Chapter 7

Variational Auto encoders and Generative Adversarial Networks

1. Variational Auto-encoders
2. Generative Adversarial Networks
3. Some Architectures (DCGAN, CGAN, ACGAN,SRGAN , InfoGAN...)

Deep Generative Models (DGMs)

Deep Generative Models are DNN architectures to generate new, synthetic instances of data that can pass for real data. They are used widely in image generation, video generation and voice generation.

Applications: Art, Animation & Entertainment , Design & Recommender systems, Data Augmentation , Simulators, Problem Solvers, Image/Video Enhancement and Repairing, Super resolution, D-Noising, Compressing, In Social Robots and Autonomous Driving Systems,….

DGMs use usually two neural networks, pitting one along (cooperative mechanism) or against (competition or adversarial mechanisms) the other.

Popular DGMs include:

1. **Variational Auto-Encoders (VAEs)**
2. **Generative Adversarial Neural (GANs)**
3. **Deep Auto Rgressive Modles***

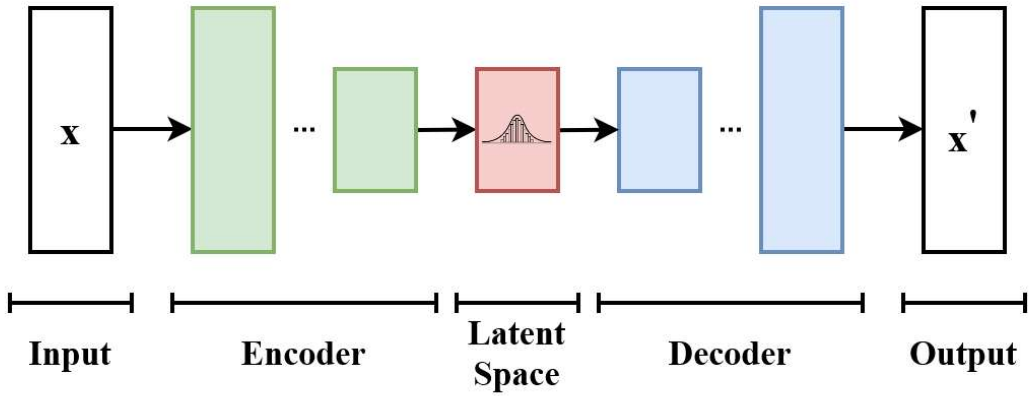
***DARNs (Deep AutoRegressive Networks) are generative sequential models**, and are therefore often compared to other generative networks like GANs or VAEs; however, they are also sequence models and show promise in traditional sequence challenges like language processing and audio generation.

1. Variational autoencoder

In machine learning, a **variational autoencoder (VAE)**, is an artificial neural network architecture introduced by Diederik P. Kingma and Max Welling, belonging to the families of probabilistic graphical models and variational Bayesian methods.[*]

[*] Auto-Encoding Variational Bayes, Submitted on 20 Dec 2013 (v1), last revised 10 Dec 2022
Diederik P Kingma, Max Welling

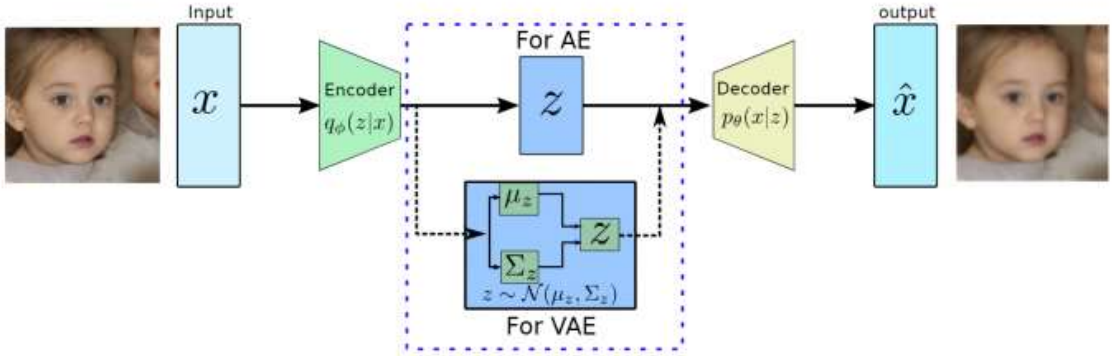
Input: One, Two or three D signals (Image, Video, Time Series, Text,...)
Encoder(Decoder): MLP, Convolution, LSTM, Self Attention, ...
Latent Space: Gaussian Distribution Space.
Output: A reconstruction or regeneration form of Input



Autoencoders have found wide applications in **dimensionality reduction, object detection, image classification, and image denoising applications**. Variational Autoencoders (VAEs) can be regarded as enhanced Autoencoders where a Bayesian approach is used to learn the probability distribution of the input data and to generate new data models.

VAEs versus autoencoder

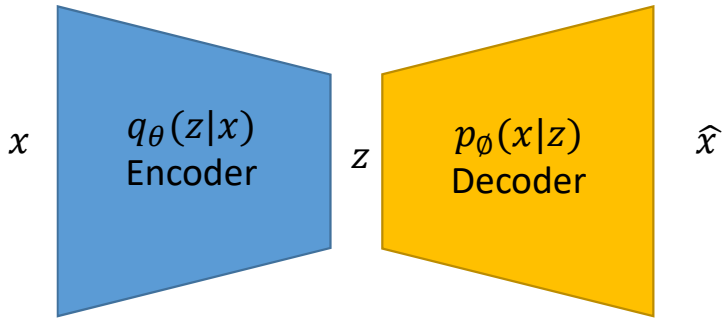
- The architecture of the VAE actually closely resembles that of the Autoencoder, with the main difference being that **the input is encoded into two vectors (μ_z, Σ_z), rather than one (z)**.
- These two vectors (μ_z, Σ_z) are used to define a normal distribution $N(0,1)$, where the latent representation of the input is then drawn from.



Training of VAEs

During the training of autoencoders, we would like to utilize the unlabeled data and try to minimize the following quadratic loss function:

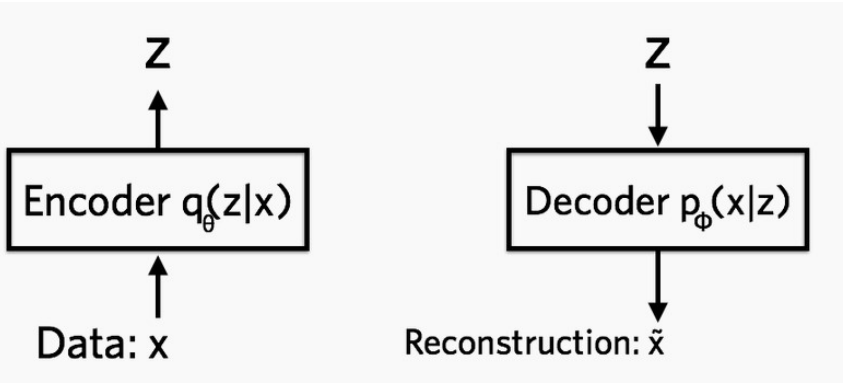
$$\mathcal{L}(\theta, \phi) = ||x - \hat{x}||^2$$



The above equation tries to minimize the distance between the original input and reconstructed image as shown in Figure

VAE has a quite similar architecture to AE except for the bottleneck part.
In AEs, the encoder converts high dimensional input data to low dimensional latent representation in a vector form.
On the other hand, VAE's encoder learns the mean vector and standard deviation diagonal matrix such that $z \sim N(\mu_z, \mathbf{\Sigma_z})$ as it will be performing probabilistic generation of data. Therefore the encoder and decoder should be probabilistic.

The *encoder* is a neural network. Its input is a datapoint x, its output is a hidden representation z, and it has weights and biases θ.
To be concrete, let's say x is a 28 by 28-pixel photo of a handwritten number. The encoder 'encodes' the data which is 784 dimensional into a latent (hidden) representation space z , which is much less than 784 dimensions. This is typically referred to as a 'bottleneck' because the encoder must learn an efficient compression of the data into this lower-dimensional space.
Let's denote the encoder $q_\theta(z|x)$. We note that the lower-dimensional space is stochastic: the encoder outputs parameters to $q_\theta(z|x)$, which is a Gaussian probability density. We can sample from this distribution to get noisy values of the representations z.



The *decoder* is another neural net. Its input is the representation z, it outputs the parameters to the probability distribution of the data, and has weights and biases $\emptyset$. The decoder is denoted by $p_\emptyset(x|z)$. Running with the handwritten digit example, let's say the photos are black and white and represent each pixel as 0 or 1. The probability distribution of a single pixel can be then represented using a Bernoulli distribution. The decoder gets as input the latent representation of a digit z and outputs 784 Bernoulli parameters, one for each of the 784 pixels in the image. The decoder 'decodes' the real-valued numbers in z into 784 real-valued numbers between 0 and 1.

Loss Function for VAEs

Information from the original 784-dimensional vector cannot be perfectly transmitted, because the decoder only has access to a summary of the information (in the form of a less-than-784-dimensional vector z). How much information is lost? We measure this using the reconstruction log-likelihood $\log(p_{\emptyset}(x|z))$ whose units are nats. This measure tells us how effectively the decoder has learned to reconstruct an input image x given its latent representation z

The loss function of the variational autoencoder is the negative log-likelihood with a regularizer.
Because there are no global representations that are shared by all datapoints, we can decompose the loss function into only terms that depend on a single datapoint l_i. The total loss is then $\sum_{i=1}^{N} l_i$ for N total datapoints. The loss function l_i for x_i is:

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \,||\, p(z))$$

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i \mid z)] + \mathbb{KL}(q_\theta(z \mid x_i) \mid\mid p(z))$$

Evidence Lower Bound (ELBO) loss

- The first term is the reconstruction loss, or expected negative log-likelihood of the ith datapoint. The expectation is taken with respect to the encoder's distribution over the representations. This term encourages the decoder to learn to reconstruct the data. If the decoder's output does not reconstruct the data well, statistically we say that the decoder parameterizes a likelihood distribution that does not place much probability mass on the true data.
- the second term is a regularizer that we throw in (we'll see how it's derived later). This is the Kullback-Leibler divergence between the encoder's distribution $q_\theta(z|x)$ and $p(z)$. This divergence measures how much information is lost (in units of nats) when using q to represent p. It is one measure of how close q is to p.

In practical cases where empirical data are available

For VAEs, the KL loss is equivalent to the *sum* of all the KL divergences between the *component* $X_i \sim N(\mu_i, \sigma_i^2)$ in **X**, and the standard normal. It's minimized when $\mu_i = 0$, $\sigma_i = 1$.

A typical Reconstruction Loss

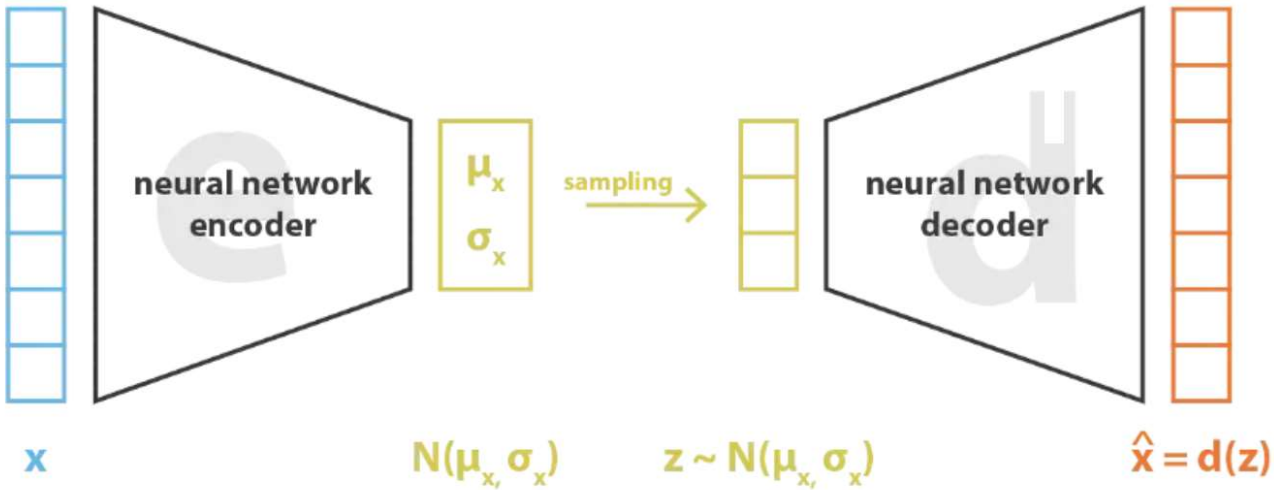
$$\mathcal{L}(\theta, \phi) = ||x - \hat{x}||^2$$

A typical KL Loss

$$\sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1$$

Understanding Variational Autoencoders (VAEs)

Joseph Rocaa, 24 sep 2019

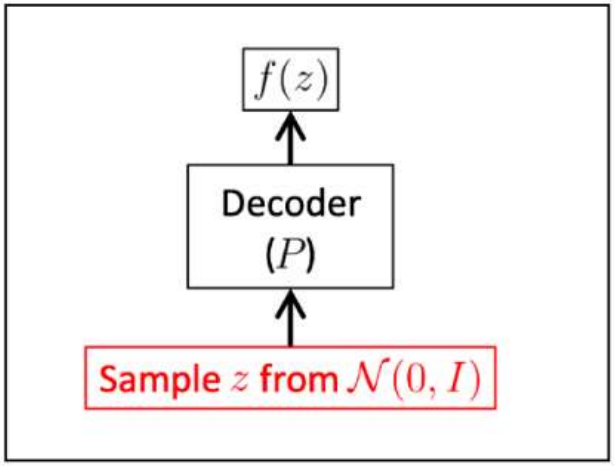


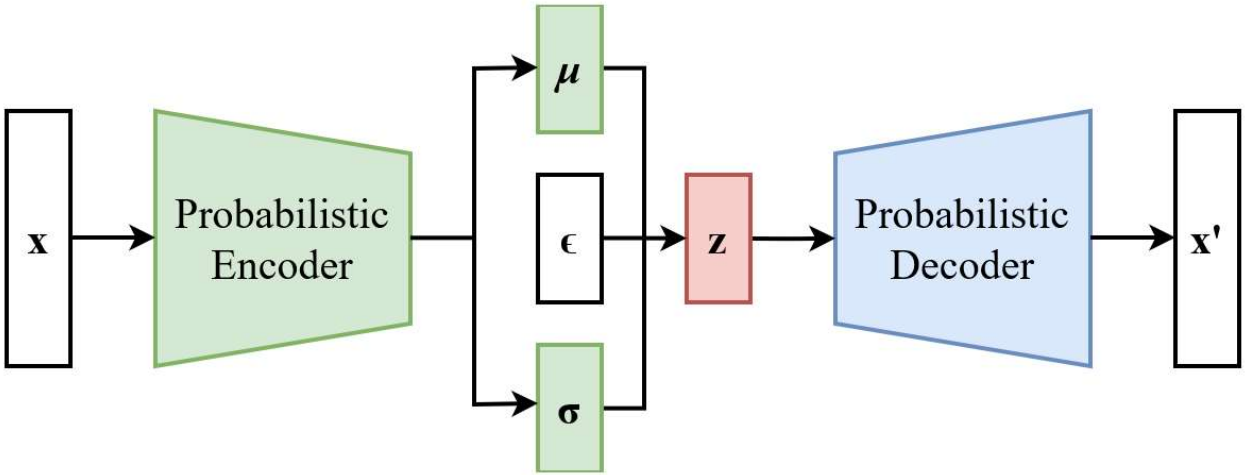
$$\text{loss} = \|x - \hat{x}\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

In variational autoencoders, the loss function is composed of a reconstruction term (that makes the encoding-decoding scheme efficient) and a regularisation term (that makes the latent space regular).

Generating samples from a VAE

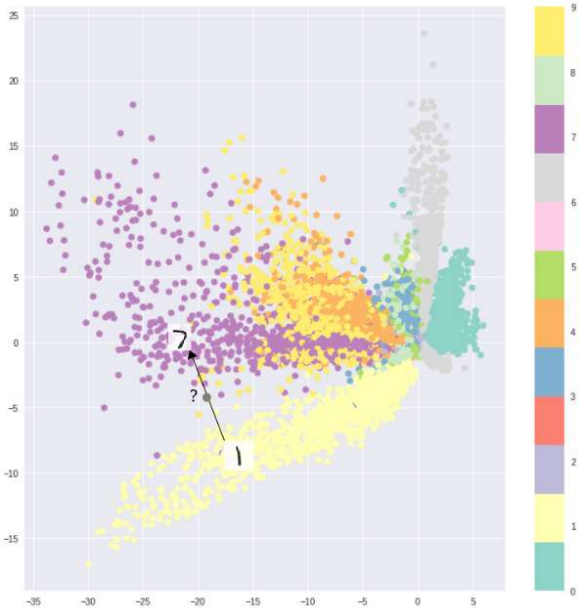
Assume a trained VAE
• Generating a sample:
1. Draw a sample from $p(\epsilon)$
2. Run z through decoder to get $p(x|z)$
• It consists of parameters of a Bernoulli or multinomial
3. Obtain sample from this distribution





The problem with standard autoencoders for generation

- The fundamental problem with autoencoders, for generation, is that the latent space they convert their inputs to and where their encoded vectors lie, may not be continuous, or allow easy interpolation.

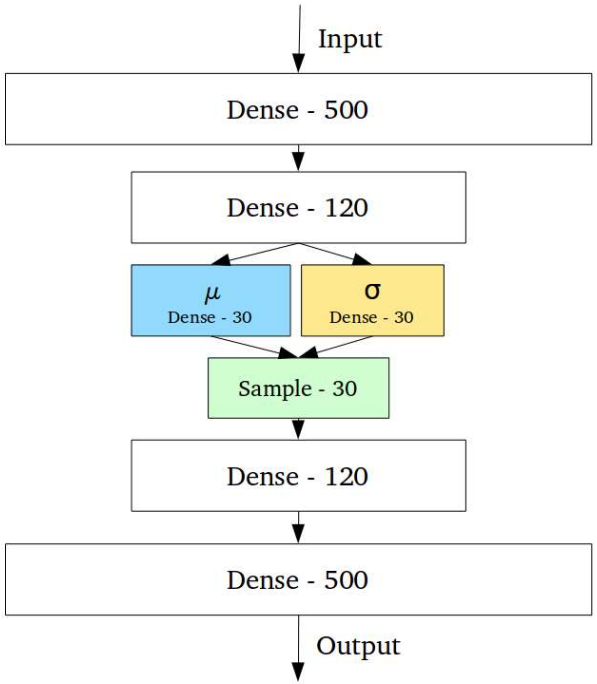


For example, training an autoencoder on the MNIST dataset, and visualizing the encodings from a 2D latent space reveals the formation of distinct clusters.
This makes sense, as distinct encodings for each image type makes it far easier for the decoder to decode them. This is fine if you're just *replicating* the same images.
But when you're building a *generative* model, you **don't** want to prepare to *replicate* the same image you put in. You want to randomly sample from the latent space, or generate variations on an input image, from a continuous latent space.

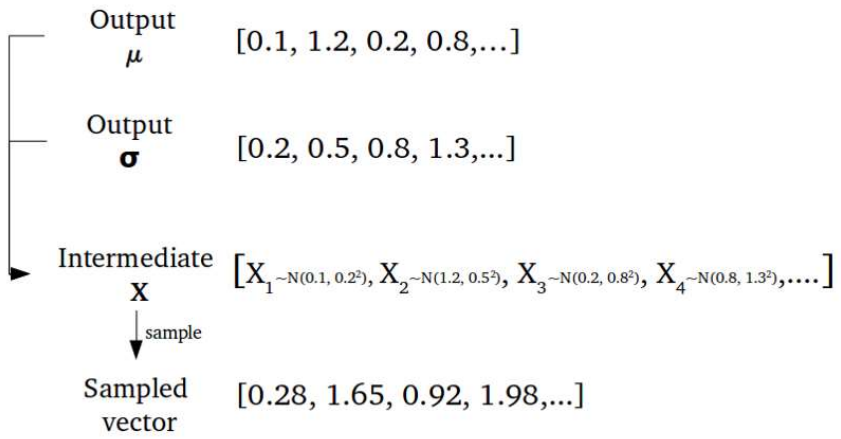
If the space has discontinuities (eg. gaps between clusters) and you sample/generate a variation from there, the decoder will simply generate an unrealistic output, because the decoder has *no idea* how to deal with that region of the latent space. During training, it *never saw* encoded vectors coming from that region of latent space.

Variational Autoencoders

- Variational Autoencoders (VAEs) have one fundamentally unique property that separates them from vanilla autoencoders, and it is this property that makes them so useful for generative modeling: their latent spaces are, *by design,* continuous, allowing easy random sampling and interpolation.
- It achieves this by doing something that seems rather surprising at first: making its encoder not output an encoding vector of size *n,* rather, outputting two vectors of size n: a vector of means**, μ**, and another vector of standard deviations, **σ**.



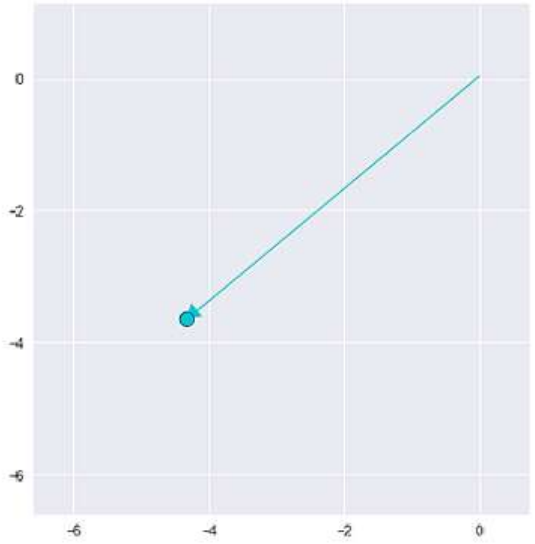
- They form the parameters of a vector of random variables of length n, with the *i* th element of **μ** and **σ** being the mean and standard deviation of the *i* th random variable, **X** i, from which we sample, to obtain the sampled encoding which we pass onward to the decoder:



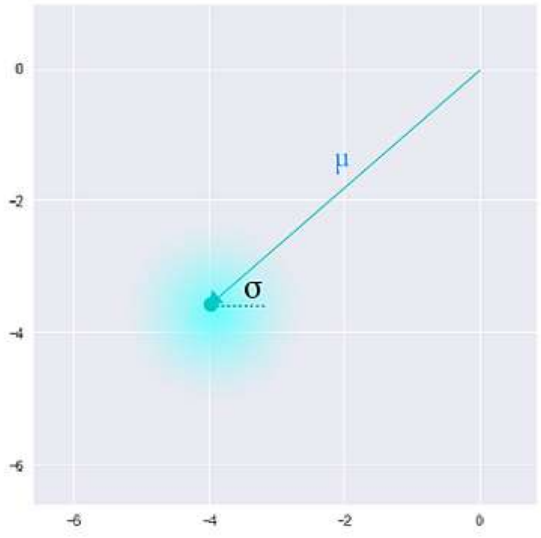
- Stochastically generating encoding vectors

Reconstruction(AEs) and Stochastic Generation(VAEs)

- This stochastic generation means, that even for the same input, while the mean and standard deviations remain the same, the actual encoding will somewhat vary on every single pass simply due to sampling.



Standard Autoencoder
(direct encoding coordinates)



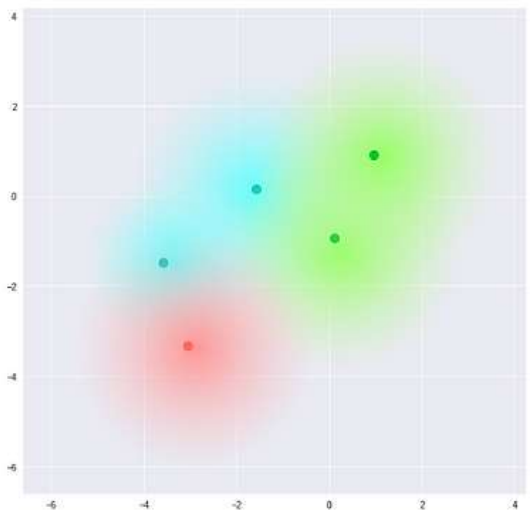
Variational Autoencoder
(μ and σ initialize a probability distribution)

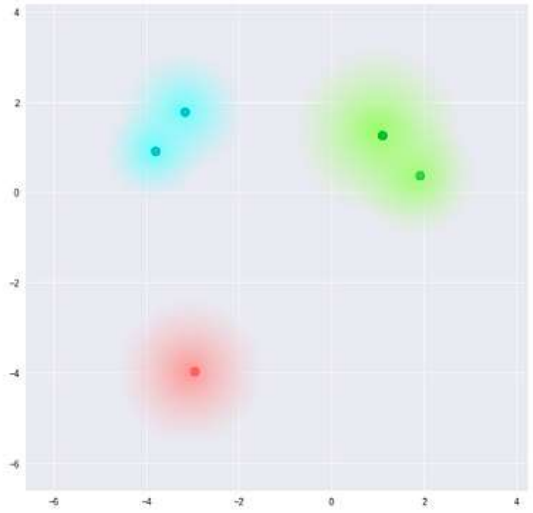
Some Notes

1. Intuitively, the mean vector controls where the encoding of an input should be centered around, while the standard deviation controls the "area", how much from the mean the encoding can vary.
2. As encodings are generated at random from anywhere inside the "circle" (the distribution), the decoder learns that not only is a single point in latent space referring to a sample of that class, but all nearby points refer to the same as well.
3. This allows the decoder to not just decode single, specific encodings in the latent space (leaving the decodable latent space discontinuous), but ones that slightly vary too, as the decoder is exposed to a range of variations of the encoding of the same input during training.

4. The model is now exposed to a certain degree of local variation by varying the encoding of one sample, resulting in smooth latent spaces on a local scale, that is, for similar samples.
5. Ideally, we want overlap between samples that are not very similar too, in order to interpolate *between* classes.
6. However, since there are *no limits* on what values vectors **μ** and **σ** can take on, the encoder can learn to generate very different **μ** for different classes, clustering them apart, and minimize **σ**, making sure the encodings themselves don't vary much for the same sample (that is, less uncertainty for the decoder).
7. This allows the decoder to efficiently reconstruct the *training* data.

- What we ideally want are encodings, *all* of which are as close as possible to each other while still being distinct, allowing smooth interpolation, and enabling the construction of *new* samples.





2.s Generative Adversarial Network (GAN)

Introductory guide to Generative Adversarial Networks (GANs) and their promise!

Faizan Shaikh, June 15, 2017

• Introduction

Neural Networks have made great progress.

1. They now recognize images and voice at levels comparable to humans.
2. They are also able to understand natural language with a good accuracy.

Let us see a few examples where we need human creativity (at least as of now):

•Train an artificial author which can write an article and explain data science concepts to a community in a very simplistic manner by learning from past articles on Analytics Vidhya.

• You are not able to buy a painting from a famous painter which might be too expensive. Can you create an artificial painter which can paint like any famous artist by learning from his / her past collections?

GANs: DNNs like VAEs which can generate new, synthetic instances of data that can pass for real data. They are used widely in image generation, video generation and voice generation.

In GANs, an adversarial mechanism is designed to generate fake patterns which are so similar to real patterns.

Ian Goodfellow (in 2014) introduce GANs for such purposes.

- Yann LeCun, a prominent figure in Deep Learning Domain said in his Quora session that:

"(GANs), and the variations that are now being proposed is the most interesting idea in the last 10 years in ML, in my opinion."

But what is a GAN?

Let us take an analogy to explain the concept:

If you want to get better at something, say chess; what would you do? You would compete with an opponent better than you. Then you would analyze what you did wrong, what he / she did right, and think on what could you do to beat him / her in the next game.

You would repeat this step until you defeat the opponent. This concept can be incorporated to build better models. So simply, for getting a powerful hero (viz generator), we need a more powerful opponent (viz discriminator)!

Another analogy from real life

A slightly more real analogy can be considered as a relation between forger and an investigator.

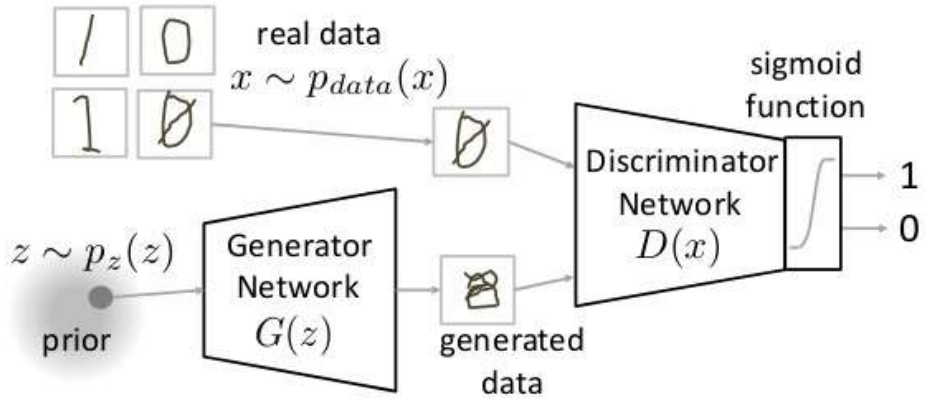
The task of a forger is to create fraudulent imitations of original paintings by famous artists. If this created piece can pass as the original one, the forger gets a lot of money in exchange of the piece.

On the other hand, an art investigator's task is to catch these forgers who create the fraudulent pieces. How does he do it? He knows what are the properties which sets the original artist apart and what kind of painting he should have created. He evaluates this knowledge with the piece in hand to check if it is real or not.

This contest of forger vs investigator goes on, which ultimately makes world class investigators (and unfortunately world class forger); a battle between good and evil.

How do GANs work?

As we saw, there are two main components of a GAN – Generator Neural Network and Discriminator Neural Network.



About the above Figure:

1. The Generator Network takes an random input and tries to generate a sample of data.

- In the above image, we can see that generator G(z) takes a input z from p(z), where z is a sample from probability distribution p(z).

2. It then generates a data which is then fed into a discriminator network D(x).
3. The task of Discriminator Network is to take input either from the real data or from the generator and try to predict whether the input is real or generated.

   It takes an input x from $p_{data}(x)$ where $p_{data}(x)$ is our real data distribution. D(x) then solves a binary classification problem using sigmoid function giving output in the range 0 to 1.

- Let us define the notations we will be using to formalize our GAN,
- Pdata(x) -> the distribution of real data
  X -> sample from pdata(x)
  P(z) -> distribution of generator
  Z -> sample from p(z)
  G(z) -> Generator Network
  D(x) -> Discriminator Network

Now the training of GAN is done (as we saw above) as a **"fight between generator and discriminator."**

This can be represented mathematically as:

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z))]$$

•In our function V(D, G) the first term is entropy that the data from real distribution (pdata(x)) passes through the discriminator (also known as (aka) best case scenario).

The discriminator tries to maximize this to 1.

The second term is entropy that the data from random input (p(z)) passes through the generator, which then generates a fake sample which is then passed through the discriminator to identify the fakeness (aka worst case scenario).

In this term, discriminator tries to maximize it to 0

(i.e. the log probability that the data from generated is fake is equal to 0).

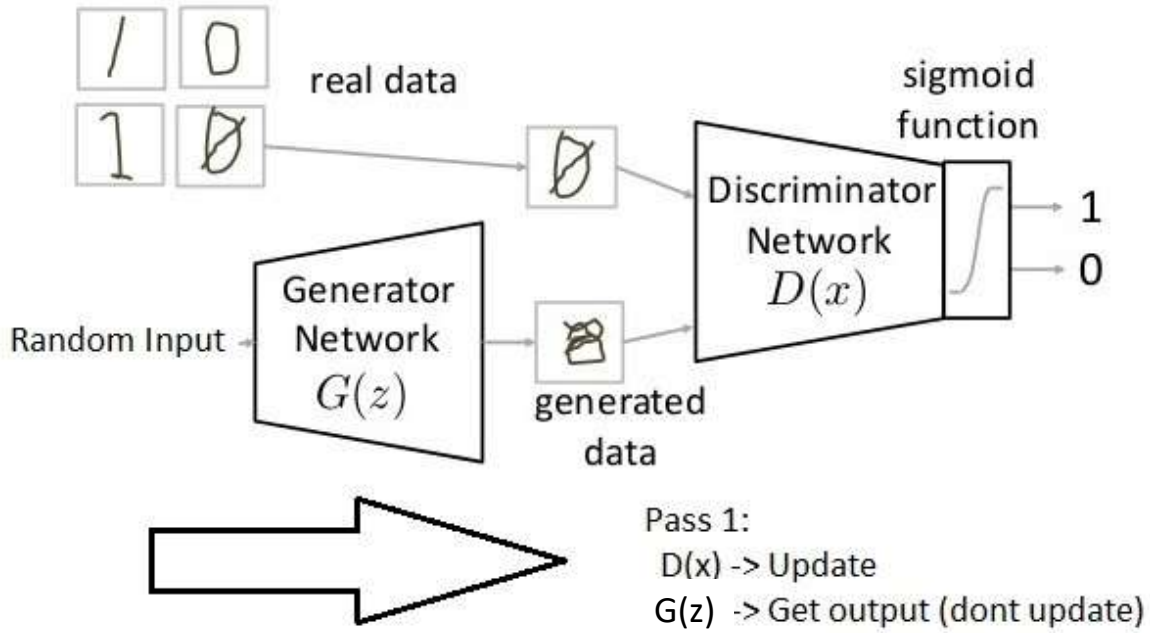
of training a GAN is taken from game theory called the minimax game.

So overall, the discriminator is trying to maximize our function V.
On the other hand, **the task of generator is exactly opposite,**
i.e. it tries to minimize the function V so that the differentiation between real and fake data is bare minimum.

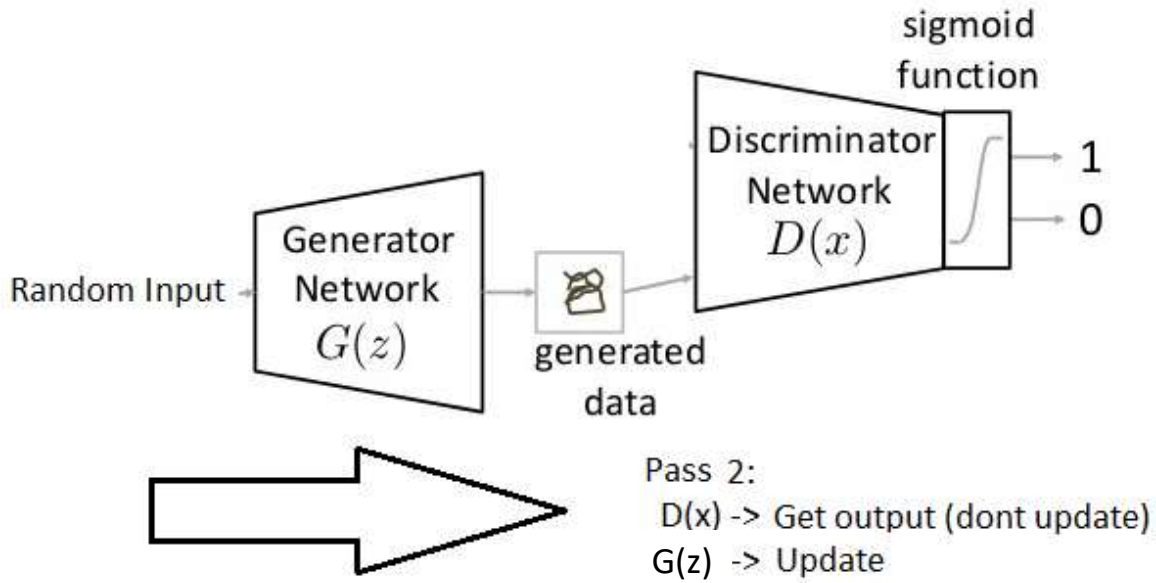
This, in other words is a cat and mouse game between generator and discriminator!

- *Note: This method of training a GAN is taken from game theory called the minimax game.*
-

Pass 1: Train discriminator and freeze generator (freezing means setting training as false. The network does only forward pass and no backpropagation is applied).



Pass 2: Train generator and freeze discriminator.



Steps to train a GAN

- **Step 1: Define the problem.** Do you want to generate fake images or fake text. Here you should completely define the problem and collect data for it.
- **Step 2: Define architecture of GAN.** Define how your GAN should look like. Should both your generator and discriminator be multi layer perceptrons, or convolutional neural networks? This step will depend on what problem you are trying to solve.
- **Step 3: Train Discriminator on real data for n epochs.** Get the data you want to generate fake on and train the discriminator to correctly predict them as real. Here value n can be any natural number between 1 and infinity.
- **Step 4: Generate fake inputs for generator and train discriminator on fake data.** Get generated data and let the discriminator correctly predict them as fake.
- **Step 5: Train generator with the output of discriminator.** Now when the discriminator is trained, you can get its predictions and use it as an objective for training the generator. Train the generator to fool the discriminator.
- **Steps to train a GAN**
- **Step 6: Repeat step 3 to step 5 for a few epochs.**
- **Step 7: Check if the fake data manually if it seems legit. If it seems appropriate, stop training, else go to step 3.** This is a bit of a manual task, as hand evaluating the data is the best way to check the fakeness. When this step is over, you can evaluate whether the GAN is performing well enough.

A pseudocode of GAN training can be thought out as follows

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

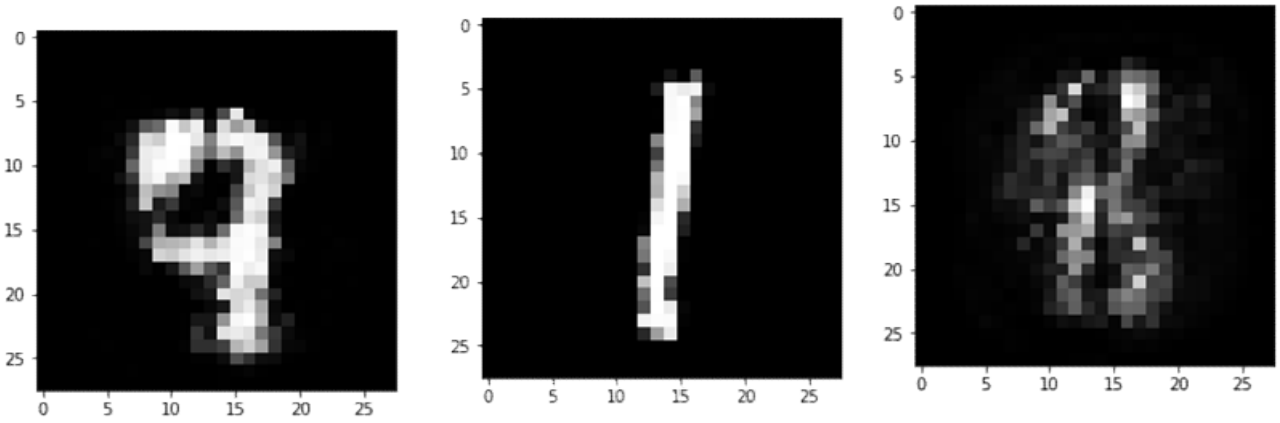
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

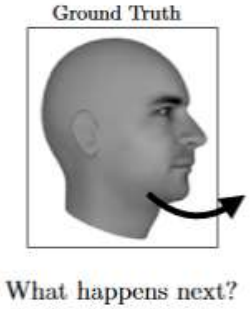
We will try to generate digits by training a GAN

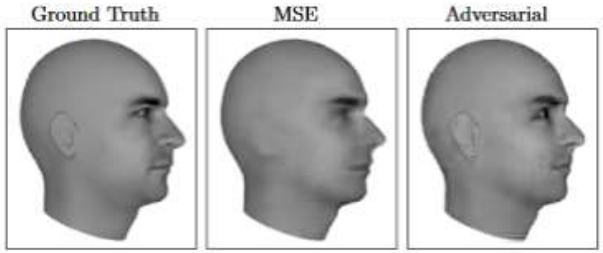
After training for 100 epochs, I got the following generated images



Applications of GAN

Predicting the next frame in a video : You train a GAN on video sequences and let it predict what would occur next





Increasing Resolution of an image : Generate a high resolution photo from a comparatively low resolution.

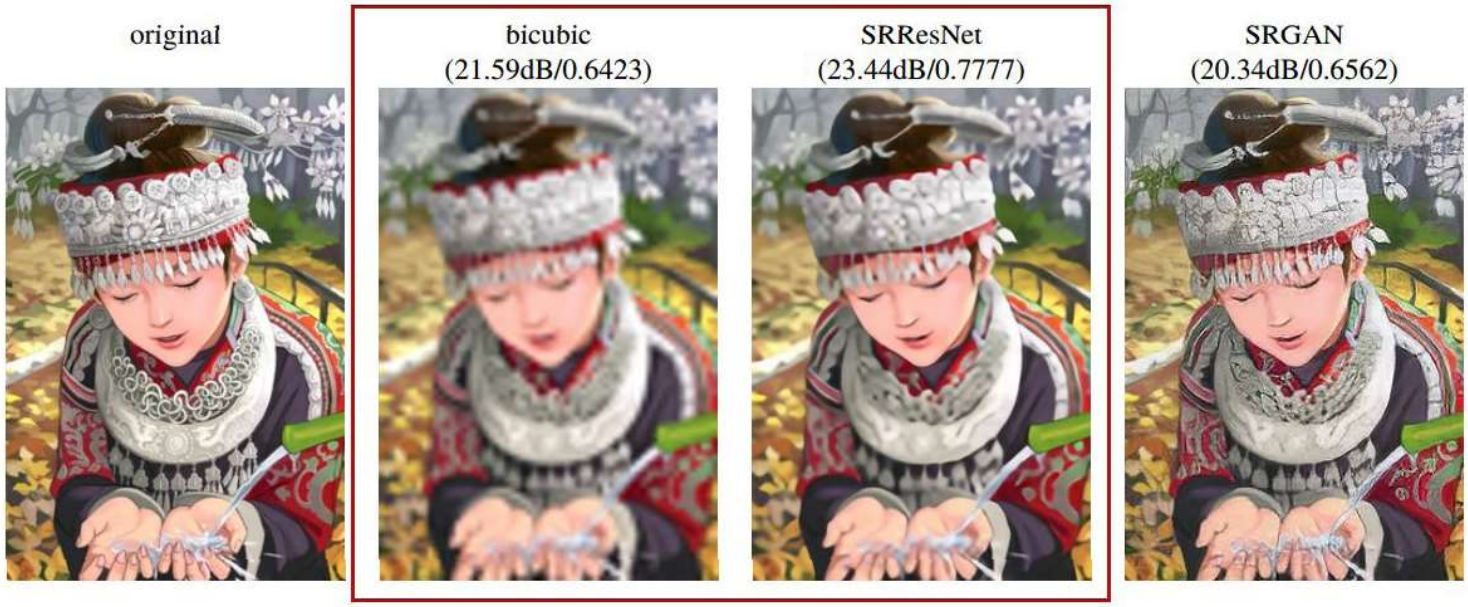
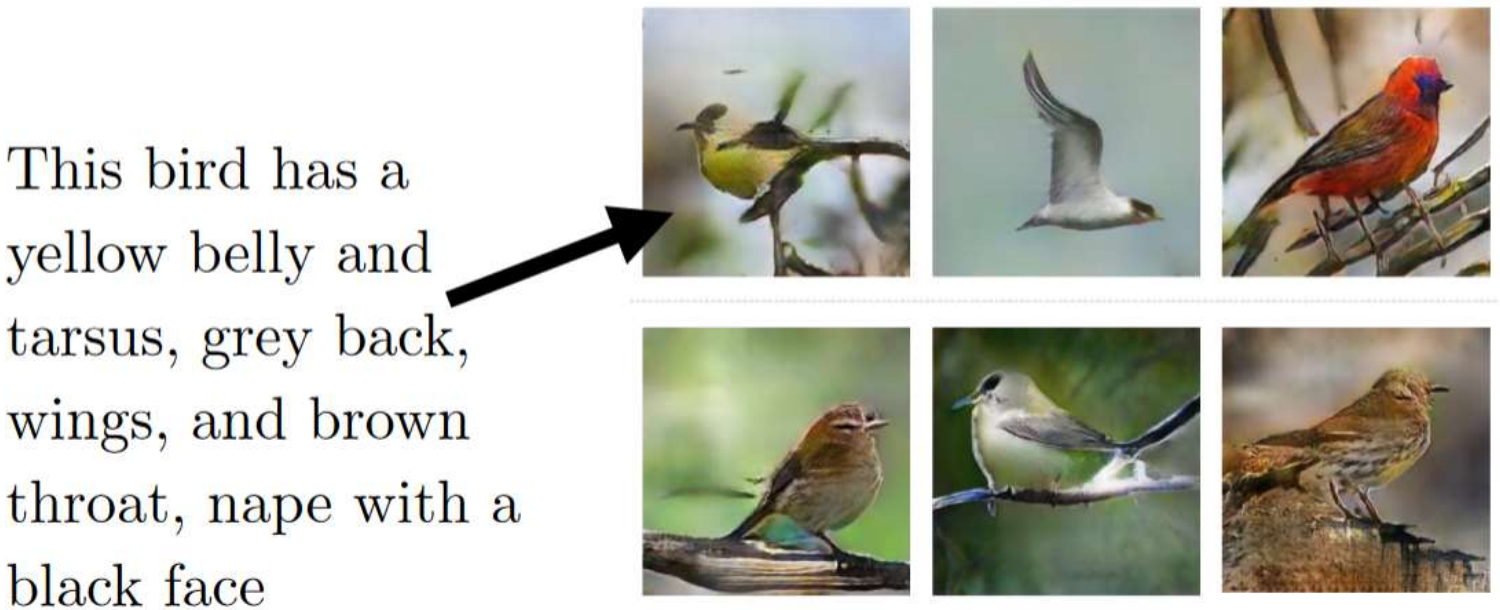
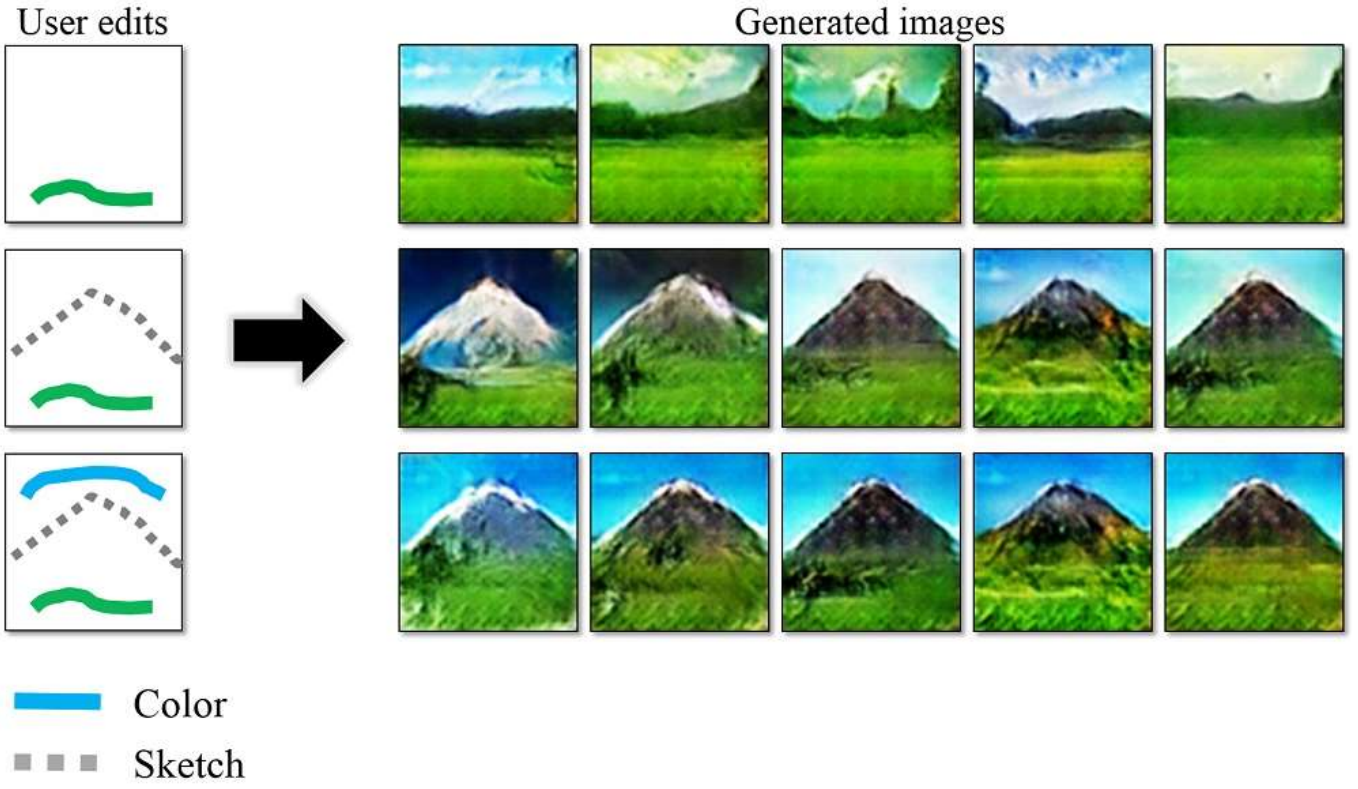


Image to Image Translation : Generate an image from another image. For example, given on the left, you have labels of a street scene and you can generate a real looking photo with GAN. On the right, you give a simple drawing of a handbag and you get a real looking drawing of a handbag.

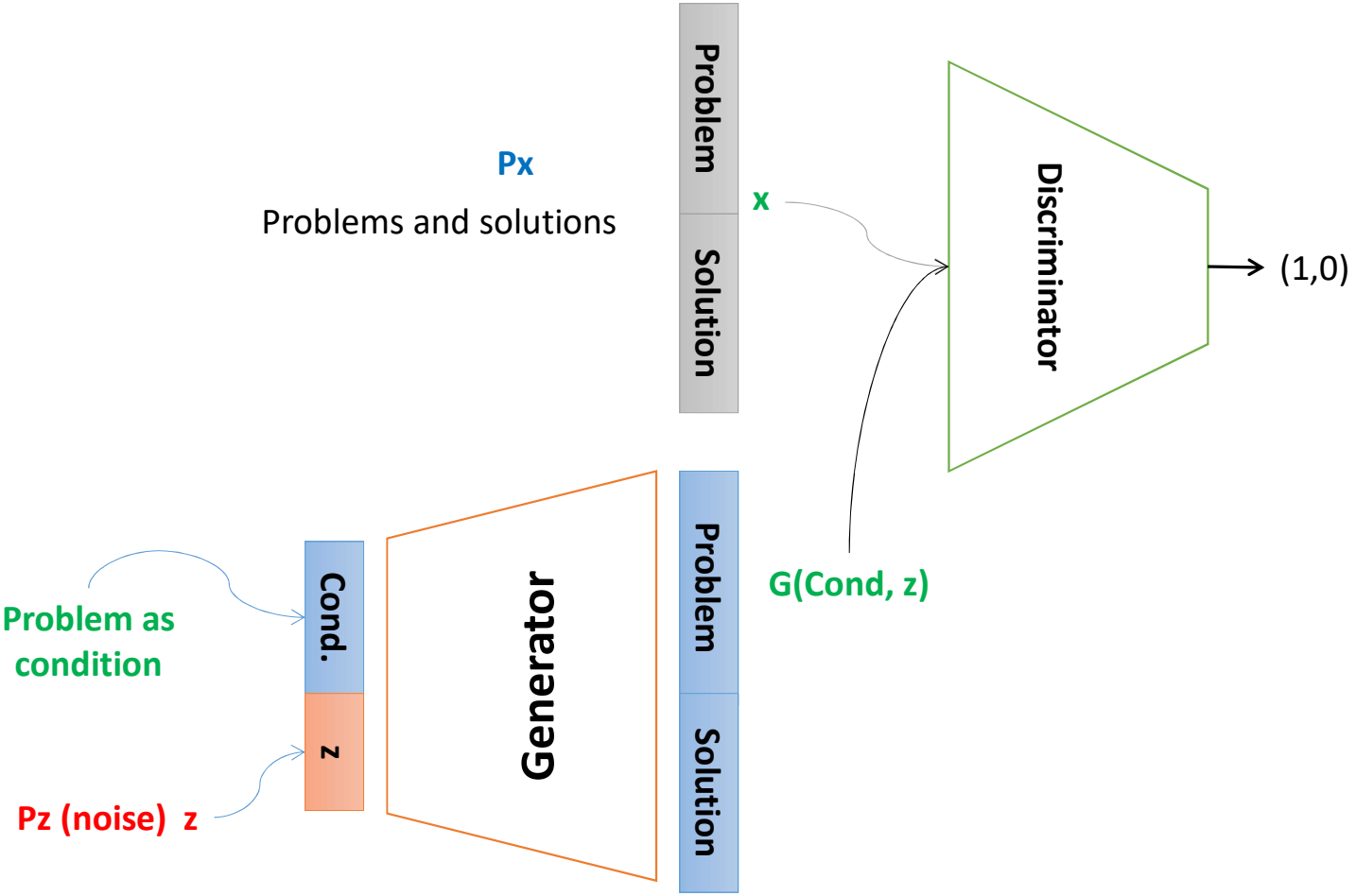
Text to Image Generation : Just say to your GAN what you want to see and get a realistic photo of the



Interactive Image Generation(IGAN) : Draw simple strokes and let the GAN draw an impressive picture for you



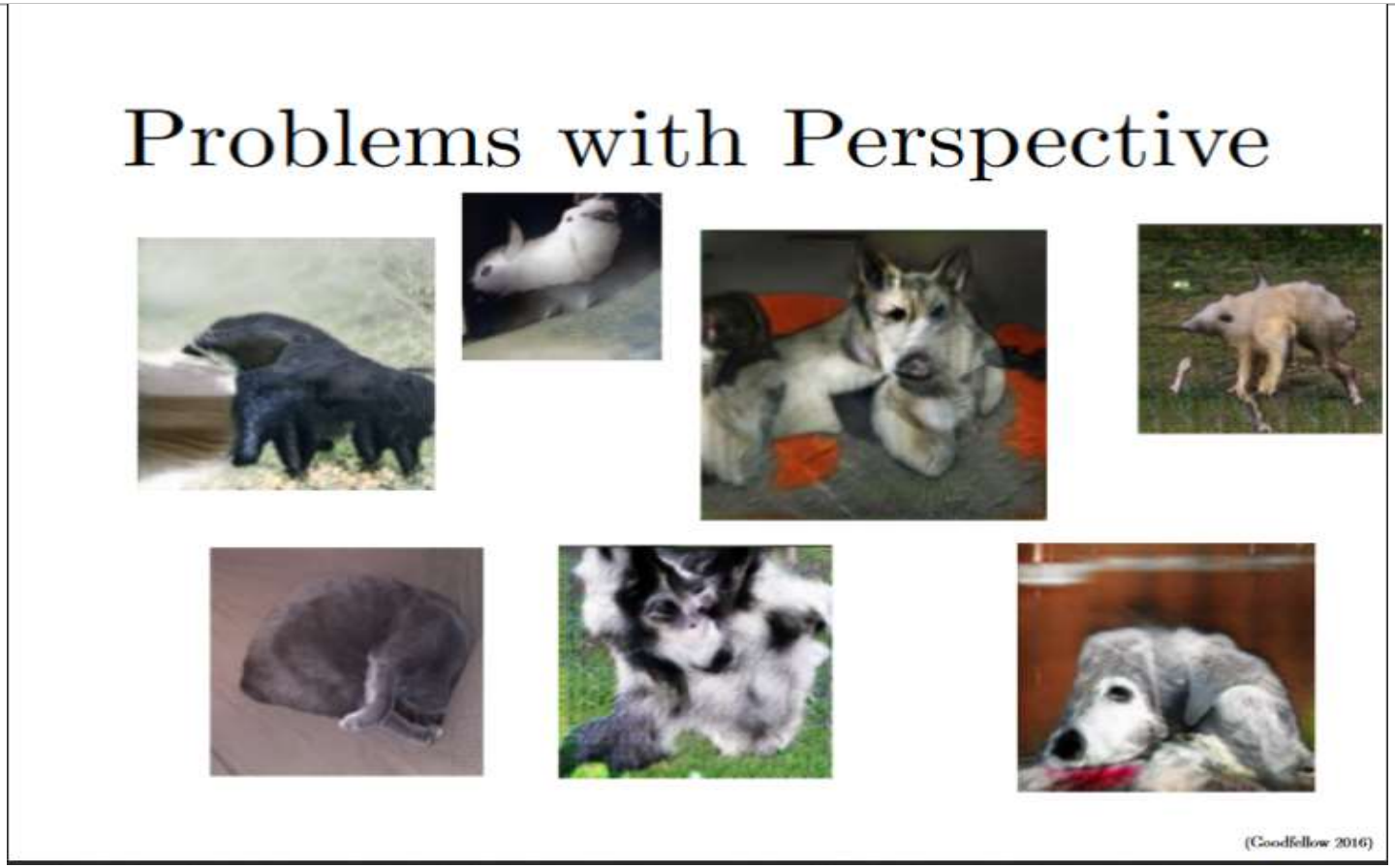
A General Plan
to generate **solutions** for challenging problems



Challenges With GAN

Problem with Counting: GANs fail to differentiate how many of a particular object should occur at a location. As we can see below, it gives more number of eyes in the head than naturally present.

GANs fail to adapt to 3D objects. It doesn't understand perspective, i.e.difference between frontview and backview. As we can see below, it gives flat (2D)



Problems with Global Structures

Same as the problem with perspective, GANs do not understand a holistic structure. For example, in the bottom left image, it gives a generated image of a quadruple cow, i.e. a cow standing on its hind legs and simultaneously on all four legs. That is definitely not possible in real life!

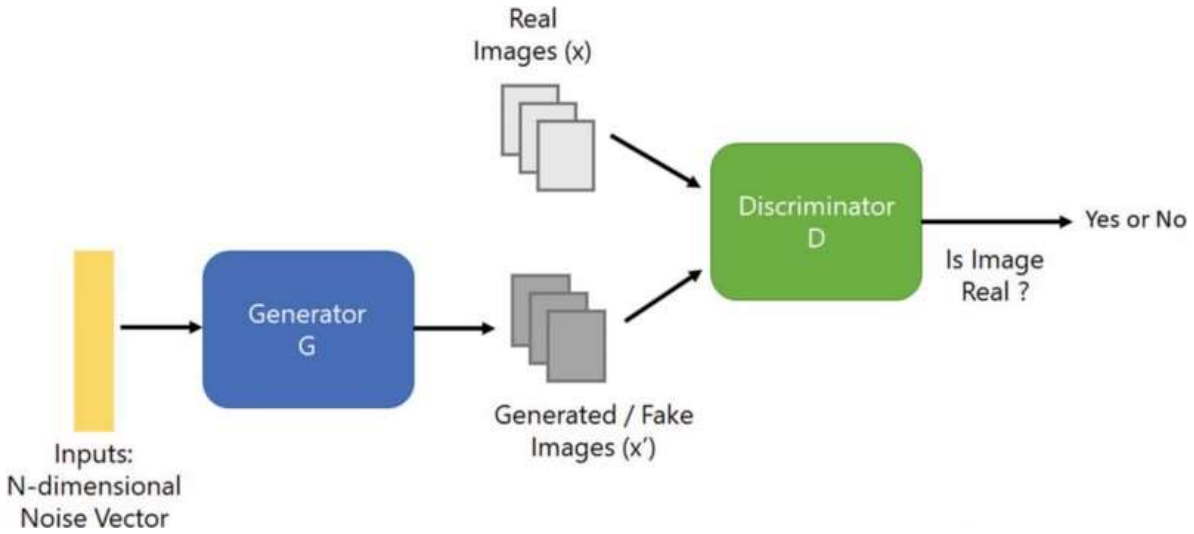
Some Extensions on GANs

1. DCGANs
2. BGAN
3. SRGAN
4. CGAN
5. Auxiliary Classifier GAN
6. Semi-Supervised GAN
7. StackGAN
8. Disco GANS
9. Flow based GANs
10. InfoGANs
11. Wasserstein GAN
12. Bidirectional GAN
13. Context-Conditional GAN
14. Context Encoder
15. Coupled GANs
16. CycleGAN
17. DualGAN
18. LSGAN
19. Pix2Pix
20. PixelDA
21. Wasserstein GAN GP
22. Adversarial Autoencoder

(1) Deep Convolutional GANs (DCGANs)

https://arxiv.org/abs/1710.10196 (Jan 2016)

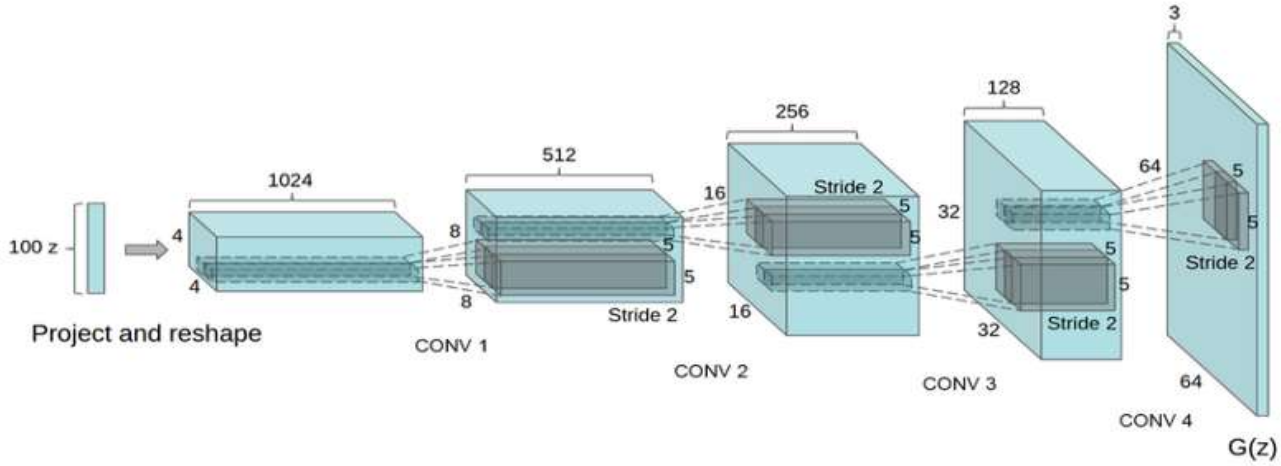
In this article, we will see how a neural net maps from random noise to an image matrix and how using Convolutional Layers in the generator network produces better results.



Original DCGAN architecture(*Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*) have *four* convolutional layers for the **Discriminator** and *four* "four fractionally-strided convolutions" layers for the **Generator.**

(DCGAN) Generator

This network takes in a 100x1 noise vector, denoted z, and maps it into the G(Z) output which is 64x64x3. This architecture is especially interesting the way the first layer expands the random noise. The network goes from 100x1 to 1024x4x4! This layer is denoted 'project and reshape'. We see that following this layer, classical convolutional layers are applied. In the diagram above we can see that the N parameter, (Height/Width), goes from 4 to 8 to 16 to 32, it doesn't appear that there is any padding, the kernel filter parameter F is 5x5, and the stride is 2. You may find this equation to be useful for designing your own convolutional layers for customized output sizes.



we see the network goes from

100x1 → 1024x4x4 → 512x8x8 → 256x16x16 → 128x32x32 → 64x64x3

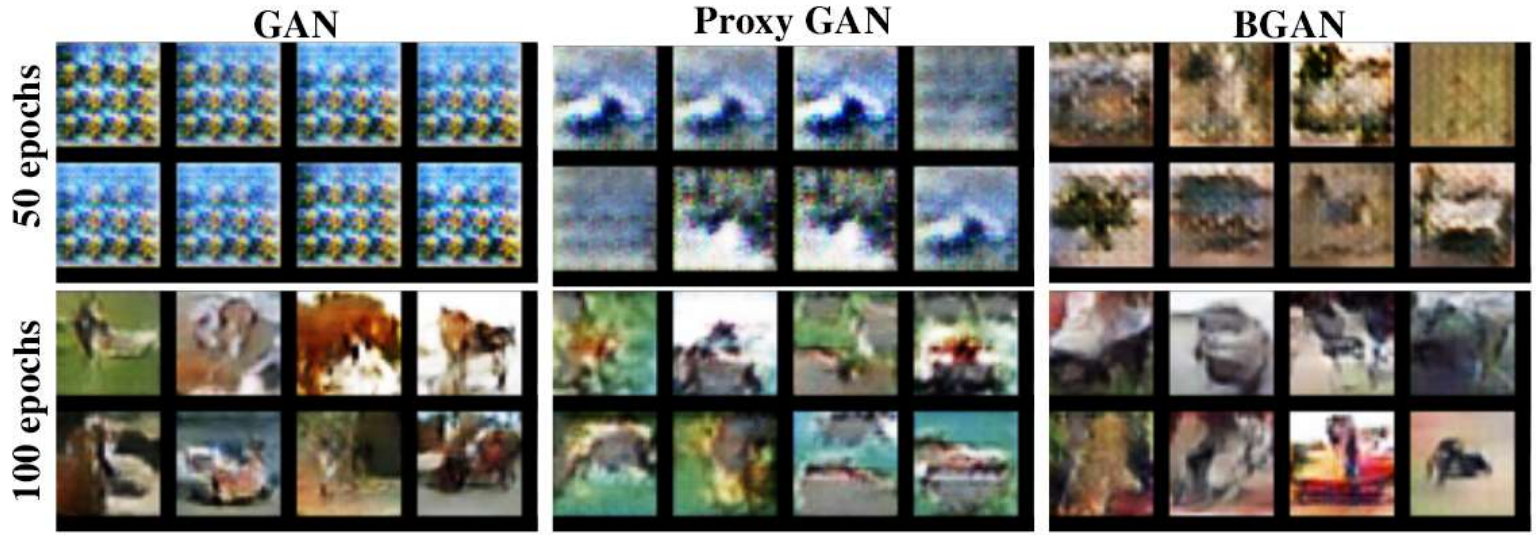
Above is the output from the network presented in the paper, citing that this came after 5 epochs of training. Pretty impressive stuff.

(2) Boundary-Seeking Generative Adversarial Networks(BGAN)

https://arxiv.org/abs/1702.08431 (2017)

- We introduce a method for training GANs with discrete data that uses the estimated difference measure from the discriminator to compute importance weights for generated samples, thus providing a policy gradient for training the generator.
- The importance weights have a strong connection to the decision boundary of the discriminator, and we call our method boundary-seeking GANs (BGANs).
- We demonstrate the effectiveness of the proposed algorithm with discrete image and character-based natural language generation.
- In addition, the boundary-seeking objective extends to continuous data, which can be used to improve stability of training, and we demonstrate this on Celeba, Large-scale Scene Understanding (LSUN) bedrooms, and Imagenet without conditioning.

Training a GAN with different generator loss functions and 5 updates for the generator for every update of the discriminator.
Over-optimizing the generator can lead to instability and poorer results depending on the generator objective function.
Samples for GAN and GAN with the proxy loss are quite poor at 50 discriminator epochs (250 generator epochs), while BGAN is noticeably better.
At 100 epochs, these models have improved, though are still considerably behind BGAN.



CelebA

Imagenet

LSUN

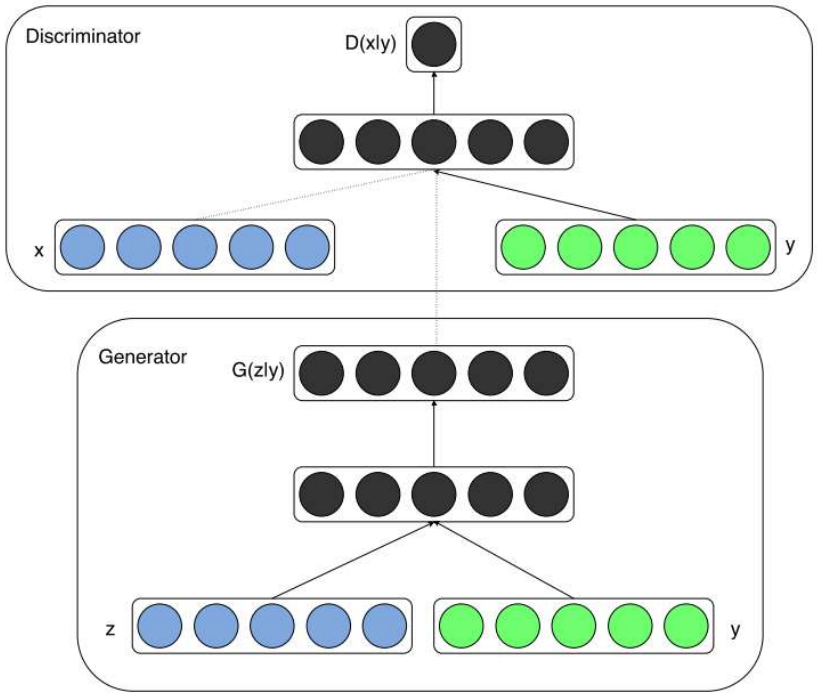
Figure 3: Highly realistic samples from a generator trained with BGAN on the CelebA and LSUN datasets. These models were trained using a deep ResNet architecture with gradient norm regularization (Roth et al., 2017). The Imagenet model was trained on the full 1000 label dataset without conditioning.

(3) Conditional GANs (cGANs)

Mehdi Mirza, Simon Osindero

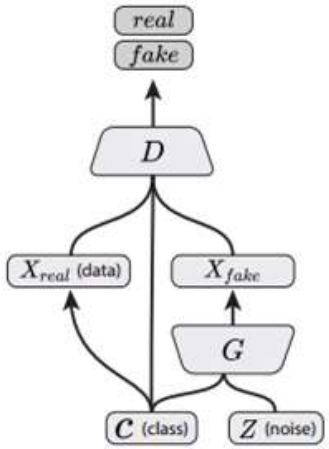
(Submitted on 6 Nov 2014)

Source: https://arxiv.org/pdf/1411.1784.pdf



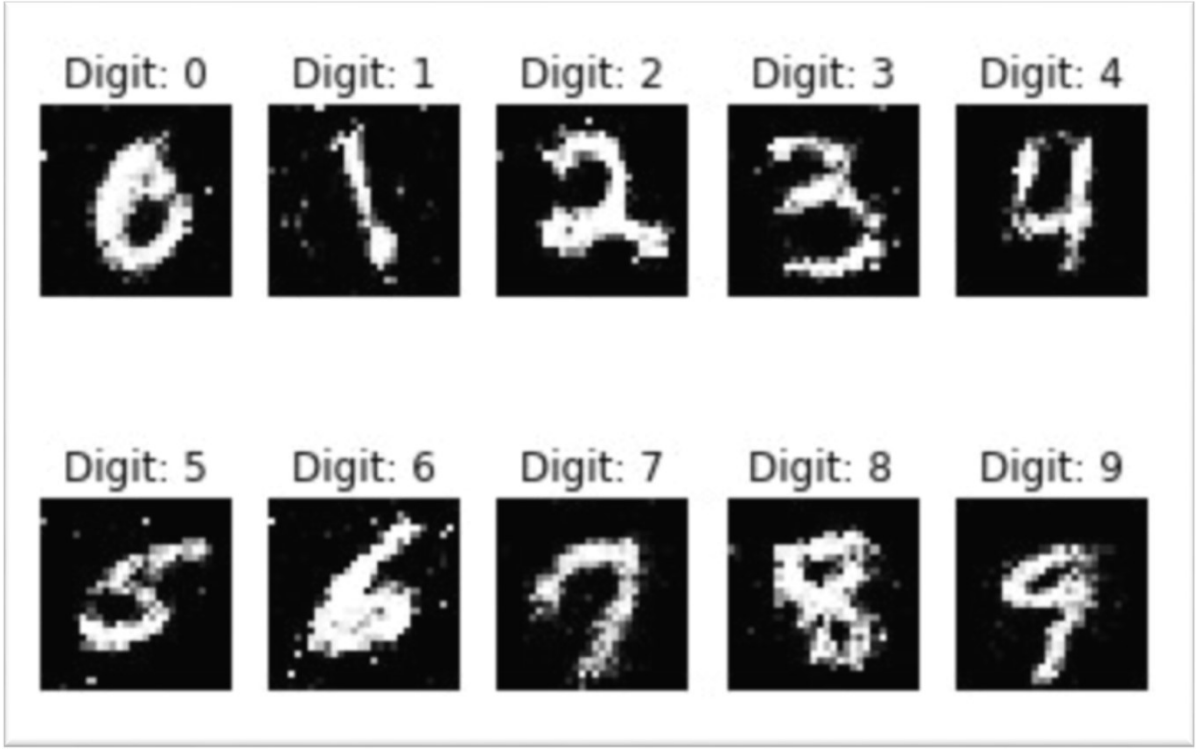
1. These GANs use extra label information and result in better quality images and are able to control how generated images will look. cGANs learn to produce better images by exploiting the information fed to the model.

2. it does give the end-user a mechanism for controlling the Generator output



Conditional GAN
(Mirza & Osindero, 2014)

The results of Conditional GANs are very impressive. They allow for much greater control over the final output from the generator

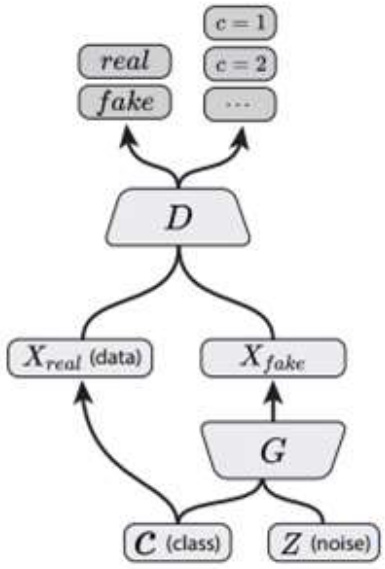


Results testing Conditional GANs on MNIST

AC-GAN

Auxiliary Classifier GANs

- By adding an auxiliary classifier to the discriminator of a GAN, the discriminator produces not only a probability distribution over sources but also probability distribution over the class labels.
- Source: Augustus Odena, Christopher Olah, Jonathon Shlens. Conditional Image Synthesis with Auxiliary Classifier GANs. 2016.
-



AC-GAN
(Present Work)

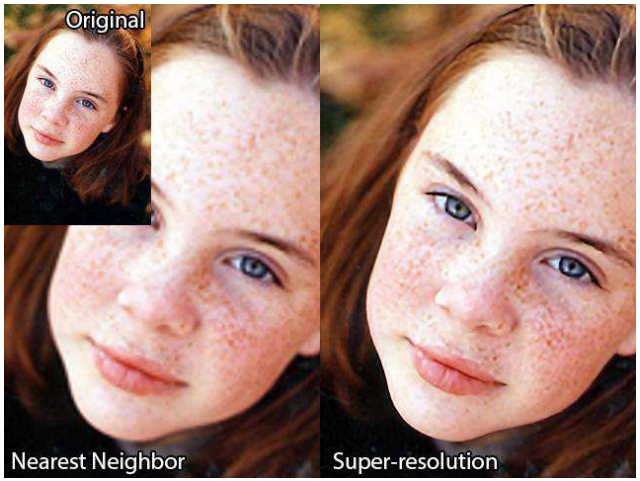
The sample generated images from CIFAR-10 dataset.

In the AC-GAN paper, 100 different GAN models each handle 10 different classes from the ImageNet dataset consisting of 1,000 different object categories

The sample generated images from ImageNet dataset.

(4) Super Resolution GAN (SR GAN)

- Super-resolution is a task concerned with upscaling images from low-resolution sizes such as 90 x 90, into high-resolution sizes such as 360 x 360.



Source: Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.
Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi.

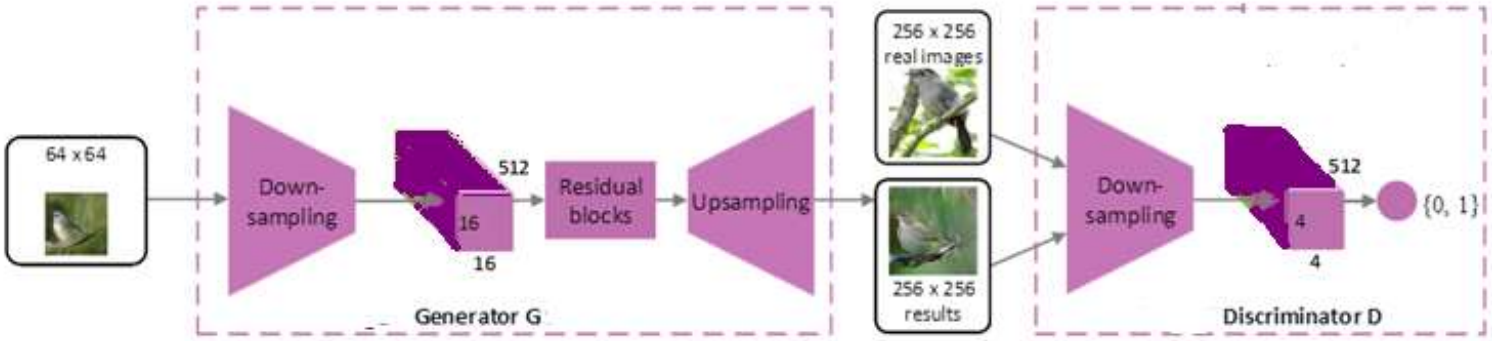
4× SRGAN (proposed)

original

In this example, 90 x 90 to 360 x 360 is denoted as an up-scaling factor of 4x.

•These networks learn a mapping from the low-resolution patch through a series of convolutional, fully-connected, or transposed convolutional layers into the high-resolution patch.

For example, this network could take a 64x 64low-resolution patch, convolve over it a couple times such that the feature map is something like 16 x 16 x 512, flatten it into a vector, apply a couple of fully-connected layers, reshape it, and finally, up-sample it into a 256x 256 high-resolution patch through transposed convolutional layers.



(5) StackGAN

Text to Photo-realistic Image Synthesis with Stacked generative Adversarial Networks, Han Zhang, ...

- The authors of this paper propose a solution to the problem of synthesizing high-quality images from text descriptions in computer vision. They propose Stacked Generative Adversarial Networks (StackGAN) to generate 256x256 photo-realistic images conditioned on text descriptions. They decompose the hard problem into more manageable sub-problems through a sketch-refinement process.

The Stage-I GAN sketches the primitive shape and colors of the object based on the given text description, yielding Stage-I low-resolution images. The Stage-II GAN takes Stage-I results and text descriptions as inputs, and generates high-resolution images with photo-realistic details.

The architecture of the proposed StackGAN.

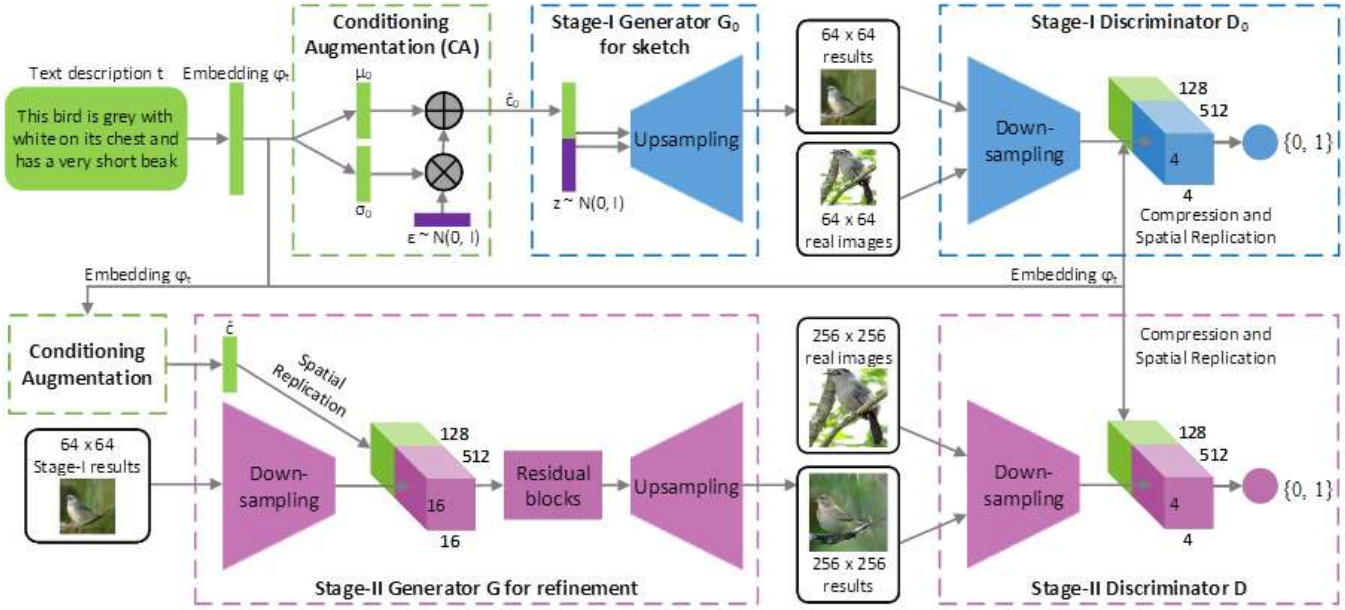


Figure 2. The architecture of the proposed StackGAN. The Stage-I generator draws a low-resolution image by sketching rough shape and basic colors of the object from the given text and painting the background from a random noise vector. Conditioned on Stage-I results, the Stage-II generator corrects defects and adds compelling details into Stage-I results, yielding a more realistic high-resolution image.

Comparison

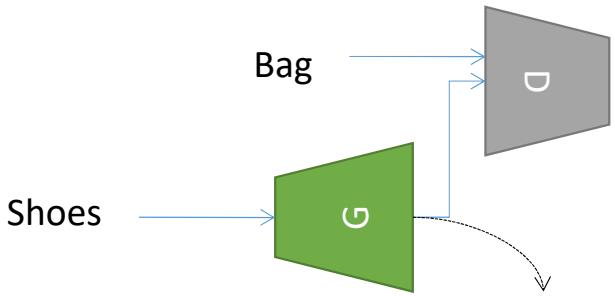
Figure 3. Example results by our StackGAN, GAWWN [24], and GAN-INT-CLS [26] conditioned on text descriptions from CUB test set.

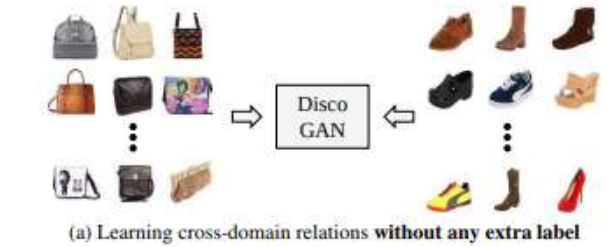
Comparison

Figure 4. Example results by our StackGAN and GAN-INT-CLS [26] conditioned on text descriptions from Oxford-102 test set (leftmost four columns) and COCO validation set (rightmost four columns).

(6) Discover Cross-Domain Relations with GANs(Disco GANS)

- The authors of this paper propose a method based on generative adversarial networks that learns to discover relations between different domains (without any extra labels). Using the discovered relations, the network transfers style from one domain to another..





(a) Learning cross-domain relations **without any extra label**

(b) Handbag images (input) & **Generated** shoe images (output)

(c) Shoe images (input) & **Generated** handbag images (output)

(7) Flow-Based GANs

Oct 13, 2018 by Lilian Weng

Here is a quick summary of the difference between GAN, VAE, and flow-based generative models:

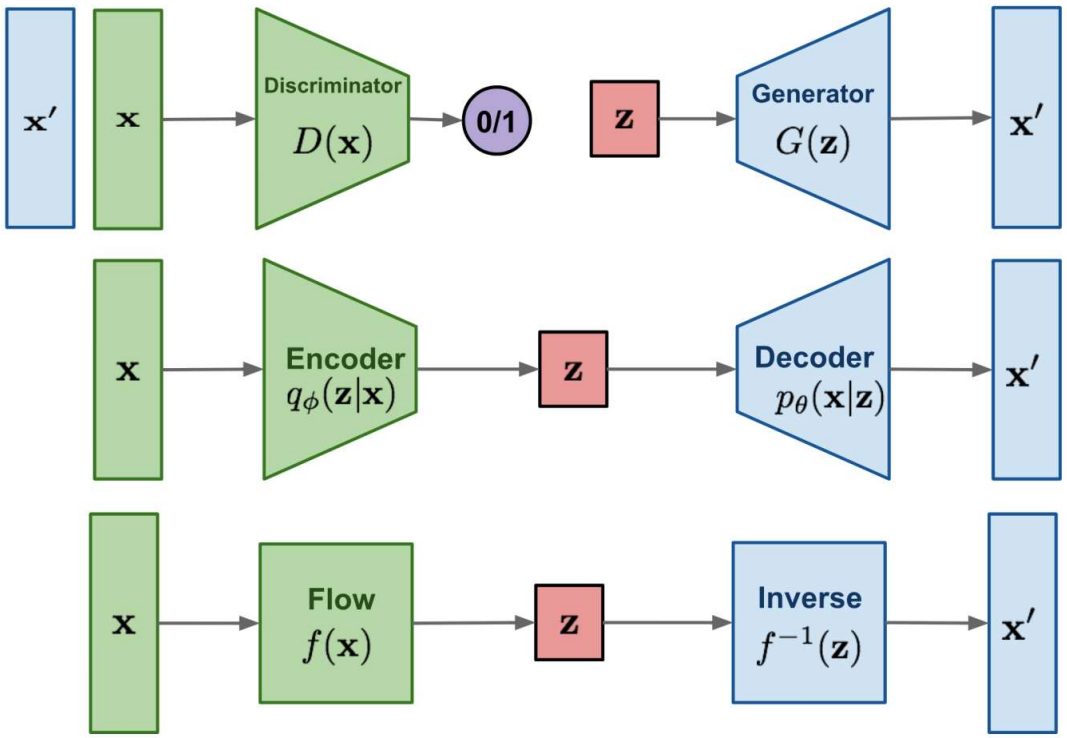
1. Generative adversarial networks: GAN provides a smart solution to model the data generation, an unsupervised learning problem, as a supervised one. The discriminator model learns to distinguish the real data from the fake samples that are produced by the generator model. Two models are trained as they are playing a minimax game.
2. Variational autoencoders: VAE inexplicitly optimizes the log-likelihood of the data by maximizing the evidence lower bound (ELBO).
3. Flow-based generative models: A flow-based generative model is constructed by a sequence of invertible transformations. Unlike other two, the model explicitly learns the data distribution $p(\mathbf{x})$ and therefore the loss function is simply the negative log-likelihood.

Types of Generative Models

GAN: minimax the classification error loss.

VAE: maximize ELBO.

Flow-based generative models: minimize the negative log-likelihood

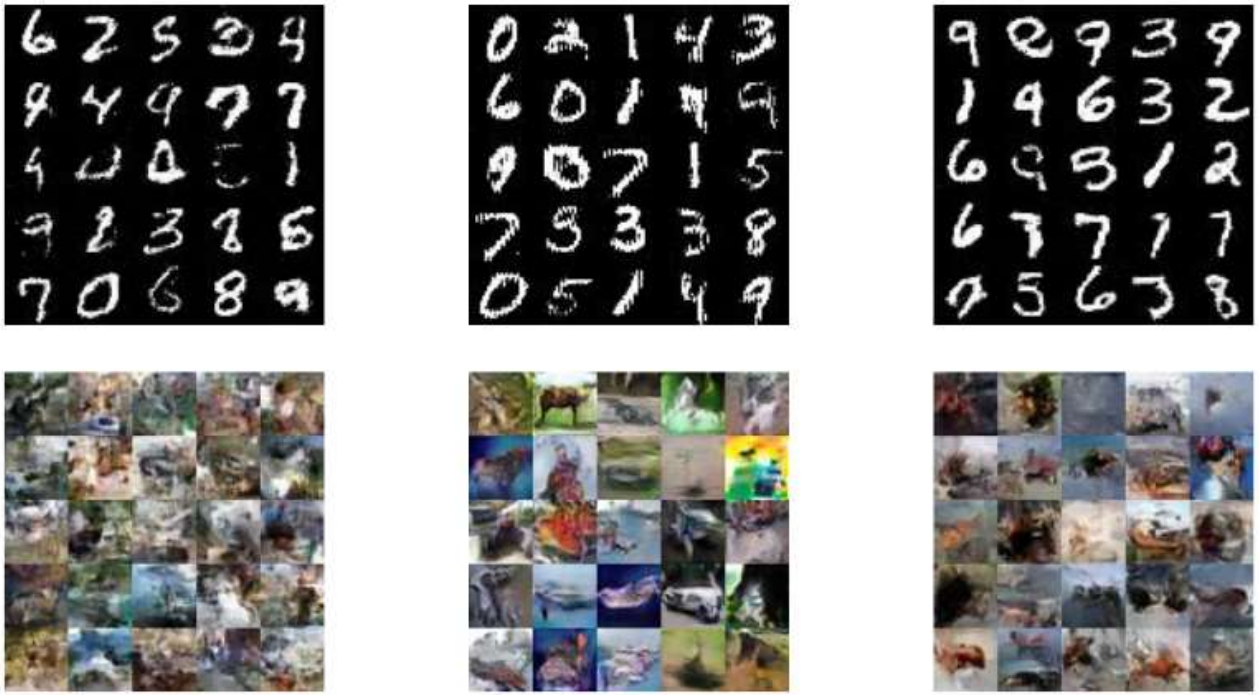


Toward using both maximum likelihood and adversarial training

1. Implicit models such as generative adversarial networks (GAN) often generate better samples compared to explicit models trained by maximum likelihood.

2. However, we know that the method based on maximum likelihood explicitly learn the probability density function of the input data.

3. To bridge this gap, we propose Flow-GANs, a generative adversarial network for which we can perform Exact likelihood evaluation, thus supporting both adversarial and maximum likelihood training.



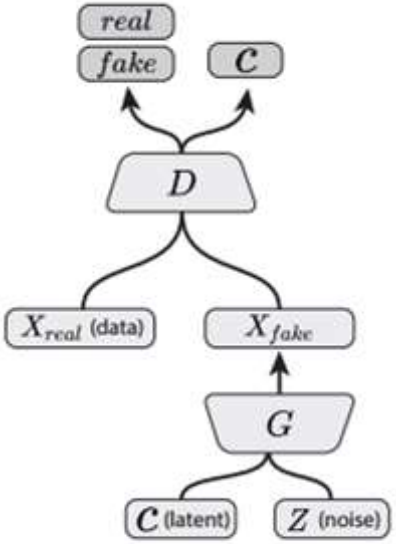
(a) MLE (b) ADV (c) Hybrid

Figure 1: Samples generated by Flow-GAN models with different objectives for MNIST (**top**) and CIFAR-10 (**bottom**).

(8) InfoGANs

Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
NIPS 2016

- **InfoGANs**
  InfoGAN is an information-theoretic extension to the GAN that is able to learn disentangled representations in an unsupervised manner. InfoGANs are used when your dataset is very complex, when you'd like to train a cGAN and the dataset is not labelled, and when you'd like to see the most important features of your images.



InfoGAN
(Chen, et al., 2016)

(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)

(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

(a) Azimuth (pose)

(b) Elevation

(c) Lighting

(d) Wide or Narrow

(a) Rotation (b) Width

Thank you