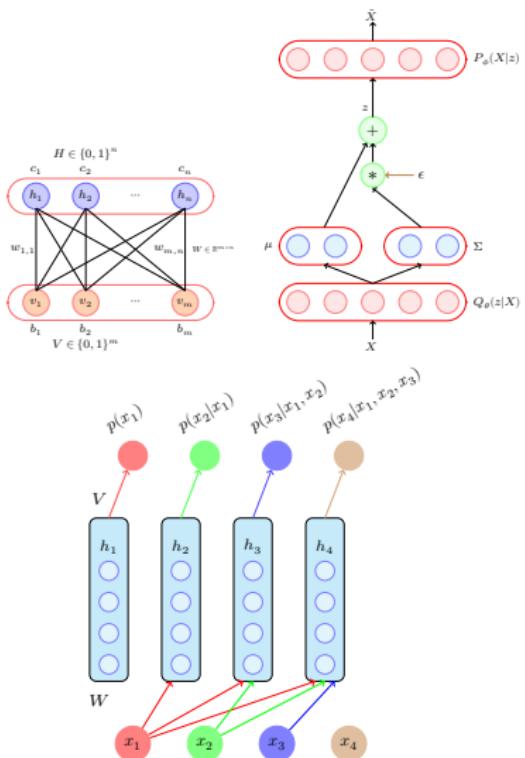
- So far we have looked at generative models which explicitly model the joint probability distribution or conditional probability distribution
- For example, in RBMs we learn $P(X, H)$, in VAEs we learn $P(z|X)$ and $P(X|z)$ whereas in AR models we learn $P(X)$



- So far we have looked at generative models which explicitly model the joint probability distribution or conditional probability distribution
- For example, in RBMs we learn $P(X, H)$, in VAEs we learn $P(z|X)$ and $P(X|z)$ whereas in AR models we learn $P(X)$
- What if we are only interested in sampling from the distribution and don't really care about explicit density function $P(X)$?



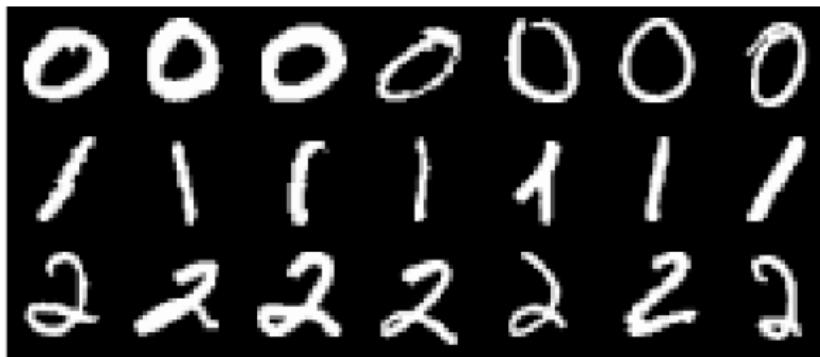
- So far we have looked at generative models which explicitly model the joint probability distribution or conditional probability distribution
- For example, in RBMs we learn $P(X, H)$, in VAEs we learn $P(z|X)$ and $P(X|z)$ whereas in AR models we learn $P(X)$
- What if we are only interested in sampling from the distribution and don't really care about explicit density function $P(X)$?
- What does this mean?



- So far we have looked at generative models which explicitly model the joint probability distribution or conditional probability distribution
- For example, in RBMs we learn $P(X, H)$, in VAEs we learn $P(z|X)$ and $P(X|z)$ whereas in AR models we learn $P(X)$
- What if we are only interested in sampling from the distribution and don't really care about explicit density function $P(X)$?
- What does this mean? Let us see



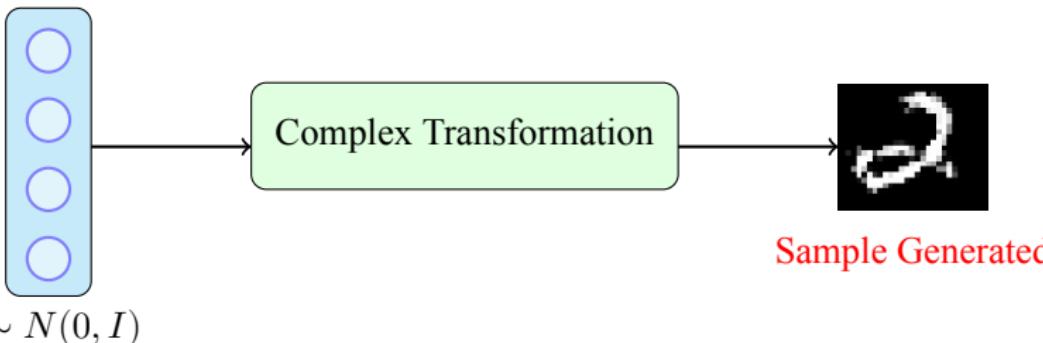
- As usual we are given some training data (say, MNIST images) which obviously comes from some underlying distribution



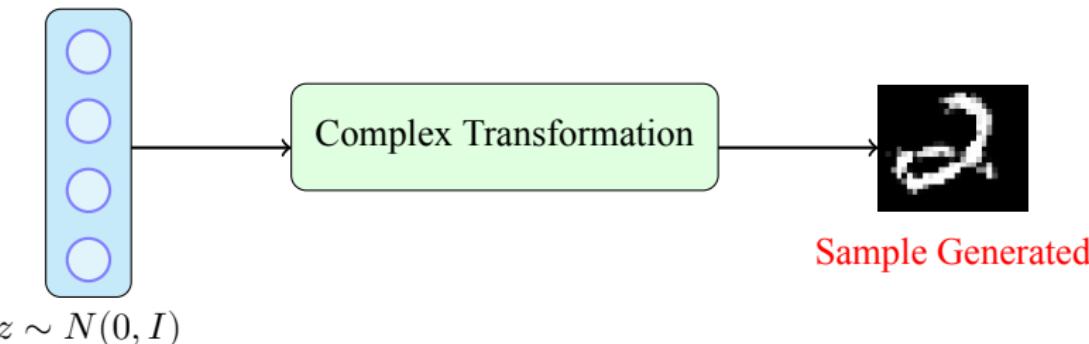
- As usual we are given some training data (say, MNIST images) which obviously comes from some underlying distribution
- Our goal is to generate more images from this distribution (*i.e.*, create images which look similar to the images from the training data)



- As usual we are given some training data (say, MNIST images) which obviously comes from some underlying distribution
- Our goal is to generate more images from this distribution (*i.e.*, create images which look similar to the images from the training data)
- In other words, we want to sample from a complex high dimensional distribution which is intractable (recall RBMs, VAEs and AR models deal with this intractability in their own way)



- GANs take a different approach to this problem where the idea is to sample from a simple tractable distribution (say, $z \sim N(0, I)$) and then learn a complex transformation from this to the training distribution



$$z \sim N(0, I)$$

Sample Generated

- GANs take a different approach to this problem where the idea is to sample from a simple tractable distribution (say, $z \sim N(0, I)$) and then learn a complex transformation from this to the training distribution
- In other words, we will take a $z \sim N(0, I)$, learn to make a series of complex transformations on it so that the output looks as if it came from our training distribution

- What can we use for such a complex transformation?

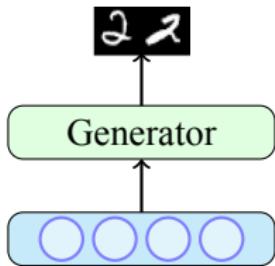
- What can we use for such a complex transformation? **A Neural Network**

- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network?

- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network?
Using a two player game

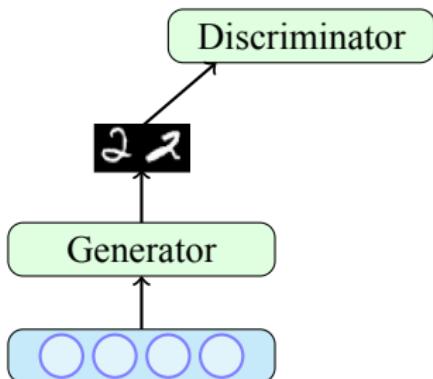
- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network?
Using a two player game
- There are two players in the game:

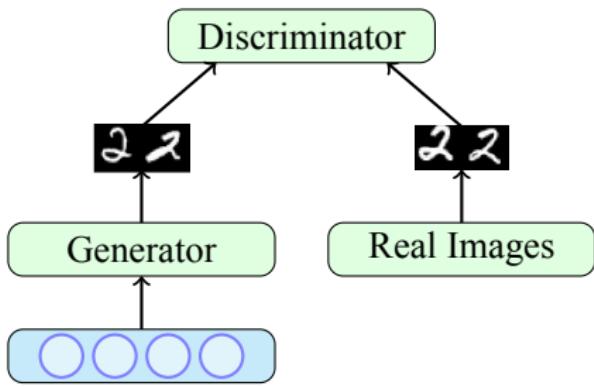
- What can we use for such a complex transformation? A Neural Network
 - How do you train such a neural network? Using a two player game
 - There are two players in the game: a generator



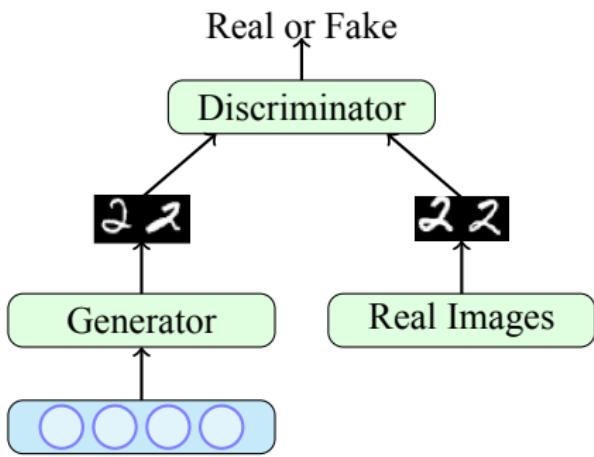
$$z \sim N(0, I)$$

- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network? Using a two player game
- There are two players in the game: a generator and a **discriminator**



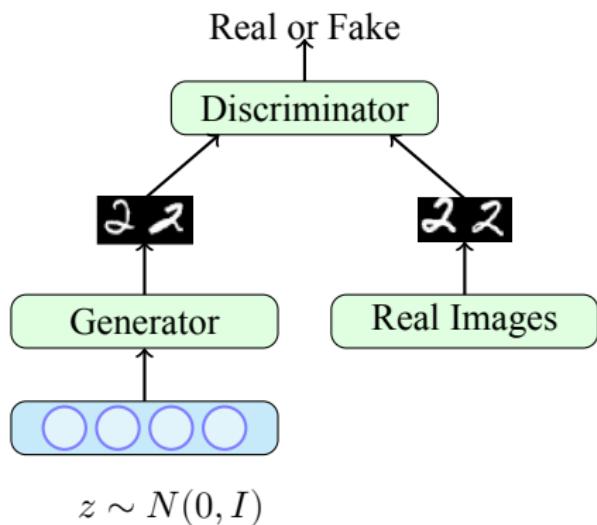


- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network? Using a two player game
- There are two players in the game: a generator and a discriminator
- The job of the generator is to produce images which look so natural that the discriminator thinks that the images came from the real data distribution

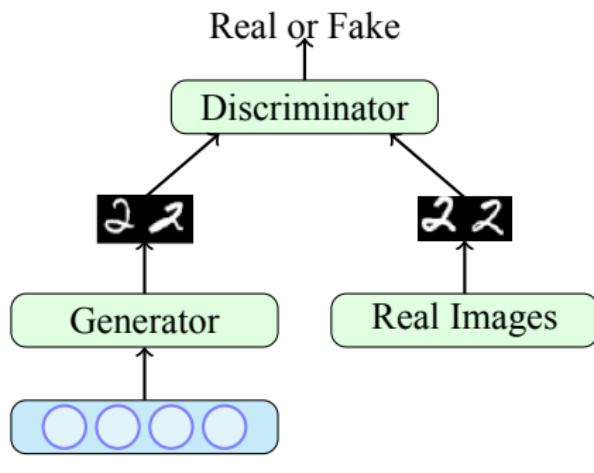


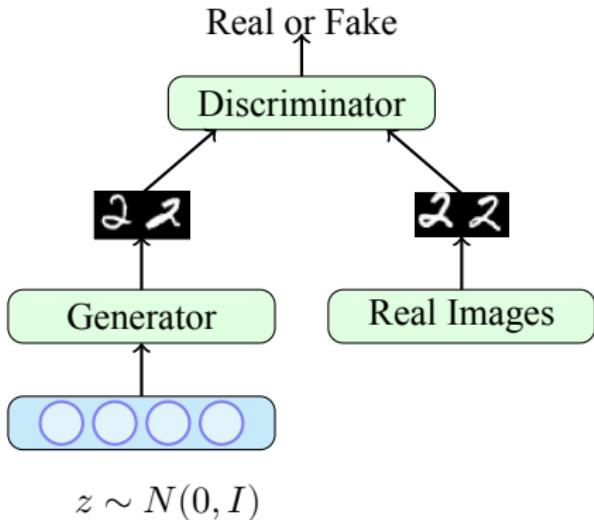
- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network? Using a two player game
- There are two players in the game: a generator and a discriminator
- The job of the generator is to produce images which look so natural that the discriminator thinks that the images came from the real data distribution
- The job of the discriminator is to get better and better at distinguishing between true images and generated (fake) images

So let's look at the full picture

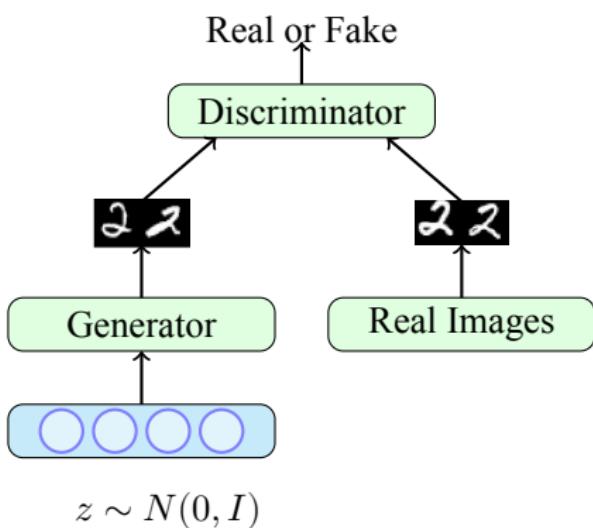


- So let's look at the full picture
- Let G_ϕ be the generator and D_θ be the discriminator (ϕ and θ are the parameters of G and D , respectively)



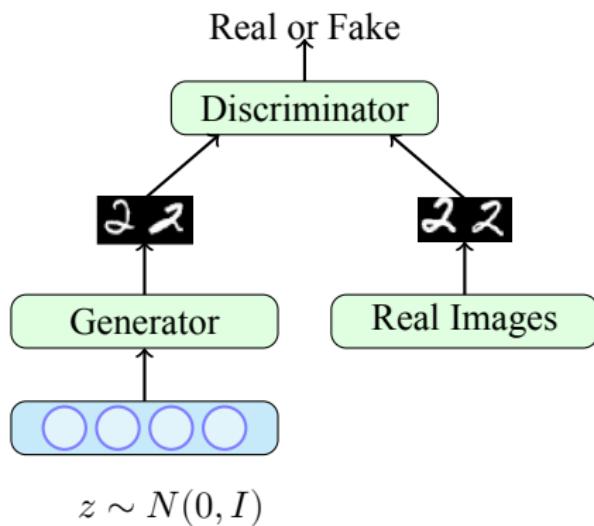


- So let's look at the full picture
- Let G_ϕ be the generator and D_θ be the discriminator (ϕ and θ are the parameters of G and D , respectively)
- We have a neural network based generator which takes as input a noise vector $z \sim N(0, I)$ and produces $G_\phi(z) = X$

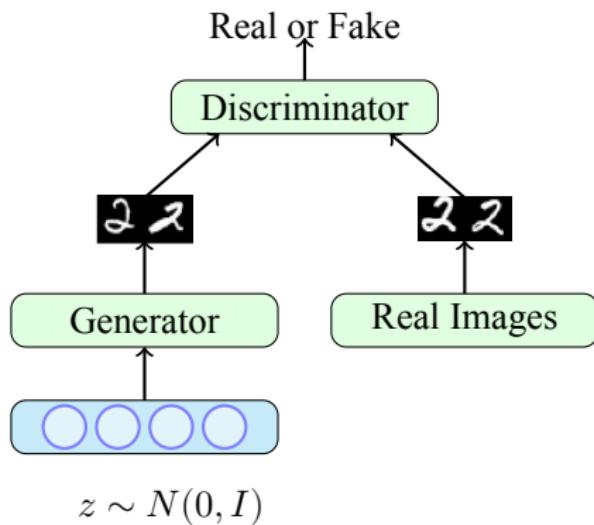


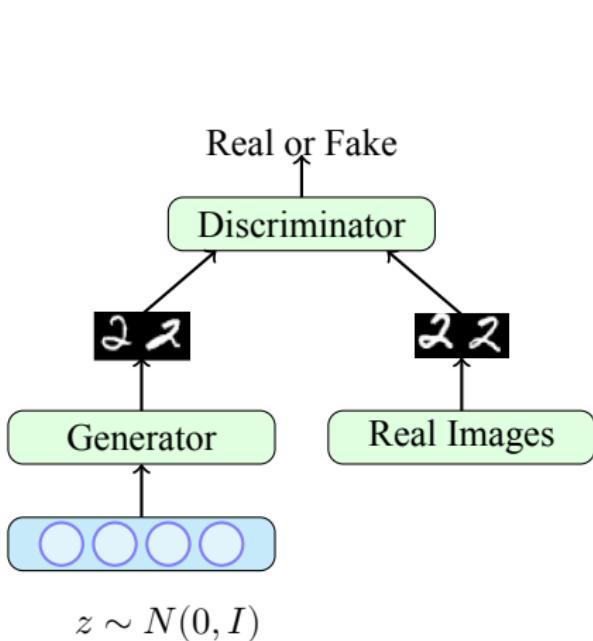
- So let's look at the full picture
- Let G_ϕ be the generator and D_θ be the discriminator (ϕ and θ are the parameters of G and D , respectively)
- We have a neural network based generator which takes as input a noise vector $z \sim N(0, I)$ and produces $G_\phi(z) = X$
- We have a neural network based discriminator which could take as input a real X or a generated $X = G_\phi(z)$ and classify the input as real/fake

- What should be the objective function of the overall network?

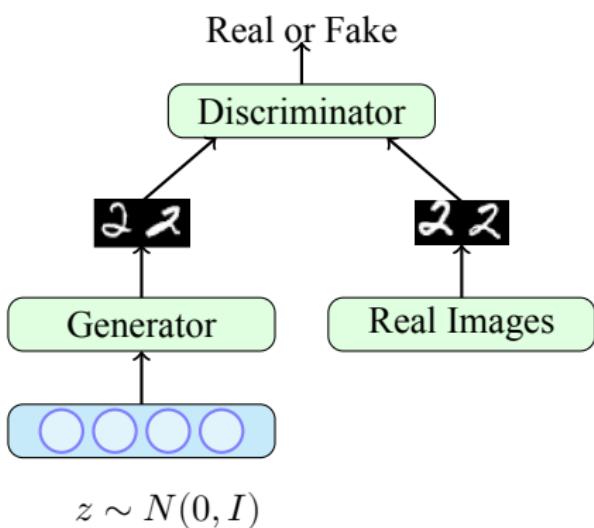


- What should be the objective function of the overall network?
- Let's look at the objective function of the generator first

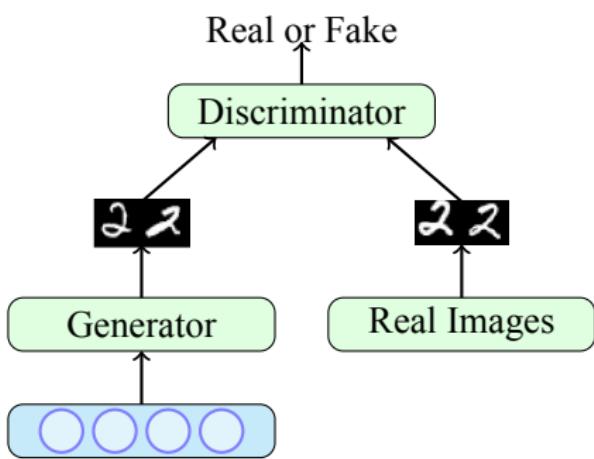




- What should be the objective function of the overall network?
- Let's look at the objective function of the generator first
- Given an image generated by the generator as $G_\phi(z)$ the discriminator assigns a score $D_\theta(G_\phi(z))$ to it

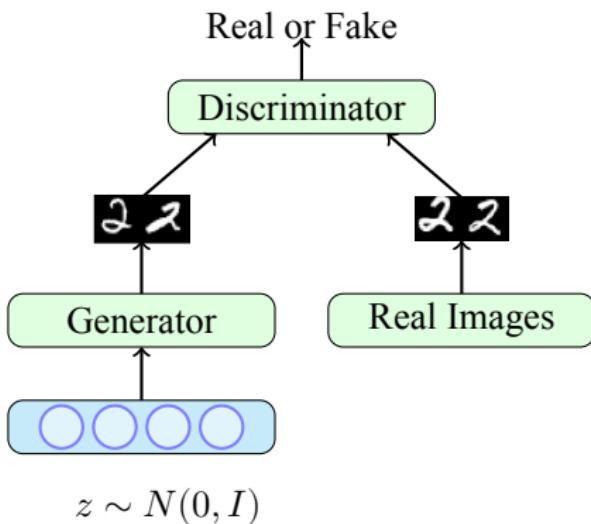


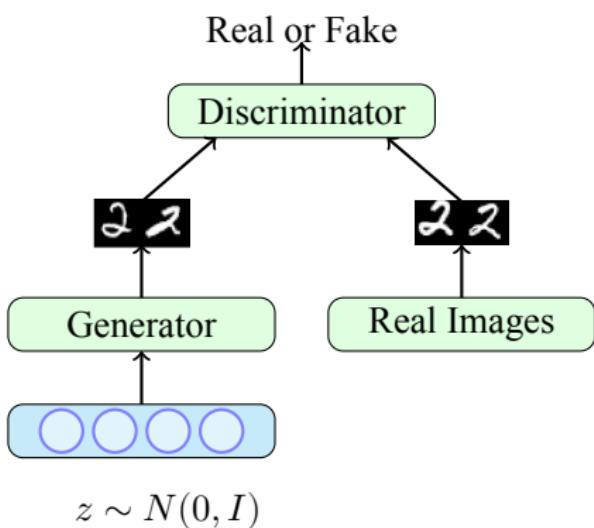
- What should be the objective function of the overall network?
- Let's look at the objective function of the generator first
- Given an image generated by the generator as $G_\phi(z)$ the discriminator assigns a score $D_\theta(G_\phi(z))$ to it
- This score will be between 0 and 1 and will tell us the probability of the image being real or fake



- What should be the objective function of the overall network?
- Let's look at the objective function of the generator first
- Given an image generated by the generator as $G_\phi(z)$ the discriminator assigns a score $D_\theta(G_\phi(z))$ to it
- This score will be between 0 and 1 and will tell us the probability of the image being real or fake
- For a given z , the generator would want to maximize $\log D_\theta(G_\phi(z))$ (log likelihood) or minimize $\log(1 - D_\theta(G_\phi(z)))$

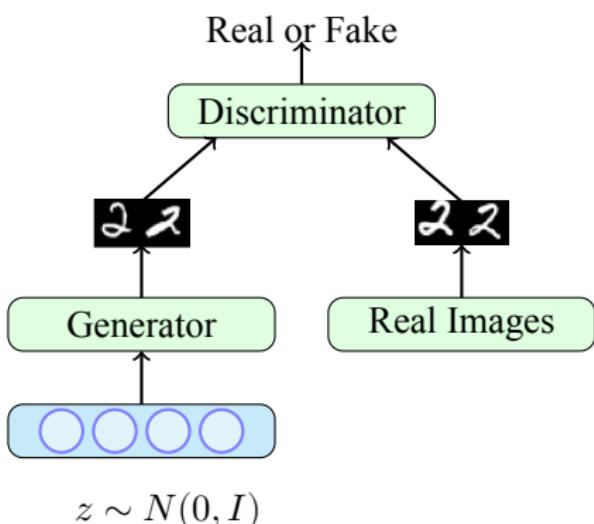
- This is just for a single z and the generator would like to do this for all possible values of z ,





- This is just for a single z and the generator would like to do this for all possible values of z ,
- For example, if z was discrete and drawn from a uniform distribution (*i.e.*, $p(z) = \frac{1}{N} \forall z$) then the generator's objective function would be

$$\min_{\phi} \sum_{i=1}^N \frac{1}{N} \log(1 - D_{\theta}(G_{\phi}(z)))$$



- This is just for a single z and the generator would like to do this for all possible values of z ,
- For example, if z was discrete and drawn from a uniform distribution (*i.e.*, $p(z) = \frac{1}{N} \forall z$) then the generator's objective function would be

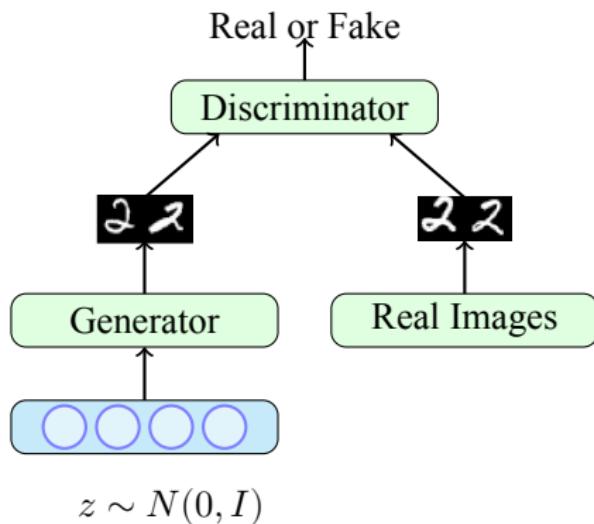
$$\min_{\phi} \sum_{i=1}^N \frac{1}{N} \log(1 - D_{\theta}(G_{\phi}(z)))$$

- However, in our case, z is continuous and not uniform ($z \sim N(0, I)$) so the equivalent objective function would be

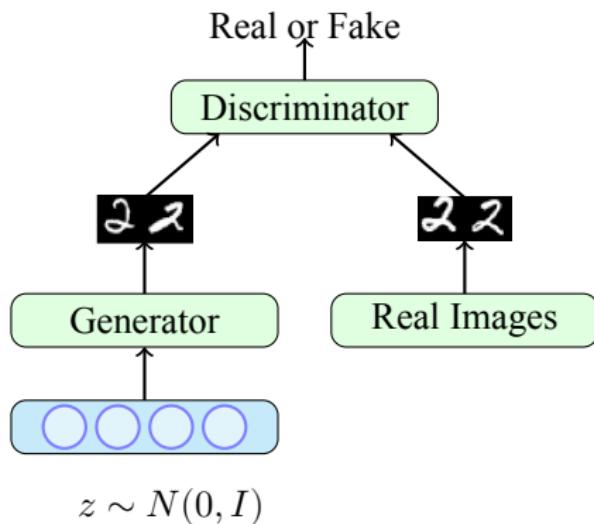
$$\min_{\phi} \int p(z) \log(1 - D_{\theta}(G_{\phi}(z)))$$

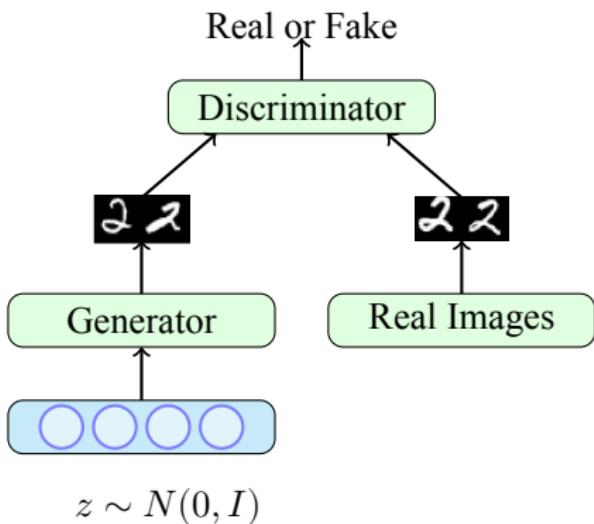
$$\min_{\phi} E_{z \sim p(z)} [\log(1 - D_{\theta}(G_{\phi}(z)))]$$

Now let's look at the discriminator

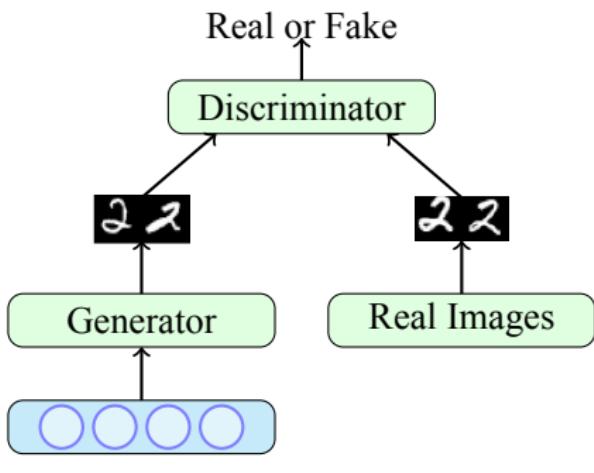


- Now let's look at the discriminator
- The task of the discriminator is to assign a high score to real images and a low score to fake images





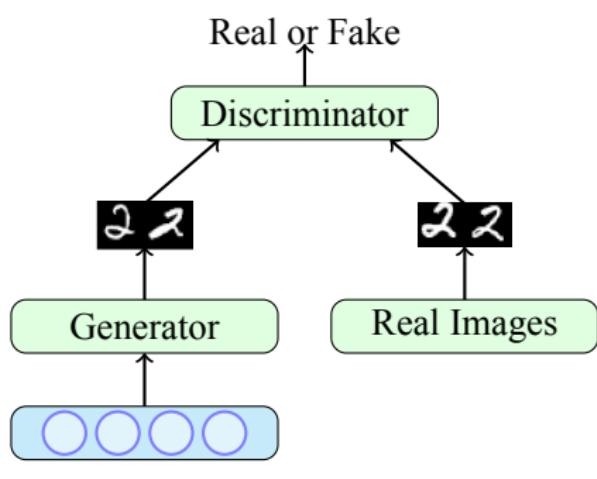
- Now let's look at the discriminator
- The task of the discriminator is to assign a high score to real images and a low score to fake images
- And it should do this for all possible real images and all possible fake images



- Now let's look at the discriminator
- The task of the discriminator is to assign a high score to real images and a low score to fake images
- And it should do this for all possible real images and all possible fake images
- In other words, it should try to maximize the following objective function

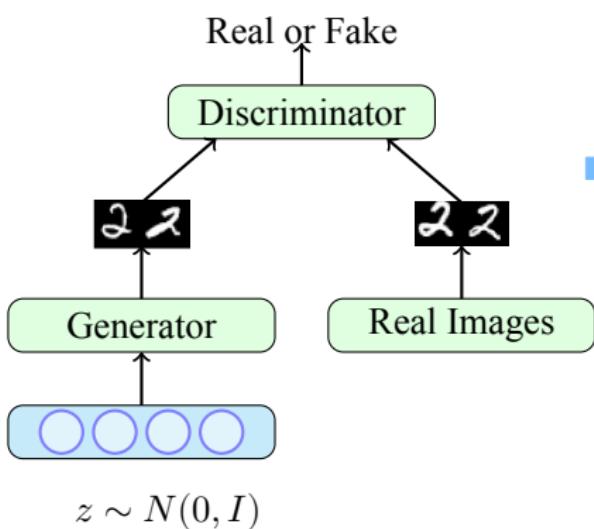
$$\max_{\theta} E_{x \sim p_{data}} [\log D_{\theta}(x)] + E_{z \sim p(z)} [\log(1 - D_{\theta}(G(z)))]$$

- If we put the objectives of the generator and discriminator together we get a minimax game



$$\min_{\phi} \max_{\theta} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))]$$

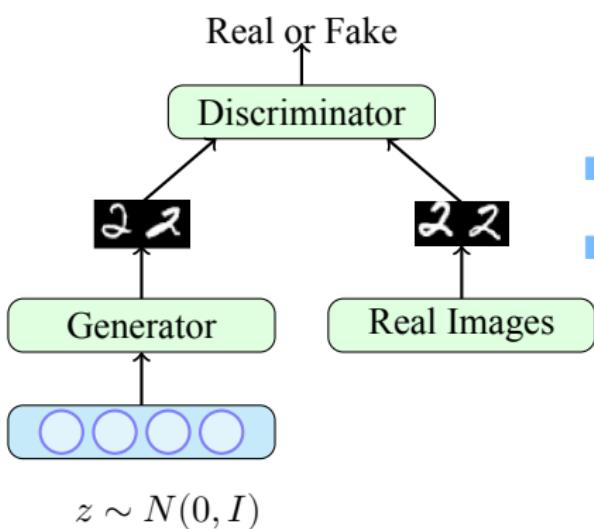
- If we put the objectives of the generator and discriminator together we get a minimax game



$$\min_{\phi} \max_{\theta} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))]$$

- The first term in the objective is only w.r.t. the parameters of the discriminator (θ)

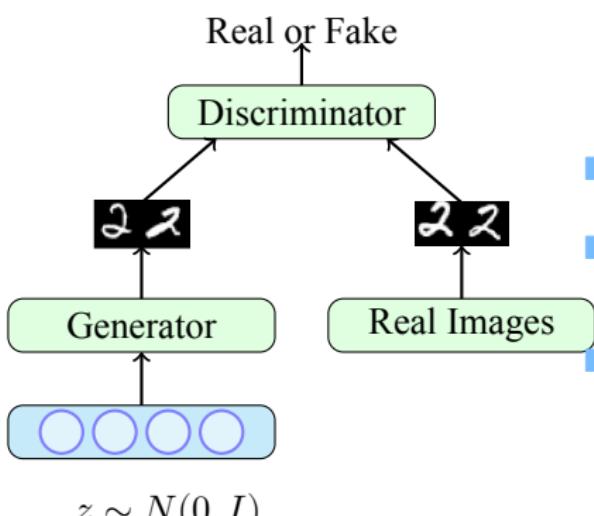
- If we put the objectives of the generator and discriminator together we get a minimax game



$$\min_{\phi} \max_{\theta} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))]$$

- The first term in the objective is only w.r.t. the parameters of the discriminator (θ)
- The second term in the objective is w.r.t. the parameters of the generator (ϕ) as well as the discriminator (θ)

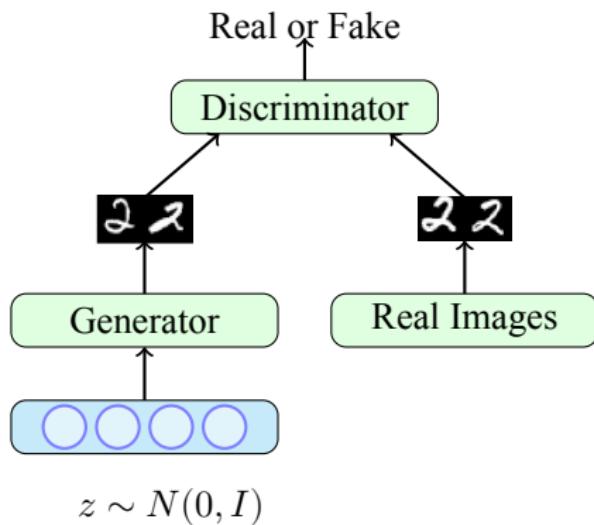
- If we put the objectives of the generator and discriminator together we get a minimax game



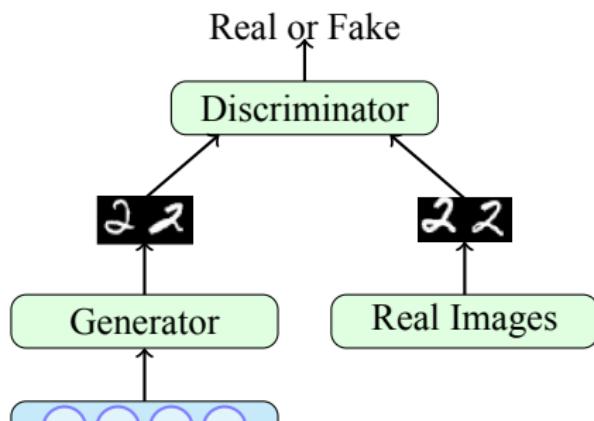
$$\min_{\phi} \max_{\theta} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))]$$

- The first term in the objective is only w.r.t. the parameters of the discriminator (θ)
- The second term in the objective is w.r.t. the parameters of the generator (ϕ) as well as the discriminator (θ)
- The discriminator wants to maximize the second term whereas the generator wants to minimize it (hence it is a two-player game)

- So the overall training proceeds by alternating between these two step

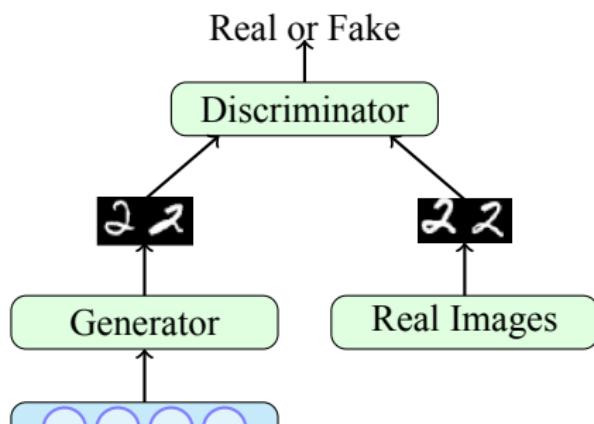


- So the overall training proceeds by alternating between these two step
- Step 1:** Gradient Ascent on Discriminator



$$\max_{\theta} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G(z)))]$$

- So the overall training proceeds by alternating between these two step
- Step 1:** Gradient Ascent on Discriminator

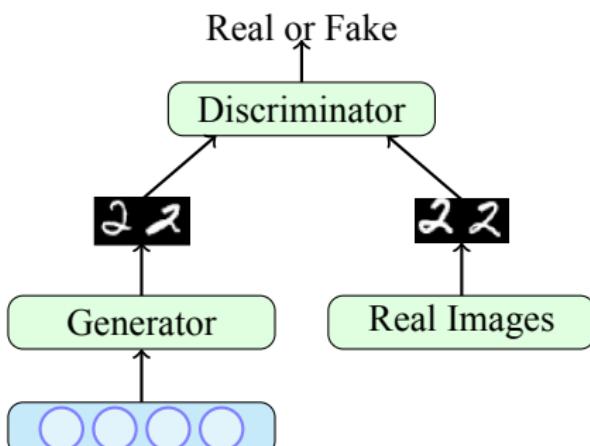


$$\max_{\theta} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))]$$

- Step 2:** Gradient Descent on Generator

$$\min_{\phi} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))$$

- So the overall training proceeds by alternating between these two step
- Step 1:** Gradient Ascent on Discriminator



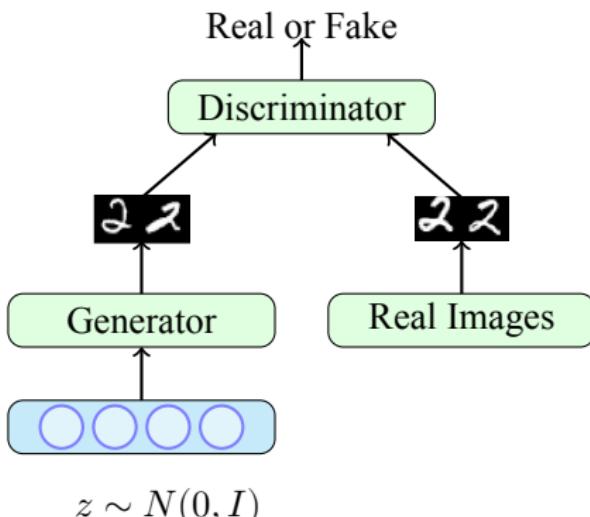
$$\max_{\theta} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))]$$

- Step 2:** Gradient Descent on Generator

$$\min_{\phi} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))$$

- In practice, the above generator objective does not work well and we use a slightly modified objective

- So the overall training proceeds by alternating between these two step
- Step 1:** Gradient Ascent on Discriminator



$$\max_{\theta} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))]$$

- Step 2:** Gradient Descent on Generator

$$\min_{\phi} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))$$

- In practice, the above generator objective does not work well and we use a slightly modified objective
- Let us see why

Generative Adversarial Networks - Architecture

- We will now look at one of the popular neural networks used for the generator and discriminator (Deep Convolutional GANs) For discriminator, any CNN based classifier with 1 class (real) at the output can be used (e.g. VGG, ResNet, etc.)

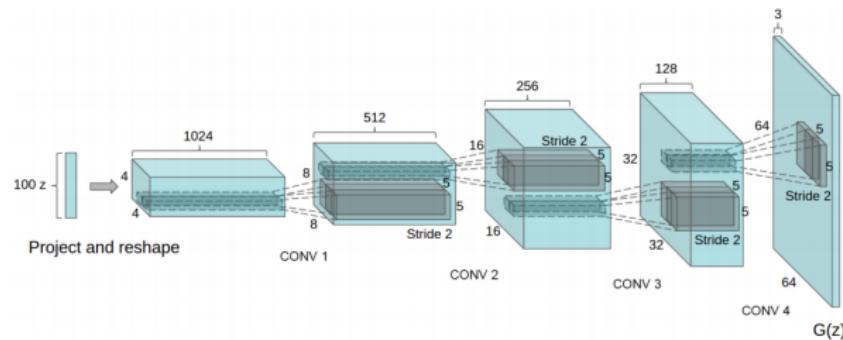


Figure: Generator (Redford et al 2015) (left) and discriminator (Yeh et al 2016) (right) used in DCGAN

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses tanh.
- Use LeakyReLU activation in the discriminator for all layers

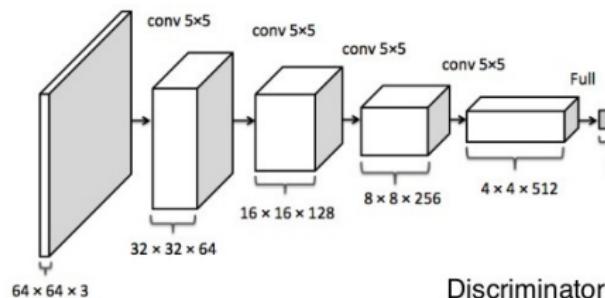
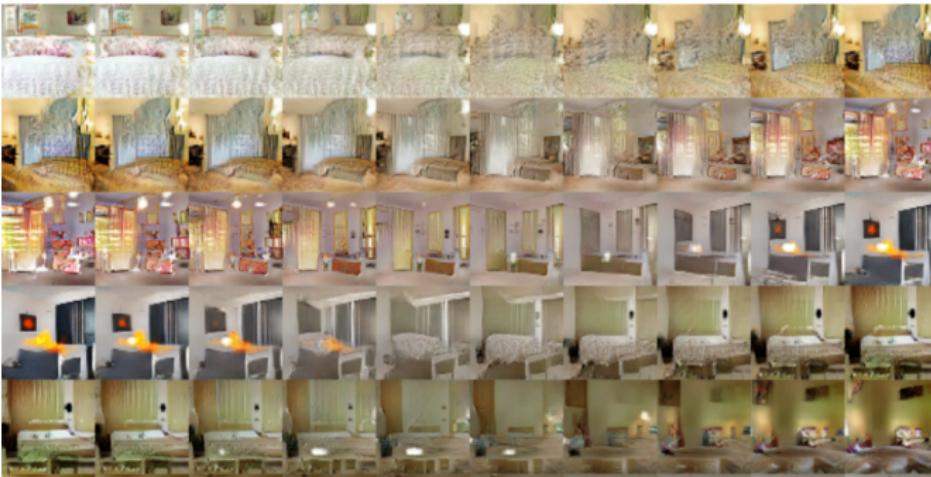


Figure: Generator (Redford et al 2015) (left) and discriminator (Yeh et al 2016) (right) used in DCGAN

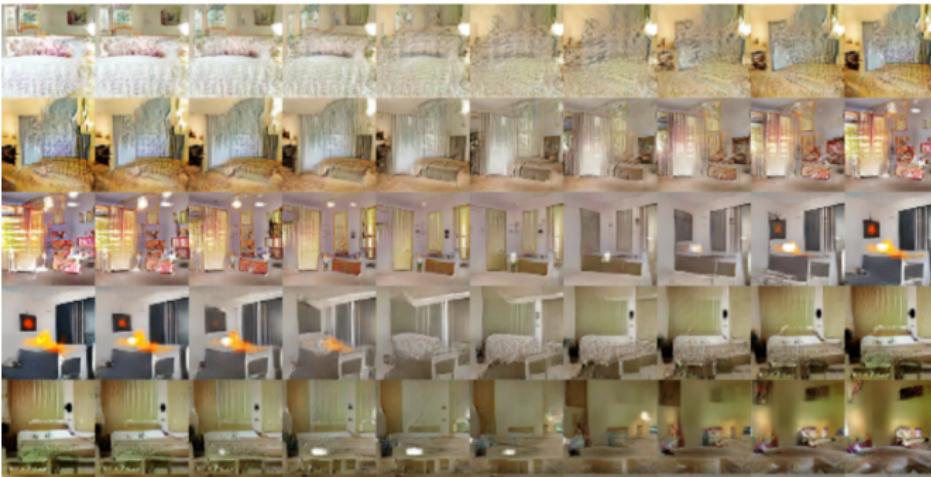
Generative Adversarial Networks - Some Cool Stuff and Applications



- In each row the first image was generated by the network by taking a vector z_1 as the input and the last images was generated by a vector z_2 as the input



- In each row the first image was generated by the network by taking a vector z_1 as the input and the last images were generated by a vector z_2 as the input
- All intermediate images were generated by feeding z 's which were obtained by interpolating z_1 and z_2 ($z = \lambda z_1 + (1 - \lambda)z_2$)



- In each row the first image was generated by the network by taking a vector z_1 as the input and the last images were generated by a vector z_2 as the input
- All intermediate images were generated by feeding z 's which were obtained by interpolating z_1 and z_2 ($z = \lambda z_1 + (1 - \lambda)z_2$)
- As we transition from z_1 to z_2 in the input space there is a corresponding smooth transition in the image space also

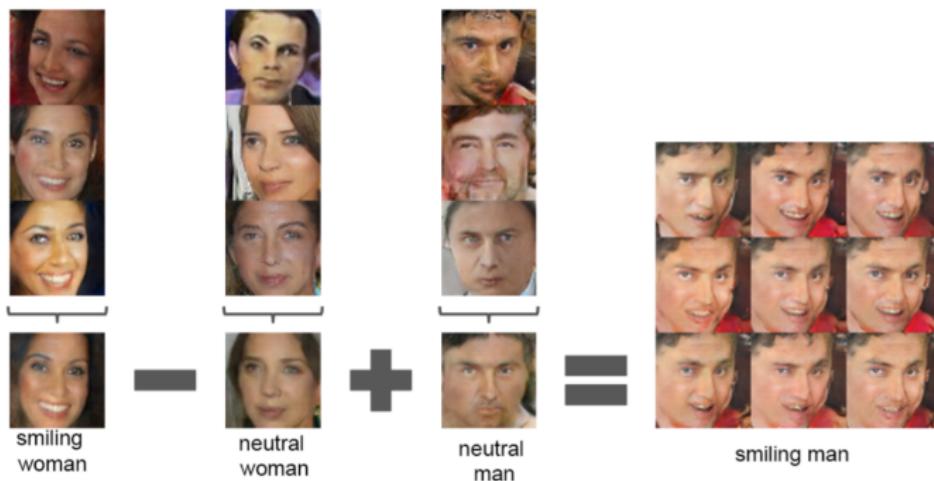


Figure: Vector arithmetic for visual concepts (part of figure 7 from the DCGAN paper)

- The first 3 images in the first column were generated by feeding some z_{11}, z_{12}, z_{13} respectively as the input to the generator

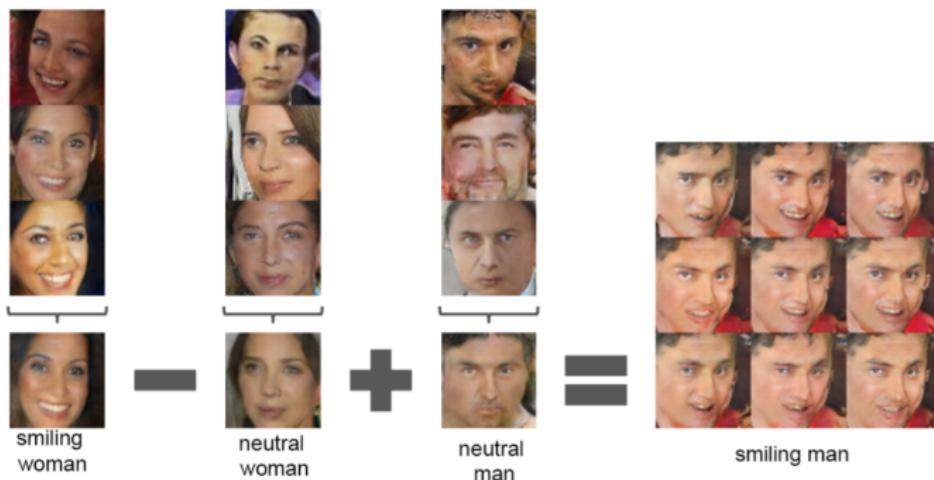


Figure: Vector arithmetic for visual concepts (part of figure 7 from the DCGAN paper)

- The first 3 images in the first column were generated by feeding some z_{11}, z_{12}, z_{13} respectively as the input to the generator
- The fourth image was generated by taking an average of $z_1 = z_{11}, z_{12}, z_{13}$ and feeding it to the generator

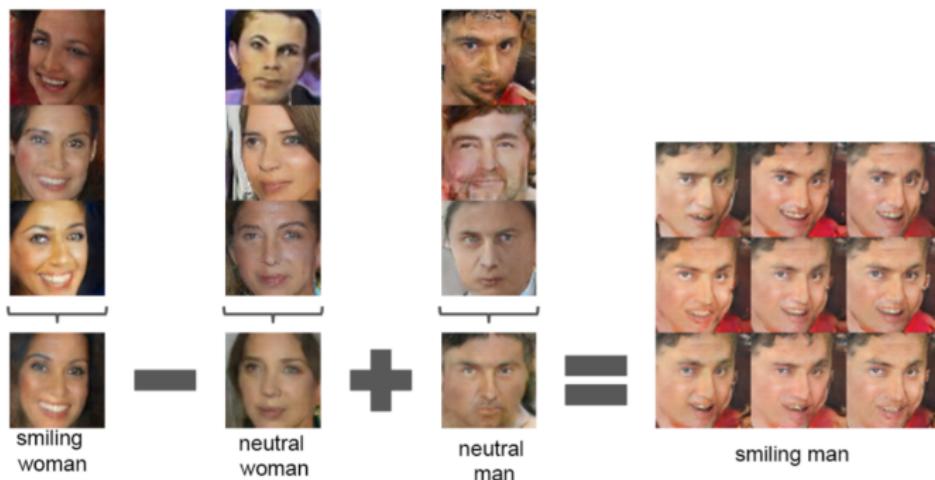


Figure: Vector arithmetic for visual concepts (part of figure 7 from the DCGAN paper)

- The first 3 images in the first column were generated by feeding some z_{11}, z_{12}, z_{13} respectively as the input to the generator
- The fourth image was generated by taking an average of $z_1 = z_{11}, z_{12}, z_{13}$ and feeding it to the generator
- Similarly we obtain the average vectors z_2 and z_3 for the 2nd and 3rd columns

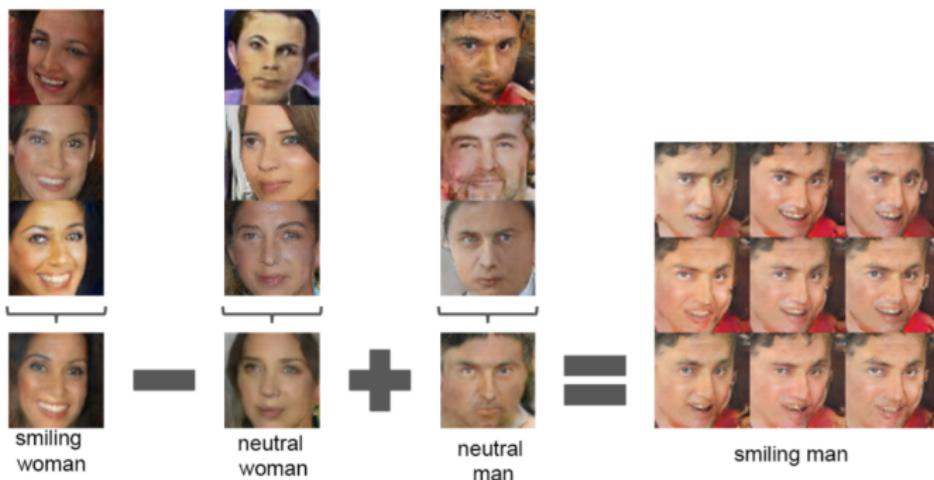


Figure: Vector arithmetic for visual concepts (part of figure 7 from the DCGAN paper)

- The first 3 images in the first column were generated by feeding some z_{11}, z_{12}, z_{13} respectively as the input to the generator
- The fourth image was generated by taking an average of $z_1 = z_{11}, z_{12}, z_{13}$ and feeding it to the generator
- Similarly we obtain the average vectors z_2 and z_3 for the 2nd and 3rd columns
- If we do a simple vector arithmetic on these averaged vectors then we see the corresponding effect in the generated images

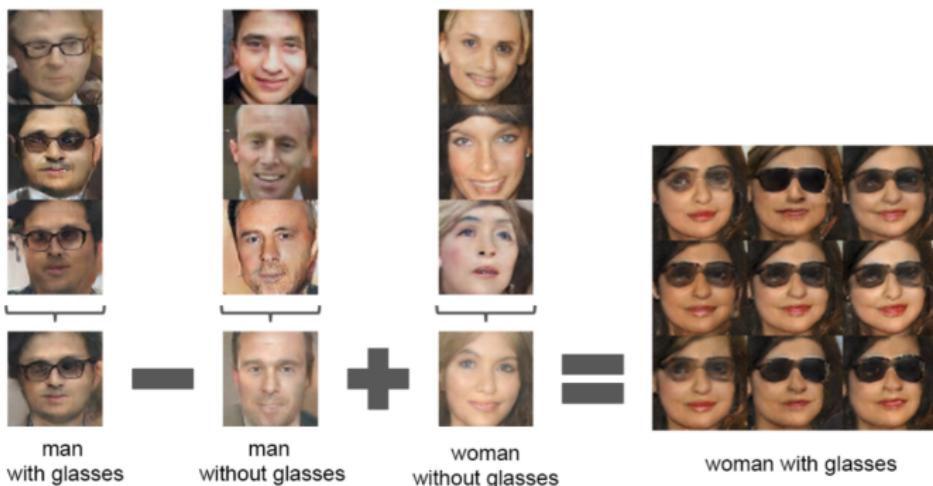
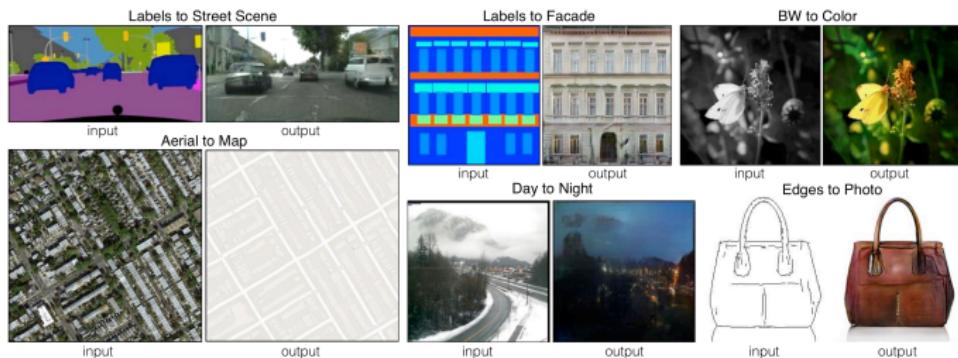


Figure: Vector arithmetic for visual concepts (part of figure 7 from the DCGAN paper)

- The first 3 images in the first column were generated by feeding some z_{11}, z_{12}, z_{13} respectively as the input to the generator
- The fourth image was generated by taking an average of $z_1 = z_{11}, z_{12}, z_{13}$ and feeding it to the generator
- Similarly we obtain the average vectors z_2 and z_3 for the 2nd and 3rd columns
- If we do a simple vector arithmetic on these averaged vectors then we see the corresponding effect in the generated images



[Phillip Isola](#), [Jun-Yan Zhu](#), [Tinghui Zhou](#), [Alexei A. Efros](#), Image-to-Image Translation with Conditional Adversarial Networks, CVPR, 2017.

Module 23.1: Generative Adversarial Networks - StyleGAN + CycleGAN



Figure: Example of One Set of Generated Faces (Left) Adopting the Coarse Style of Another Set of Generated Faces (Top) Taken from: A Style-Based Generator Architecture for Generative Adversarial Networks.

■ GAN: Lacking Control Over Synthesized Images



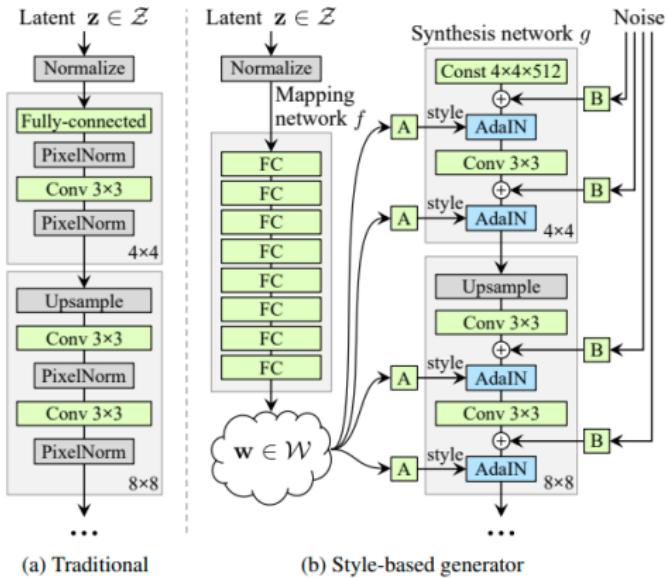
Figure: Example of One Set of Generated Faces (Left) Adopting the Coarse Style of Another Set of Generated Faces (Top) Taken from: A Style-Based Generator Architecture for Generative Adversarial Networks.

- GAN: Lacking Control Over Synthesized Images
- Style Generative Adversarial Network(StyleGAN) Controls Style Using New Generator Model



Figure: Example of One Set of Generated Faces (Left) Adopting the Coarse Style of Another Set of Generated Faces (Top) Taken from: A Style-Based Generator Architecture for Generative Adversarial Networks.

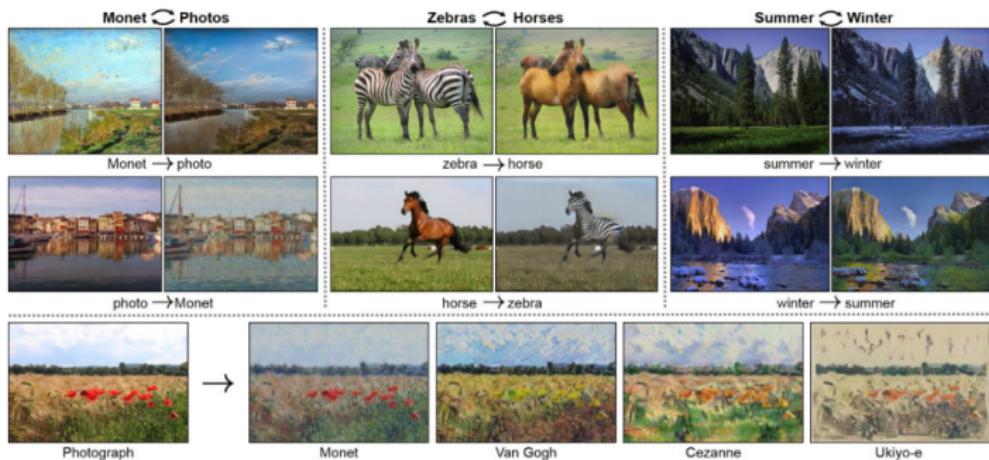
- GAN: Lacking Control Over Synthesized Images
- Style Generative Adversarial Network(StyleGAN) Controls Style Using New Generator Model
- StyleGan is proficient in producing impressively photorealistic high-quality photos of faces and grants control over the characteristic of the created image at different specification levels by changing the style vectors and noise.



The StyleGAN is described as a progressive growing GAN architecture with five modifications, each of which was added and evaluated incrementally in an ablative study. The incremental list of changes to the generator are:

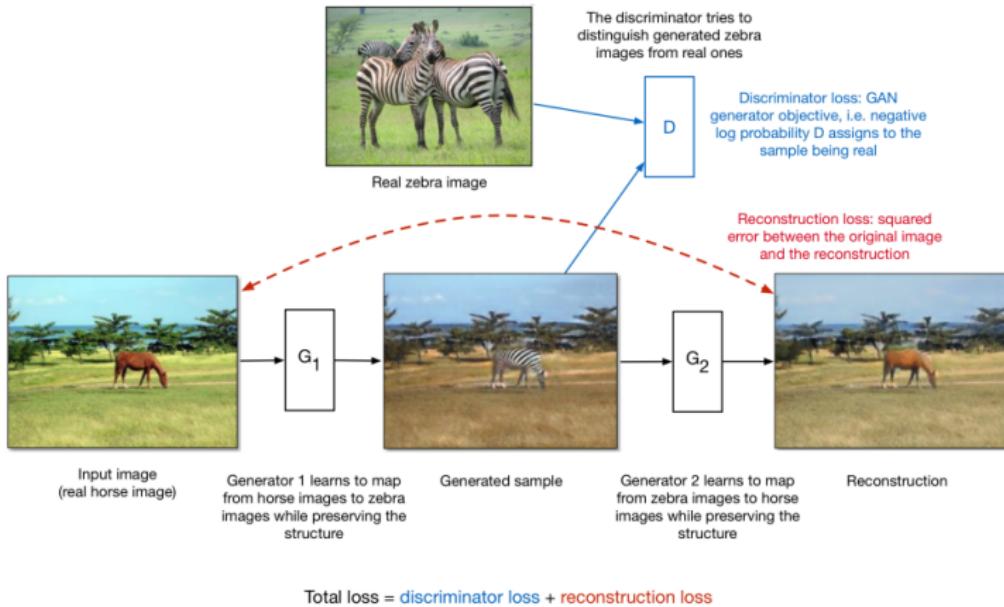
- Baseline Progressive GAN.
- Addition of tuning and bilinear upsampling.
- Addition of mapping network and AdaIN (styles).
- Removal of latent vector input to generator.
- Addition of noise to each block.
- Addition Mixing regularization.

Style transfer problem: change the style of an image while preserving the content.

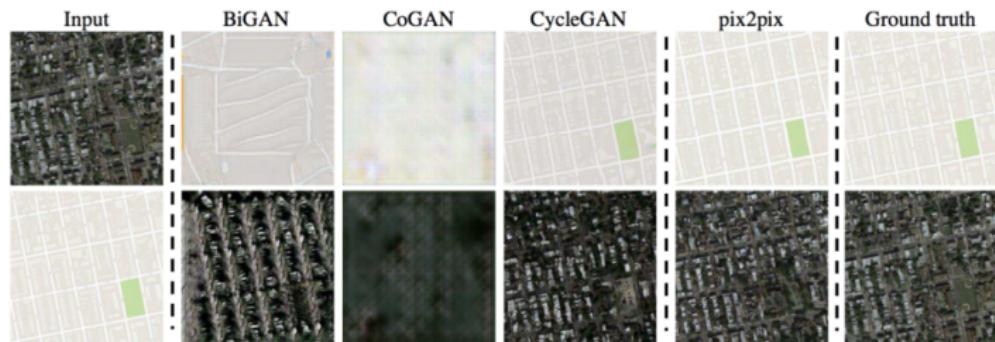


Data: Two unrelated collections of images, one for each style

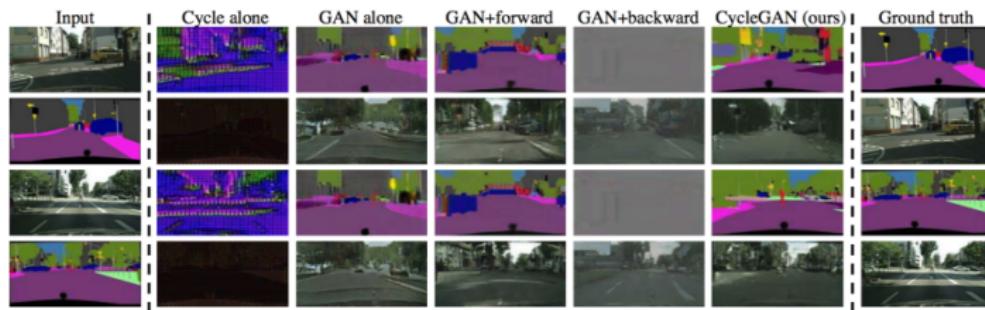
- If we had paired data (same content in both styles), this would be a supervised learning problem. But this is hard to find.
- The CycleGAN architecture learns to do it from unpaired data.
 - ▶ Train two different generator nets to go from style 1 to style 2, and vice versa.
 - ▶ Make sure the generated samples of style 2 are indistinguishable from real images by a discriminator net.
 - ▶ Make sure the generators are cycle-consistent: mapping from style 1 to style 2 and back again should give you almost the original image.



Style transfer between aerial photos and maps:



Style transfer between road scenes and semantic segmentations (labels of every pixel in an image by object category):



Generative Adversarial Networks - Summary

	RBM	VAE	AR models	GANs
Abstraction	Yes	Yes	No	No

Table: Comparison of Generative Models

	RBM	VAE	AR models	GANs
Abstraction	Yes	Yes	No	No
Generation	Yes	Yes	Yes	Yes

Table: Comparison of Generative Models

	RBM	VAE	AR models	GANs
Abstraction	Yes	Yes	No	No
Generation	Yes	Yes	Yes	Yes
Compute $P(X)$	Intractable	Intractable	Tractable	No

Table: Comparison of Generative Models

	RBM	VAE	AR models	GANs
Abstraction	Yes	Yes	No	No
Generation	Yes	Yes	Yes	Yes
Compute $P(X)$	Intractable	Intractable	Tractable	No
Sampling	MCMC	Fast	Slow	Fast

Table: Comparison of Generative Models

	RBM	VAE	AR models	GANs
Abstraction	Yes	Yes	No	No
Generation	Yes	Yes	Yes	Yes
Compute $P(X)$	Intractable	Intractable	Tractable	No
Sampling	MCMC	Fast	Slow	Fast
Type of GM	Undirected	Directed	Directed	Directed

Table: Comparison of Generative Models

	RBM	VAE	AR models	GANs
Abstraction	Yes	Yes	No	No
Generation	Yes	Yes	Yes	Yes
Compute $P(X)$	Intractable	Intractable	Tractable	No
Sampling	MCMC	Fast	Slow	Fast
Type of GM	Undirected	Directed	Directed	Directed
Loss	KL-divergence	KL-divergence	KL-divergence	JSD

Table: Comparison of Generative Models

	RBM	VAE	AR models	GANs
Abstraction	Yes	Yes	No	No
Generation	Yes	Yes	Yes	Yes
Compute $P(X)$	Intractable	Intractable	Tractable	No
Sampling	MCMC	Fast	Slow	Fast
Type of GM	Undirected	Directed	Directed	Directed
Loss	KL-divergence	KL-divergence	KL-divergence	JSD
Assumptions	X independent given z	X independent given z	None	N.A.

Table: Comparison of Generative Models

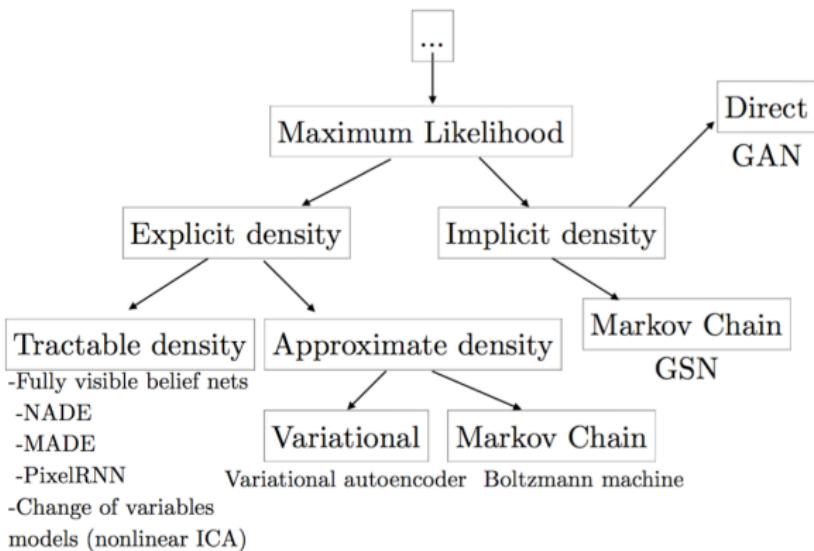
	RBM	VAE	AR models	GANs
Abstraction	Yes	Yes	No	No
Generation	Yes	Yes	Yes	Yes
Compute $P(X)$	Intractable	Intractable	Tractable	No
Sampling	MCMC	Fast	Slow	Fast
Type of GM	Undirected	Directed	Directed	Directed
Loss	KL-divergence	KL-divergence	KL-divergence	JSD
Assumptions	X independent given z	X independent given z	None	N.A.
Samples	Bad	Ok	Good	Good (best)

Table: Comparison of Generative Models

	RBM	VAE	AR models	GANs
Abstraction	Yes	Yes	No	No
Generation	Yes	Yes	Yes	Yes
Compute $P(X)$	Intractable	Intractable	Tractable	No
Sampling	MCMC	Fast	Slow	Fast
Type of GM	Undirected	Directed	Directed	Directed
Loss	KL-divergence	KL-divergence	KL-divergence	JSD
Assumptions	X independent given z	X independent given z	None	N.A.
Samples	Bad	Ok	Good	Good (best)

Table: Comparison of Generative Models

Recent works look at combining these methods: e.g. Adversarial Autoencoders (Makhzani 2015), PixelVAE (Gulrajani 2016) and PixelGAN Autoencoders (Makhzani 2017)



Source: Ian Goodfellow, NIPS 2016 Tutorial: Generative Adversarial Networks

Thank You!

Any Question?