Deep Learning 1

2024-2025 – Pascal Mettes

Lecture 10

Auto-encoding and generation

Previous lecture

Lecture	Title
1	Intro and history of deep learning
3	Deep learning optimization I
5	Convolutional Neural Networks I
7	Attention
9	Self-supervised and vision-language learning
11	Deep learning for videos
13	The oddities of deep learning

Lecture	Title
2	Manually forward, automatically backward
4	Deep learning optimization II
6	Convolutional Neural Networks II
8	Graph Neural Networks
10	Auto-encoding and generation
12	Non-Euclidean deep learning
14	Q&A

This lecture

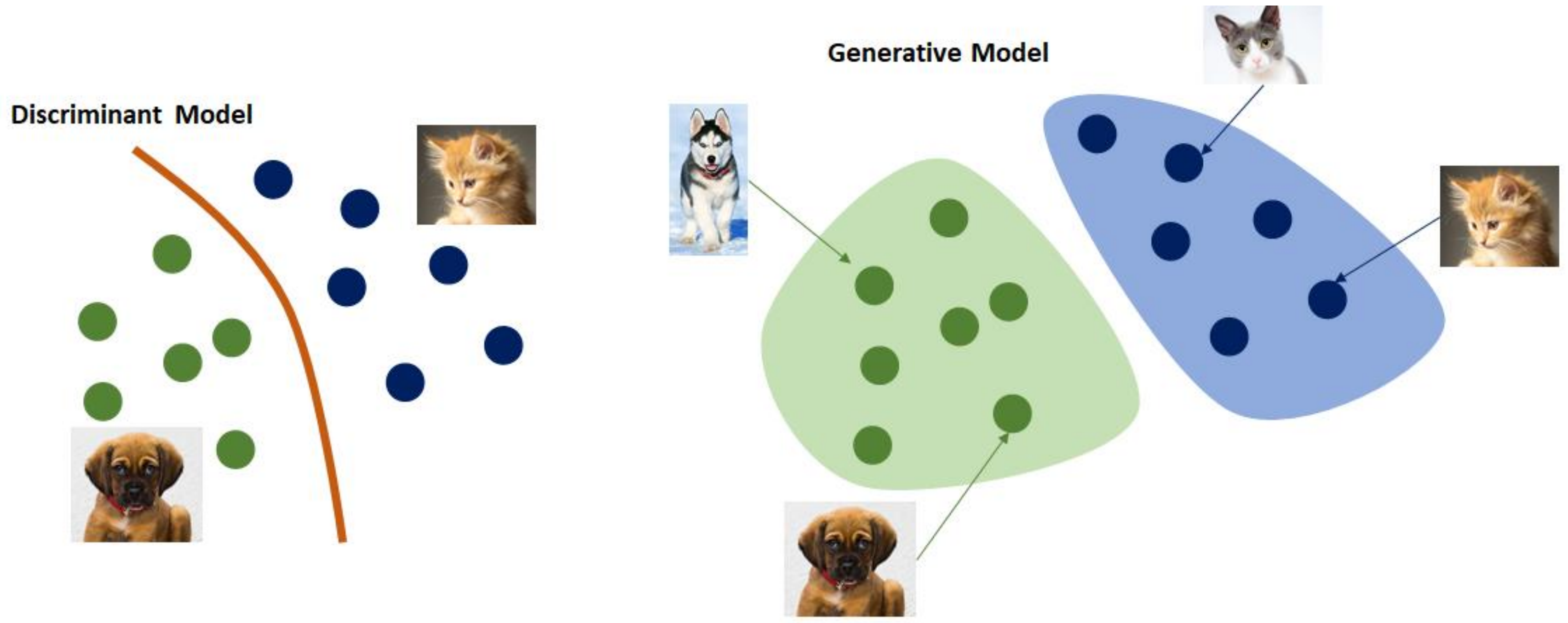
Generative learning 1: The variational era

Generative learning 2: The adversarial era

Generative learning 3: The diffusion era

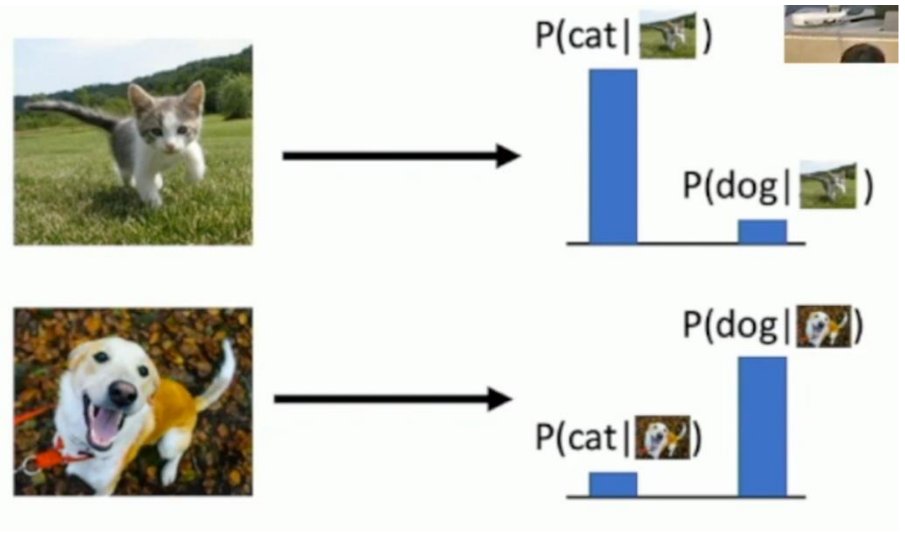
Generative versus discriminative learning

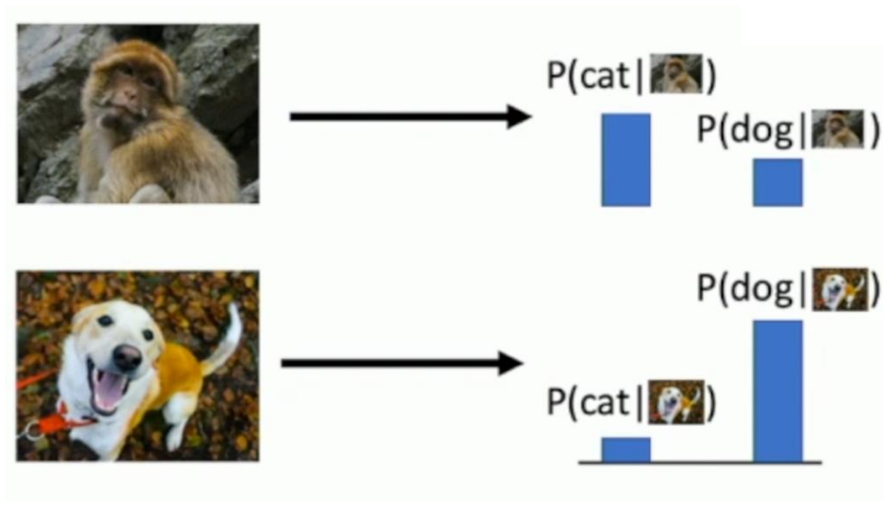
p(y|x) vs p(x)



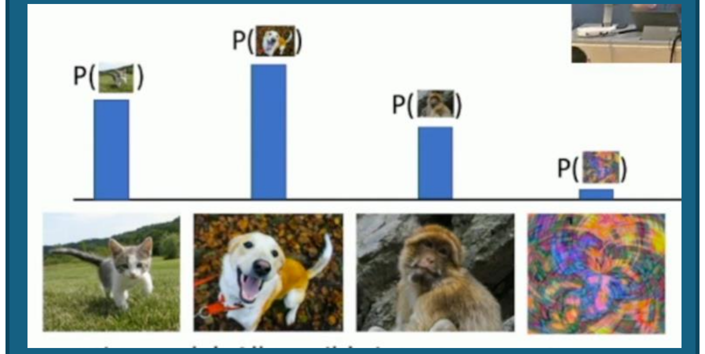
The importance of generative learning

Discriminative model: p is normalized for *outputs*, but not for inputs:





Generative model:
p is normalized for inputs



The importance of generative learning

Learn the distribution of data itself.

Physics: Model its laws, predict planet motions, etc.

Economics: Forecast financial patterns.

Idem for math, biology, geology, …

Make data and models more interpretable.

Generate new samples.

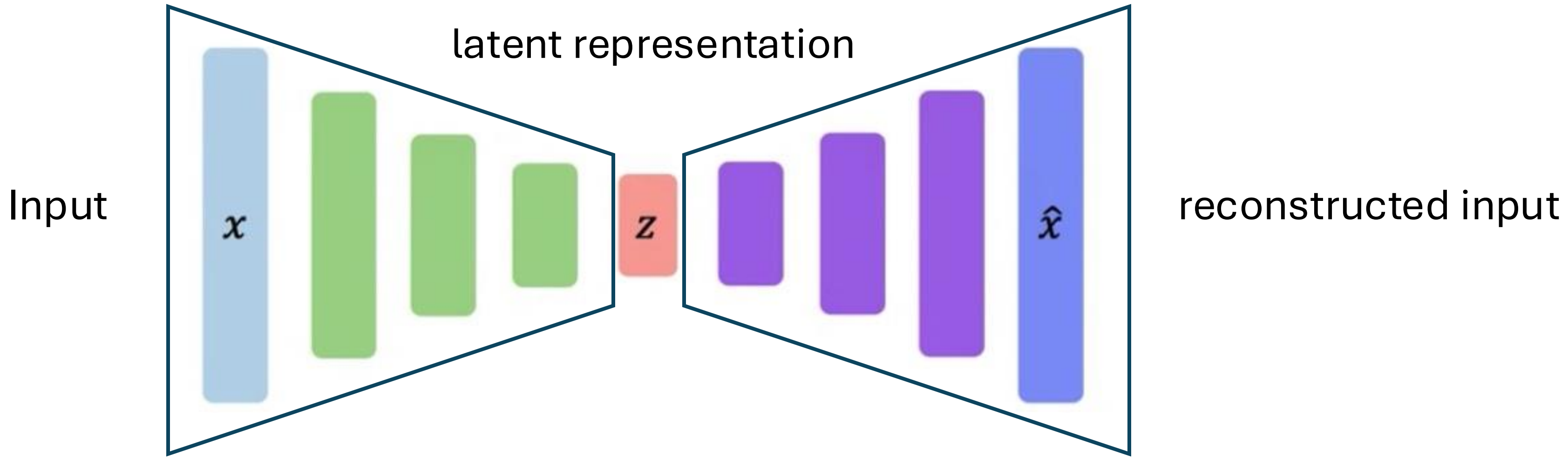
Enhance discriminative models.

Generative learning 1: The variational era

The autoencoder

The autoencoder is a feedforward network with a bottleneck layer.

Optimization is easy: minimize error between input and reconstructed output.



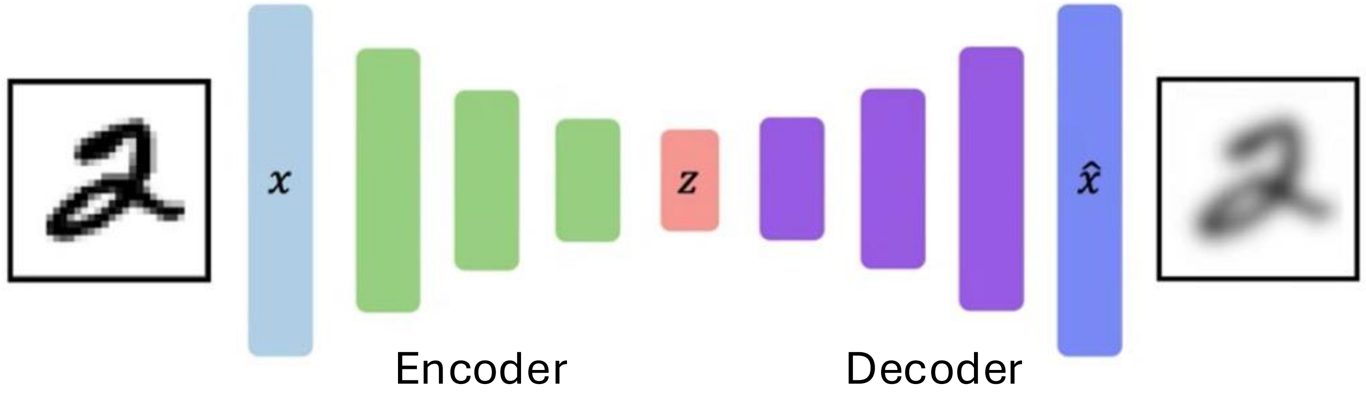
Structure of the autoencoder

Two parts: an encoder and a decover.

Encoder: Map from input to bottleneck.

Decoder: Map from bottleneck to output.

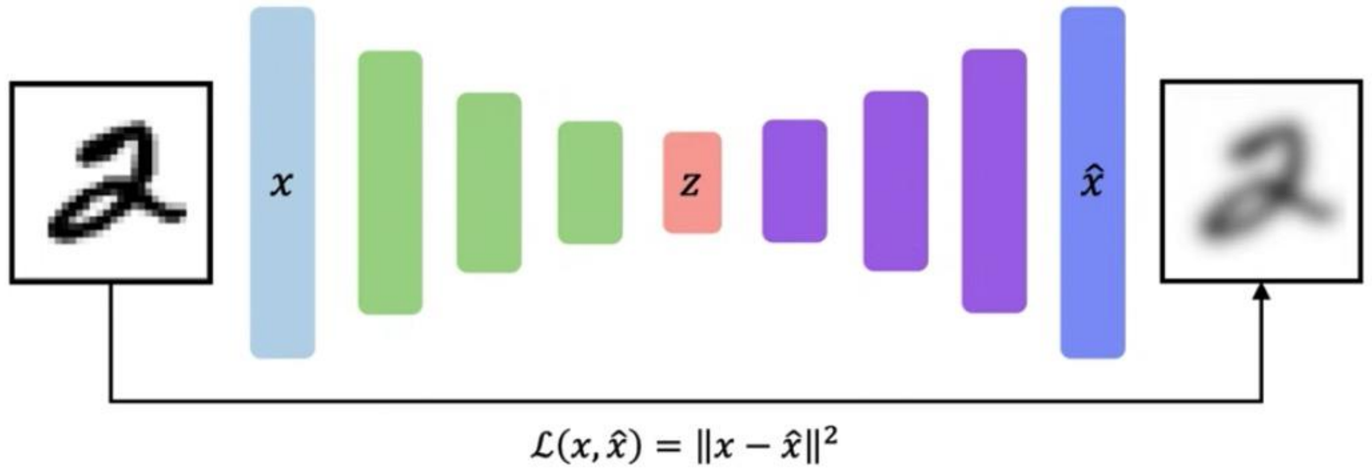
There are no restrictions on the architecture and choice of layers.



Training autoencoders

Simply minimize the error between output and input!

Unsupervised: no labels used to train parameters, hence the “auto” part.

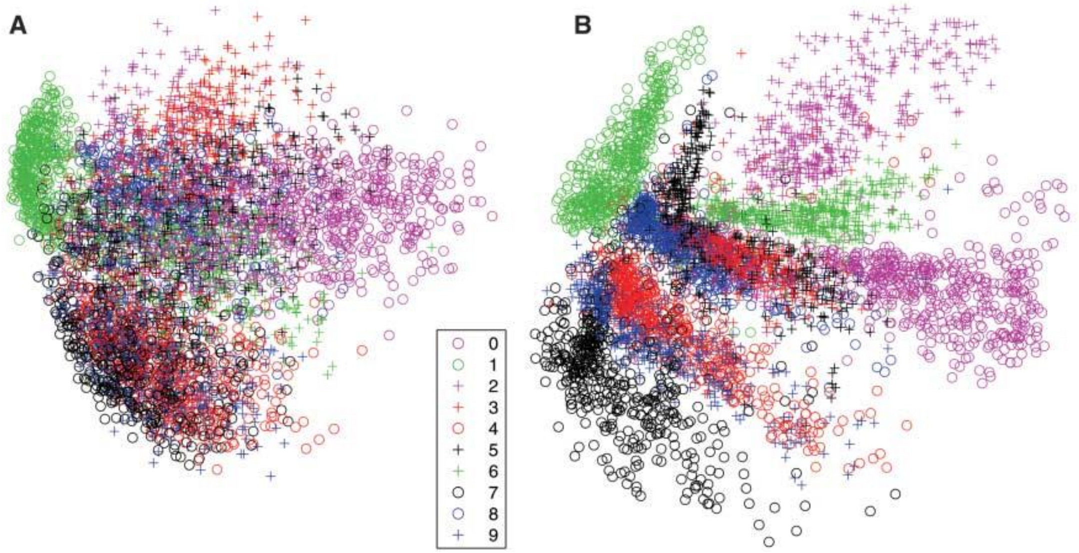


Purpose of autoencoders

Learning lower-dimensional feature representations.

Compression / invariance / redundancy removal.

Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).



PCA

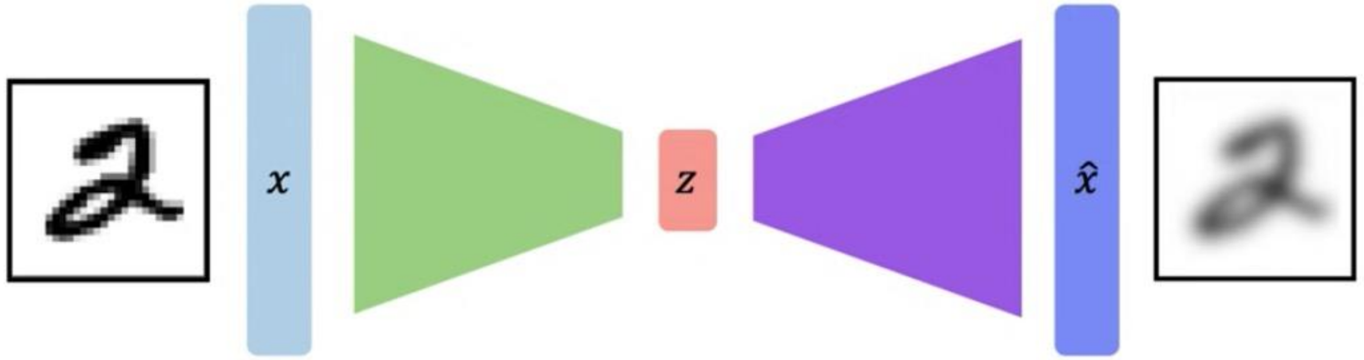
Autoencoder

Hinton and Salakhutdinov (2006)

Generative models from autoencoders?

We can technically generate samples by picking points in latent space *z*.

But we don't know how to sample from the latent space and we did not put any constraints on the latent space itself, hence not a practical setting.

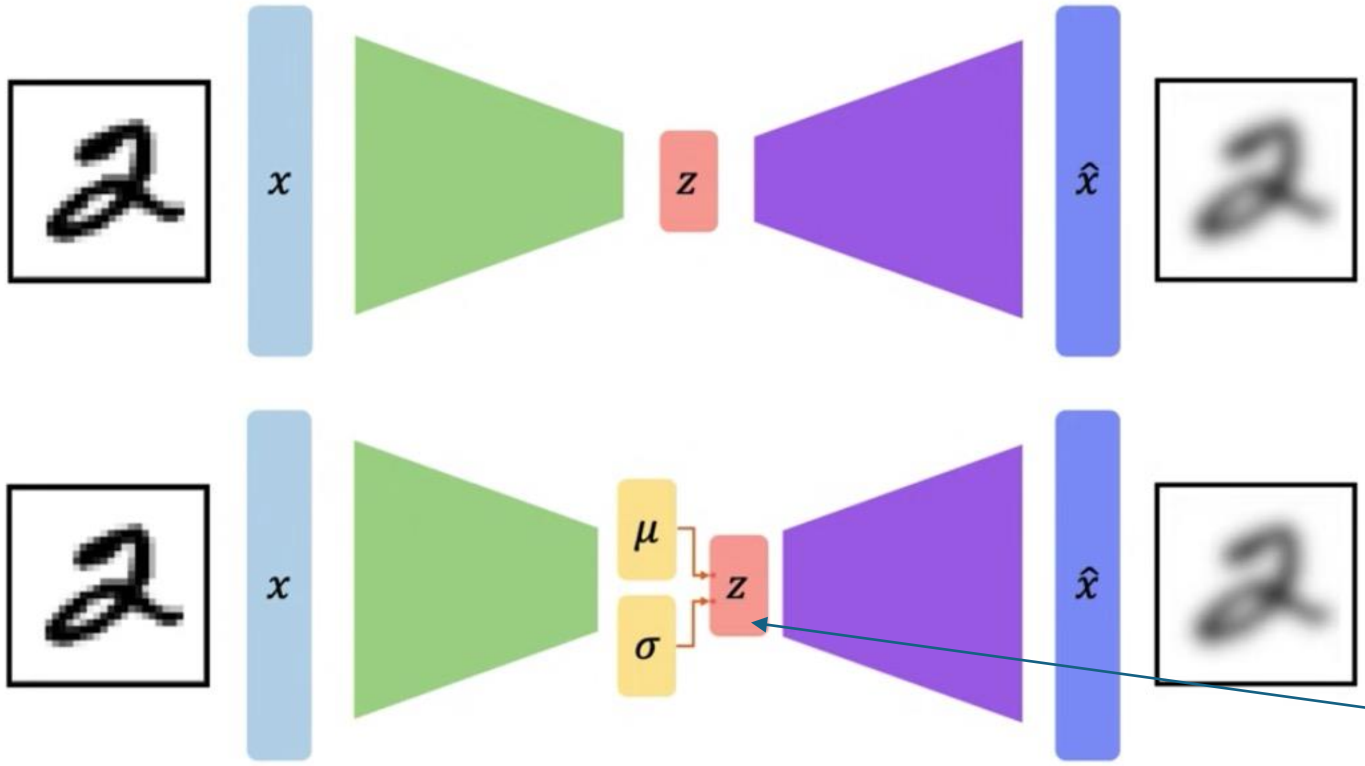


Variational autoencoder

Natural solution: constrain the latent space to follow a Gaussian distribution.

Autoencoder

VAE

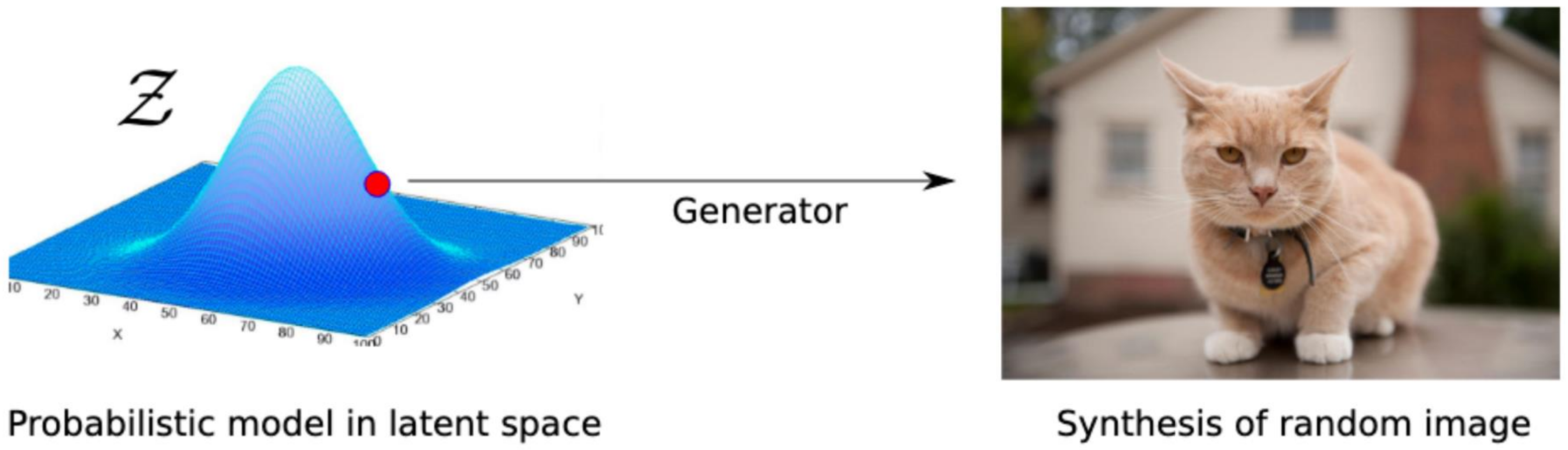


z is a latent variable

Main idea of VAEs

Train encoder and decoder with Gaussian constraint.

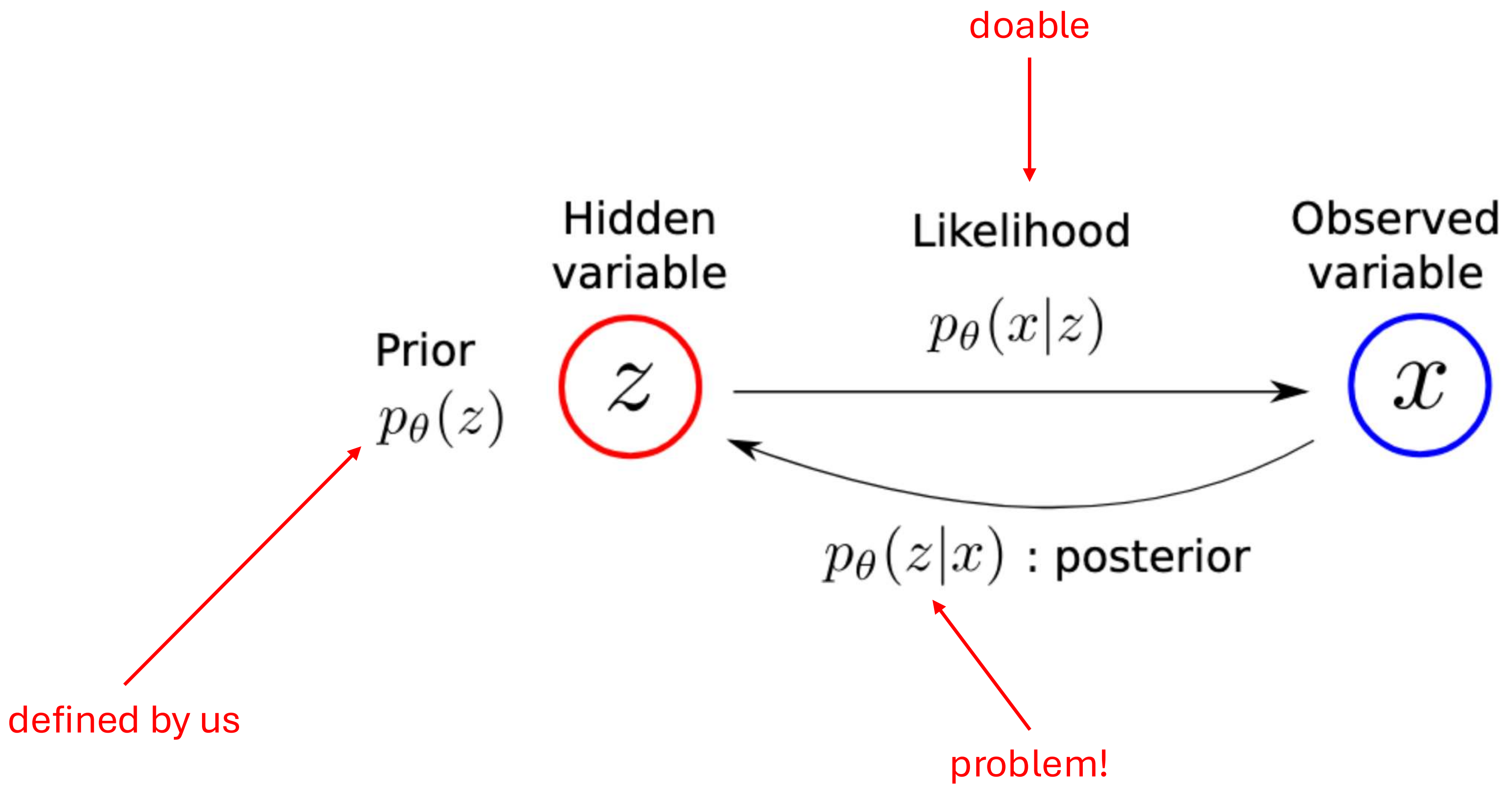
During inference, sample from the constrained space to produce new samples.



$\mathcal{Z}$
Generator
X
Y
Probabilistic model in latent space
Synthesis of random image

Illustrations courtesy of: Alasdair Newson, Telecom Paris

Probability density functions in a VAE



Analogies between Bayes and autoencoders

$$\begin{tikzcd}
x \arrow[dr, "p_\theta(z|x)"', "\text{Encoder}"] & & y \\
& z \arrow[ur, "\text{Decoder}", "p_\theta(x|z)"'] &
\end{tikzcd}$$

Encoder : posterior $p_\theta(z|x)$

Decoder : likelihood $p_\theta(x|z)$

Solving the inference problem

Distribution $p_\theta(z|x)$ is unknown, can we approximate it with a network?

Our goal: Approximate $p_\theta(z|x)$ with $q_\theta(z|x)$ as:

$$q_\phi^* = \arg\min_{q_\phi} KL\left(q_\phi(z|x) \mid\mid p_\theta(z|x)\right)$$

But we don't know $p_\theta(z|x)$, so we've gained nothing.

We need to optimize this objective in another way.

The ELBO

What will we set as our objective if the amrginal log-likelihood is intractible?

$$\underset{\substack{\uparrow\\ \text{intractable}}}{\log p_\theta(x)} = \underset{\substack{\uparrow\\ \text{our way in}}}{\text{ELBO}(q_\phi)} + KL(\underset{\substack{\uparrow\\ \text{unknown}}}{q_\phi(z|x)} \;\; || \; p_\theta(z|x)))$$

The KL divergence on the right is positive only, so ELBO is a lower bound.

Maximizing this ELBO minimizes the KL divergence on the right.

The Evidence Lower BOund

The ELBO consists of two parts, with both known!

$$\mathrm{ELBO}(q_\phi) = \mathbb{E}_{q_\phi}\left[\log(\underset{\uparrow}{p_\theta(x|z)})\right] - KL(q_\phi(z\underset{\uparrow}{|}x) \,||\, p_\theta(z))$$

This is our
reconstruction error.

This is our
Way to enforce the prior.

We can implement these two parts as losses to *maximize*.

VAEs summarized

We extend autoencoders with an alternative with a Gaussian as latent.

Direct optimization requires calculating the posterior, which is not feasible.

Instead we approximate it with a simpler (learned) function.

This function is optimized through the ELBO.

Additional depth and considerations beyond DL1

Optimization requires backpropagating through random variables.

This is not differentiable, solution: reparametrisation trick.

TL;DR: factor out the Gaussian.

For in-depth derivations, this resource by Yuge Shi recommended:

How I learned to stop worrying and write ELBO (and its gradients) in a billion ways

https://yugeten.github.io/posts/2020/06/elbo/

Generative learning 2:
The adversarial era

Explicit versus implicit density

With $p^*(x)$ being the real distribution:

- The model p_θ assigns high density to samples taken from the true distribution p^*:

$$\boldsymbol{x} \sim p^*(\boldsymbol{x}) \implies p_\theta(\boldsymbol{x}) \text{ is "high".}$$

Explicit density

- Samples taken from the model p_θ behave similarly to real samples from p^*:

$$\boldsymbol{x} \sim p_\theta(\boldsymbol{x}) \implies p^*(\boldsymbol{x}) \text{ is "high".}$$

Implicit density

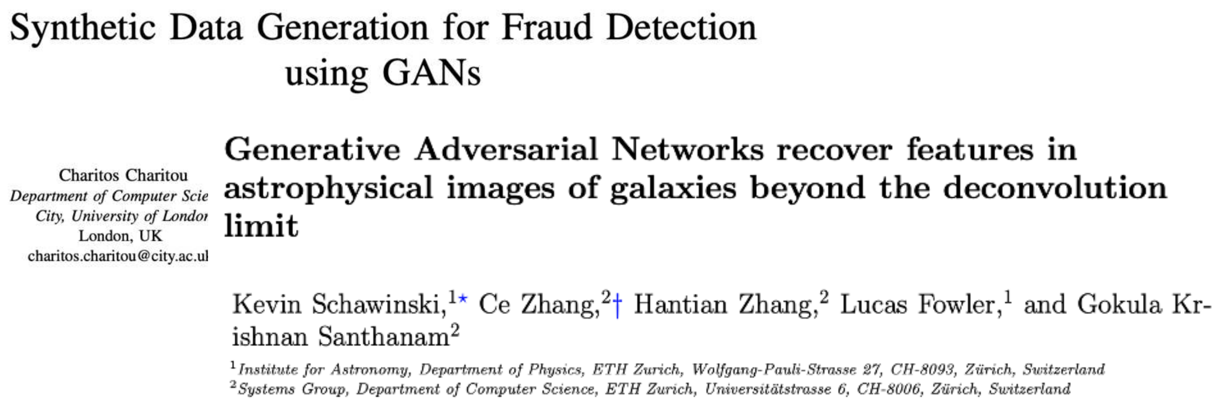
Why learn implicit densities?

Learning explicit densities is hard.

Often in practice, we care only about how something looks.

This is the idea behind the Generative Adversarial Network (GAN).

GAN: High quality generation through game theory



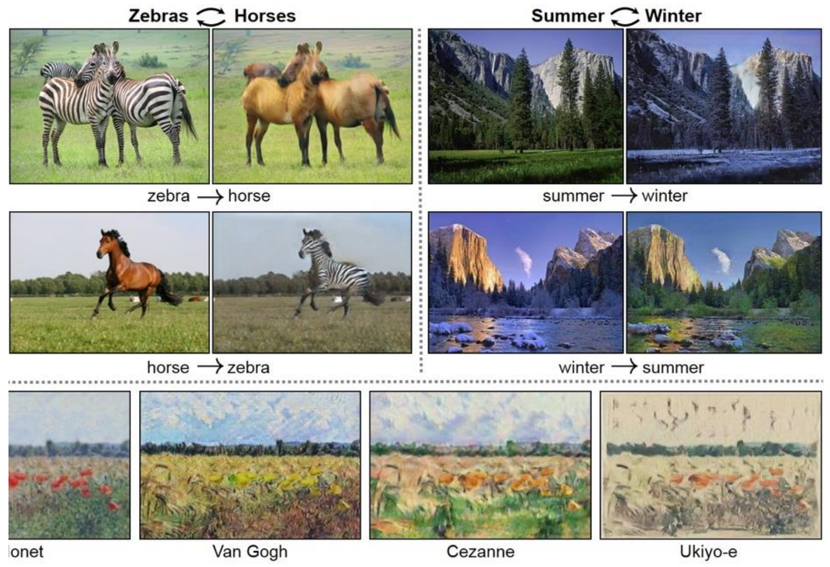
Synthetic Data Generation for Fraud Detection using GANs

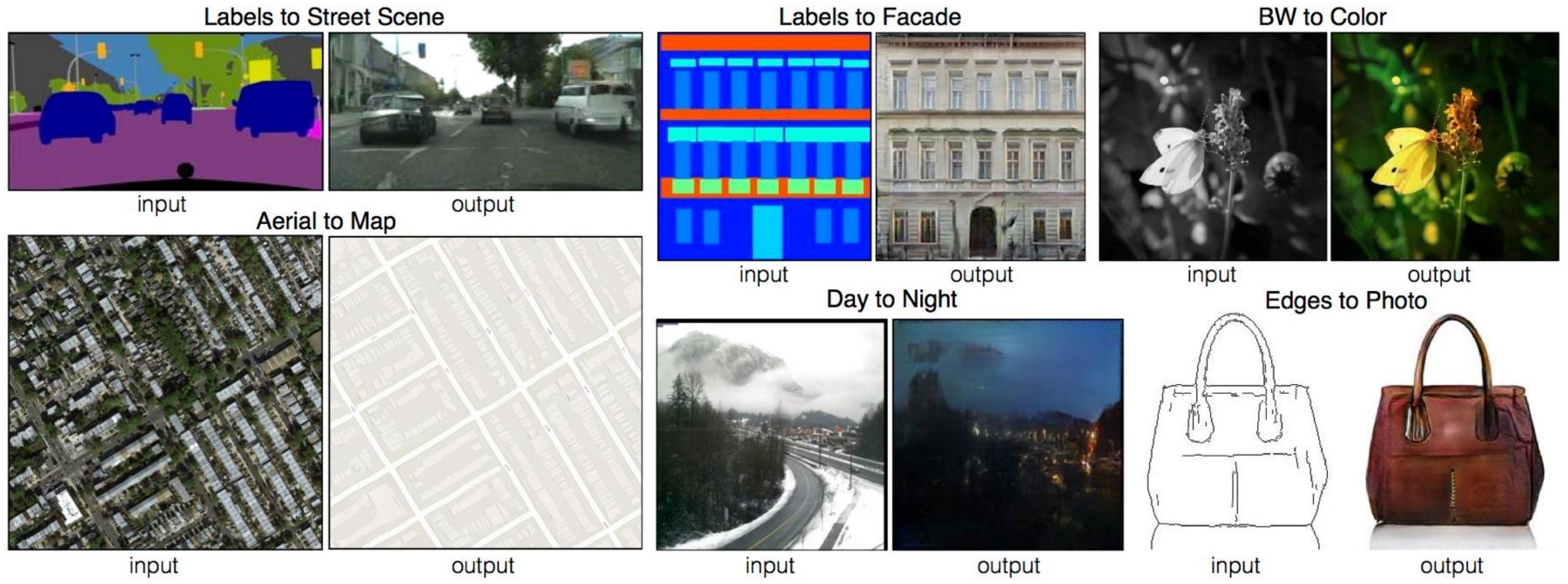
Charitos Charitou
Department of Computer Scie
City, University of Londor
London, UK
charitos.charitou@city.ac.ul

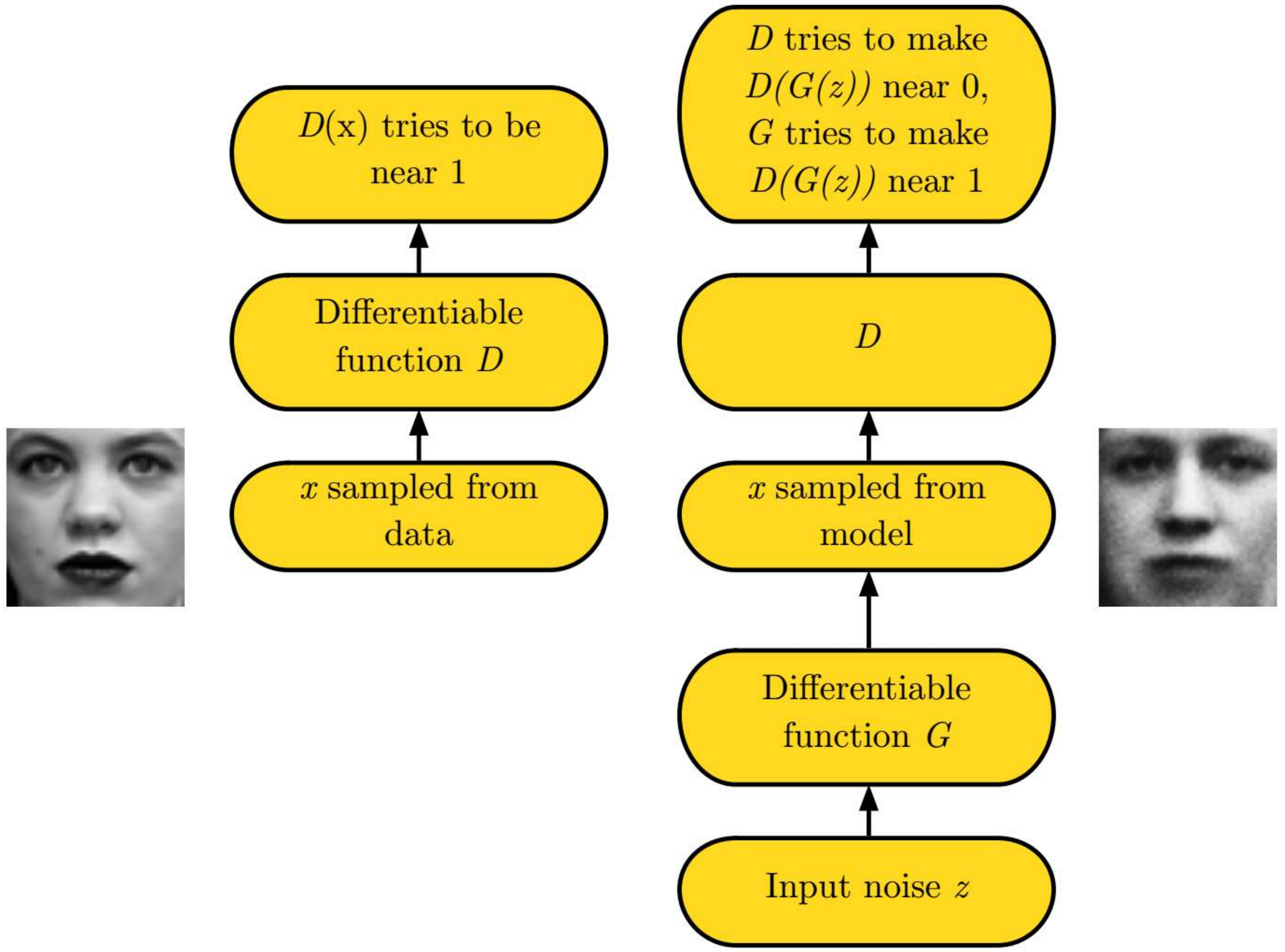
Generative Adversarial Networks recover features in astrophysical images of galaxies beyond the deconvolution limit

Kevin Schawinski,[1*] Ce Zhang,[2†] Hantian Zhang,[2] Lucas Fowler,[1] and Gokula Krishnan Santhanam[2]

[1]Institute for Astronomy, Department of Physics, ETH Zurich, Wolfgang-Pauli-Strasse 27, CH-8093, Zürich, Switzerland
[2]Systems Group, Department of Computer Science, ETH Zurich, Universitätstrasse 6, CH-8006, Zürich, Switzerland






$D(\mathrm{x})$ tries to be near 1
Differentiable function D
x sampled from data
D tries to make $D(G(z))$ near 0, G tries to make $D(G(z))$ near 1
D
x sampled from model
Differentiable function G
Input noise z

NeurIPS 2016 Tutorial: Generative Adversarial Networks

What is a GAN?

Generative

You can sample novel inputs and “create” what never existed.

Adversarial

Train two models: a generator (creator) and a discriminator (evaluator).

Network

Implemented as a deep network and learned with backprop.

Intuition behind GANs

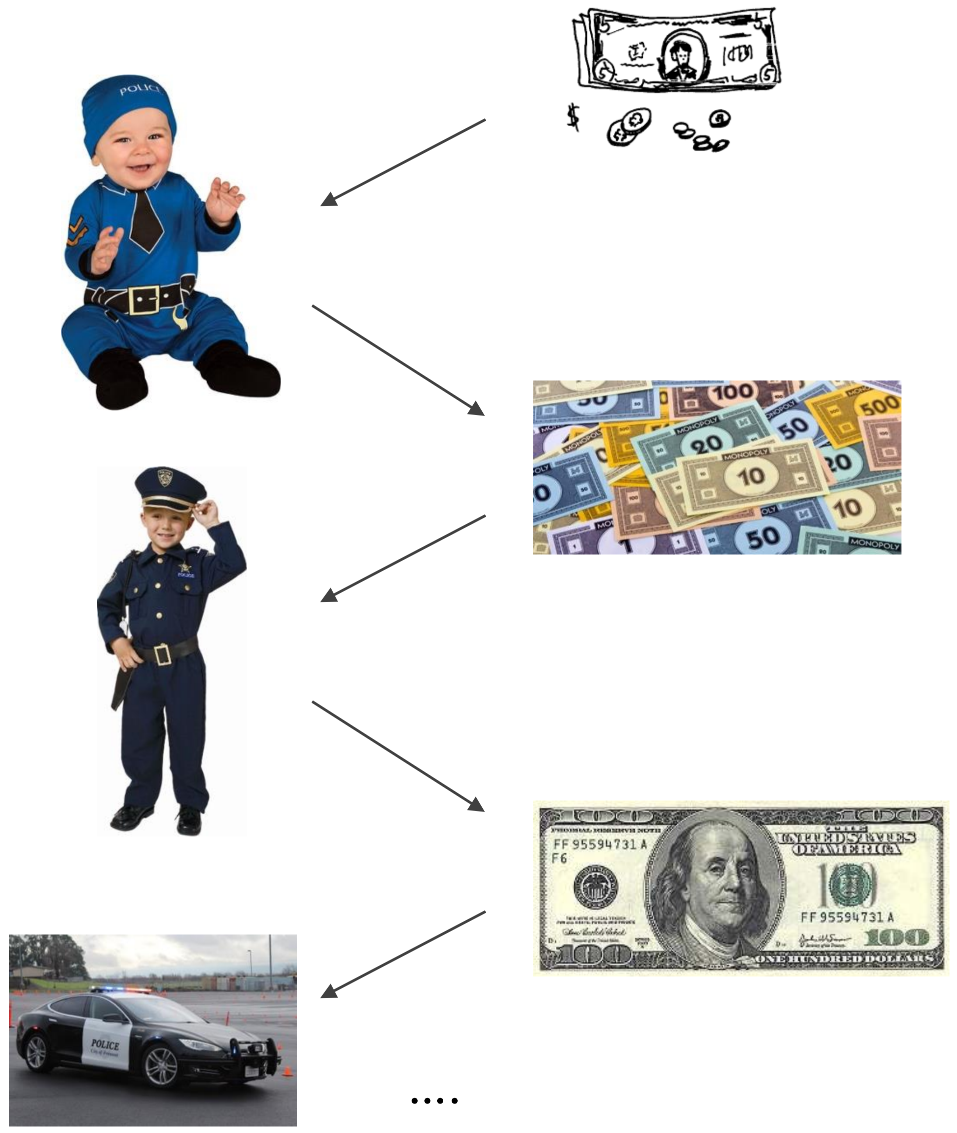
Police: wants to detect fake money as reliably as possible.

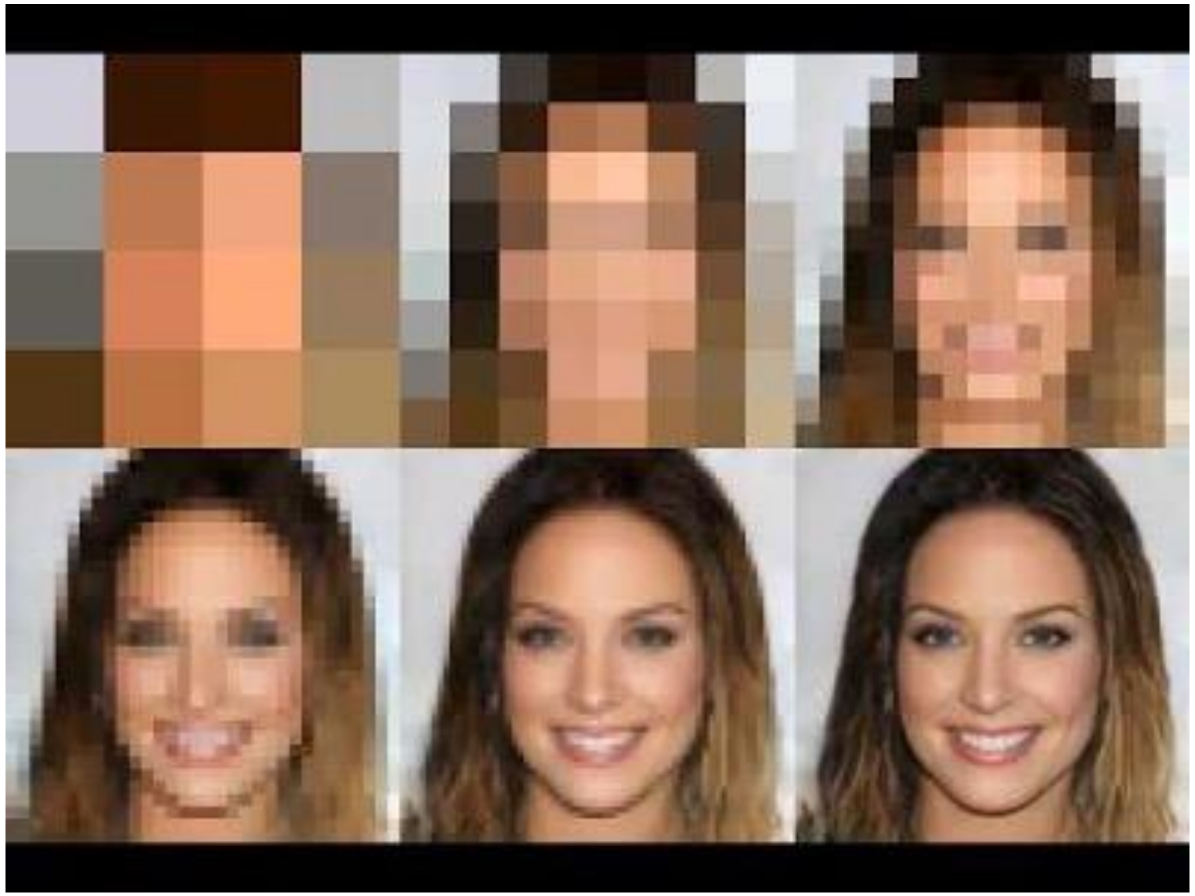
Counterfeiter: wants to make as realistic fake money as possible.

At beginning: both have no clue.

The police forces the counterfeiter to get better as it compares it to real money (and vice versa).

Convergent solution ~ Nash equilibrium



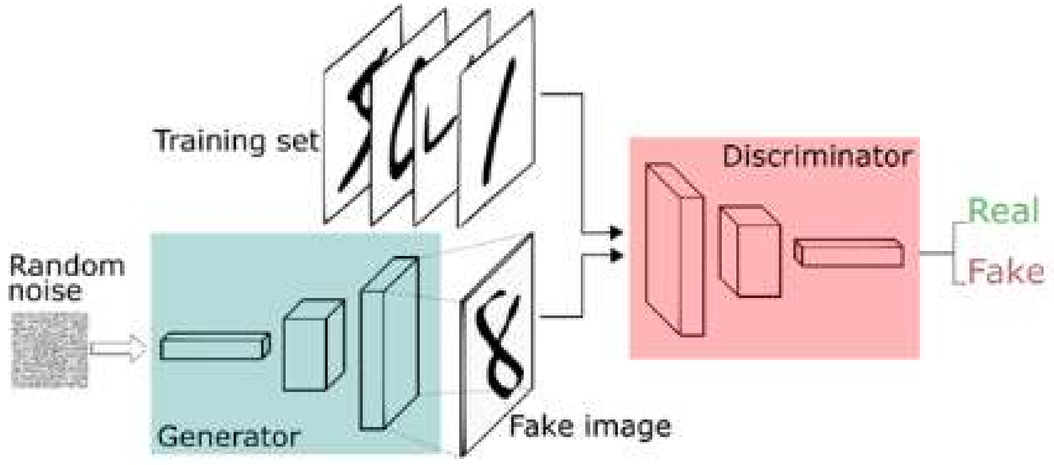


GAN architecture

The GAN comprises two neural networks:

Generator network $\boldsymbol{x} = G(\boldsymbol{z}; \boldsymbol{\theta}_{\mathrm{G}})$

Discriminator network $\mathrm{y} = D(\boldsymbol{x}; \boldsymbol{\theta}_{\mathrm{D}}) = \begin{cases} +1, \text{if } \mathbf{x} \text{ is predicted 'real'} \\ 0, \text{if } \boldsymbol{x} \text{ is predicted 'fake'} \end{cases}$



GAN generator $\boldsymbol{x} = G(\boldsymbol{z}; \boldsymbol{\theta}_{\mathrm{G}})$

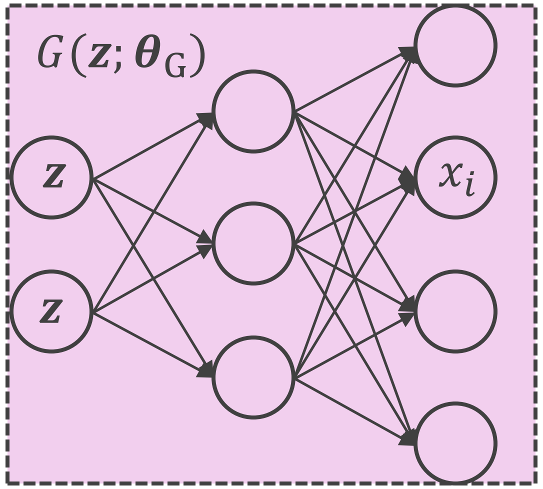
Any differentiable neural network.

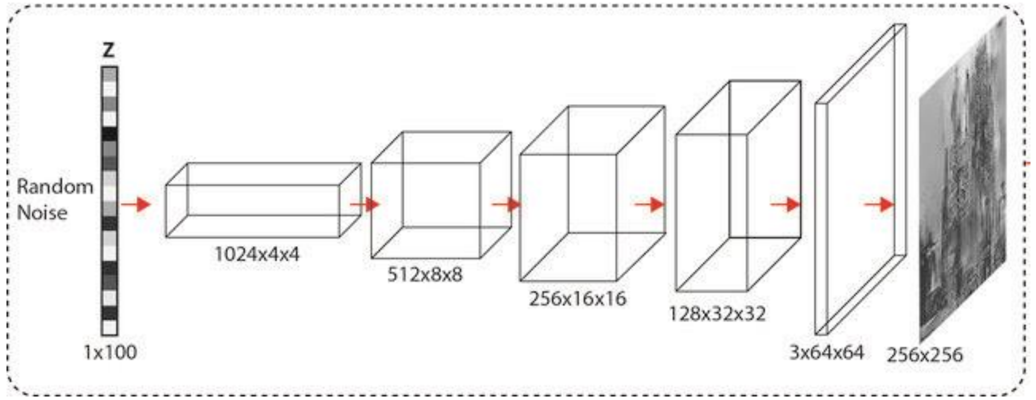
No invertibility requirement → More flexible modelling.

Starts with some random, typically lower dimensional input z.

Various density functions for the noise variable $\boldsymbol{z}$.

$\boldsymbol{z} \sim \mathcal{N}(0,1)$ or
$\boldsymbol{z} \sim \mathrm{Uniform}(0,1)$
…





GAN discriminator $\boldsymbol{y} = D(\boldsymbol{x}; \boldsymbol{\theta}_{\mathrm{D}})$

Any differentiable neural network.

Receives as inputs:

either real images from the training set

or generated images from the generator

usually a mix of both in mini-batches

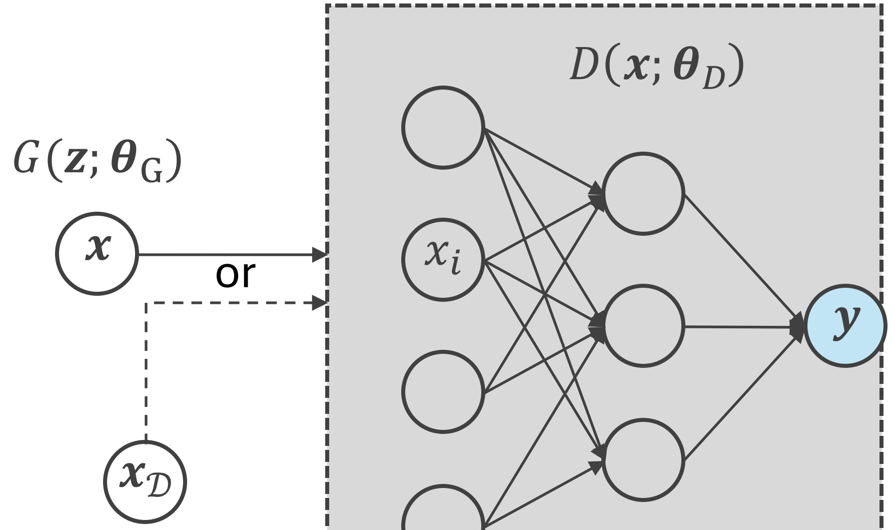
Must recognize the real from the fake inputs.

The discriminator loss:

$$J_D(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G) = \frac{1}{2}\mathrm{BCE}(Data, 1) + \frac{1}{2}\mathrm{BCE}(fake, 0)$$

$$= -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D(x)] - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}}\left[\log\left(1 - D\big(G(\boldsymbol{z})\big)\right)\right]$$

$$= -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D(x)] - \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{generator}}}[\log(1 - D(\boldsymbol{x}))]$$


$D(\boldsymbol{x}; \boldsymbol{\theta}_D)$
$G(\boldsymbol{z}; \boldsymbol{\theta}_{\mathrm{G}})$
$\boldsymbol{x}$
or
x_i
$\boldsymbol{y}$
$\boldsymbol{x}_{\mathcal{D}}$

Binary Cross Entropy loss:

$$l_n = -w_n\left[y_n \cdot \log x_n + (1 - y_n)\cdot \log(1 - x_n)\right],$$

GAN implementation

The discriminator is just a standard neural network.

The generator looks like an inverse discriminator (or like a decoder).

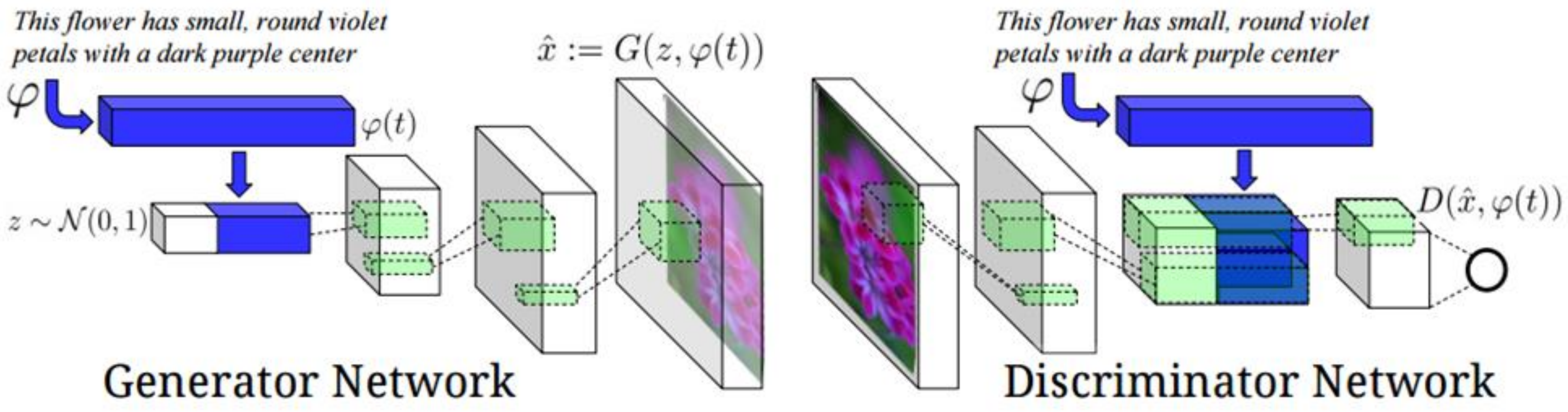

This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
Generator Network
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Discriminator Network

Figure 2. Our text-conditional convolutional GAN architecture. Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

Network Architecture

How to train a GAN

Given a generated image, we do not have its “equivalent” image in our batch.

Generative generates some random images independent of comparison batch.

How can we get meaningful gradients?

The minimax loss

Simplest case: Generator loss is negative discriminator loss (“zero-sum game”).

$$J_G = -J_D$$

The lower the generator loss, the higher the discriminator loss

Symmetric definitions

Our learning objective then becomes

$$V = -J_D(\boldsymbol{\theta}_{\mathrm{D}}, \boldsymbol{\theta}_{\mathrm{G}})$$

$D(\boldsymbol{x}) = 1 \rightarrow$ The discriminator believes that x is a true image

$D(G(\boldsymbol{z})) = 1 \rightarrow$ The discriminator believes that $G(z)$ is a true image

So overall loss:

$$\text{Minimize}_G \, \text{Maximize}_D \, J_D$$

Heuristic non-saturating loss

Discriminator loss (maximize likelihood of correctly labelling real/fake data).

$$J_D = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}} \log D(\boldsymbol{x}) \; -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}} \log(1 - D(G(\boldsymbol{z})))$$

Generator loss (maximize likelihood of discriminator being wrong).

$$J_G = -\frac{1}{2}\mathbb{E}_{z\sim p_z} \log(D(G(\boldsymbol{z}))$$

Generator learns even when discriminator is too good on real images.

Training GANs is a pain

1. Vanishing gradients
2. Batchnorm
3. Convergence
4. Mode collapse

1. Vanishing gradients

If the discriminator is quite bad
→ the generator gets confused
→ no reasonable generator gradients

If the discriminator is near perfect
→ gradients go to 0, no learning anymore

Bad if early in the training

Easier to train the discriminator than generator

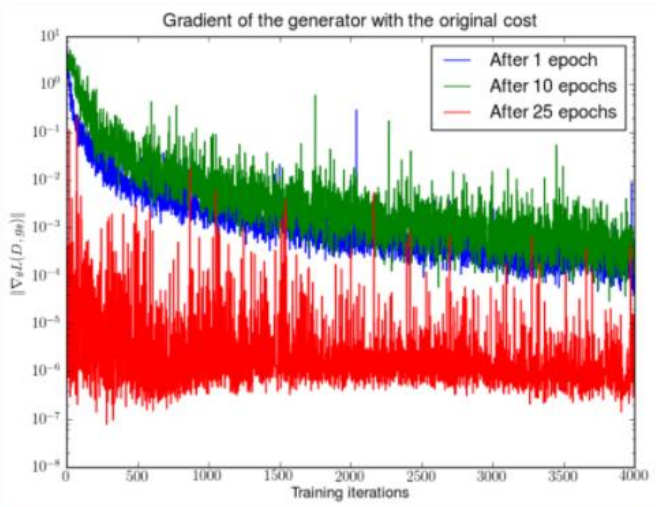


*Fig. 5. First, a DCGAN is trained for 1, 10 and 25 epochs. Then, with the **generator fixed**, a discriminator is trained from scratch and measure the gradients with the original cost function. We see the gradient norms **decay quickly** (in log scale), in the best case 5 orders of magnitude after 4000 discriminator iterations. (Image source: Arjovsky and Bottou, 2017)*

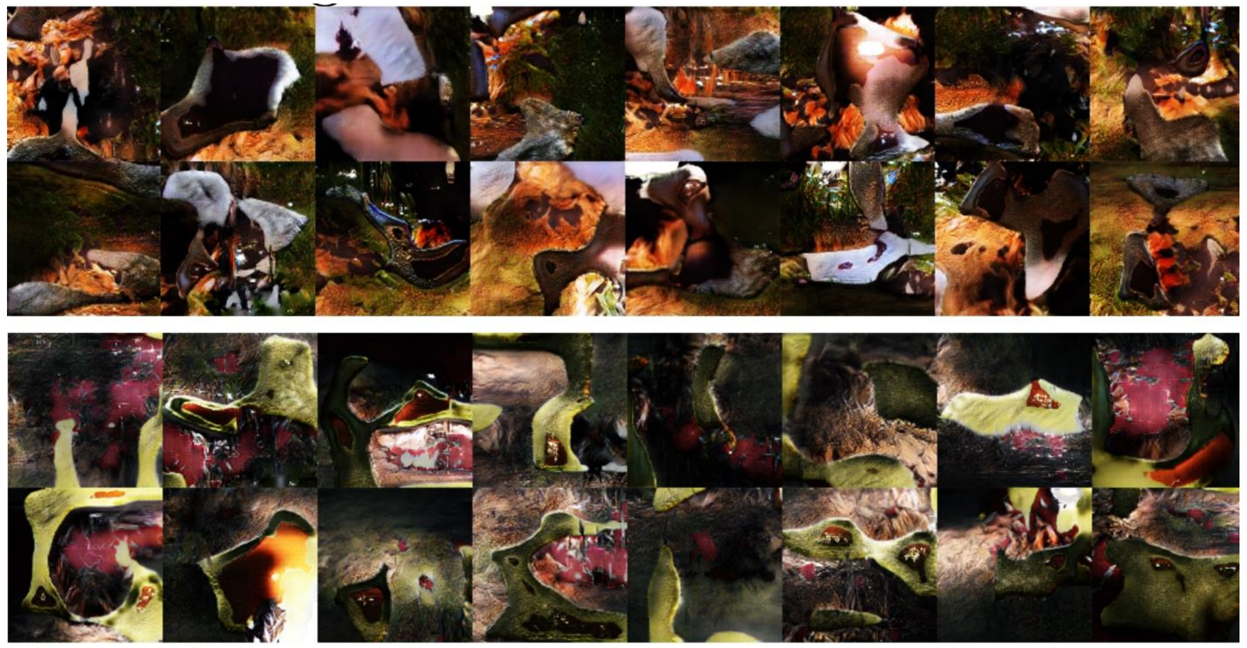
2. Batchnorm

Batch-normalization causes strong intra-batch correlation.

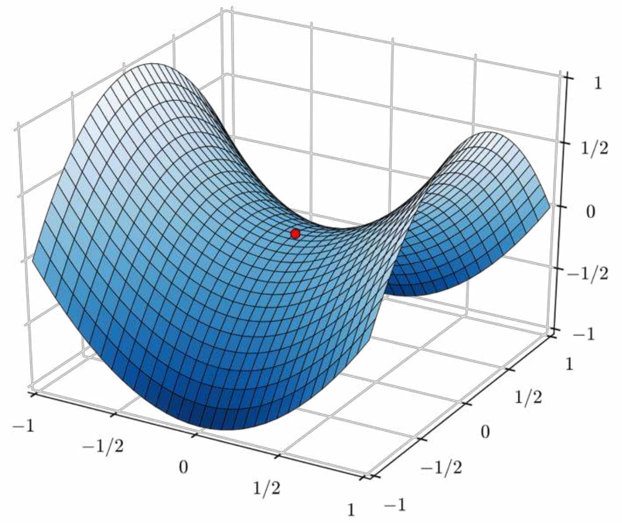
Generations look smooth but awkward.

Easiest solution: drop batchnorm.

(See e.g. StyleGAN)



3. Convergence



Optimization is tricky and unstable.

- finding a saddle point does not imply a global minimum
- A saddle point is also sensitive to disturbances

An equilibrium might not even be reached (models can train for weeks).

Mode-collapse is the most severe form of non-convergence.

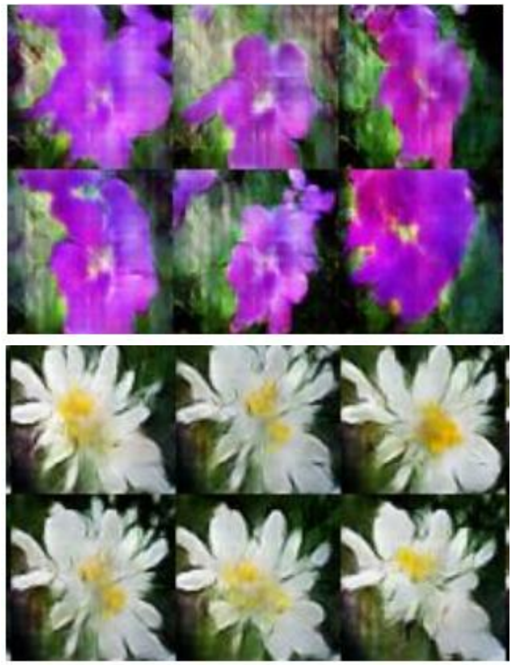
4. Mode collapse

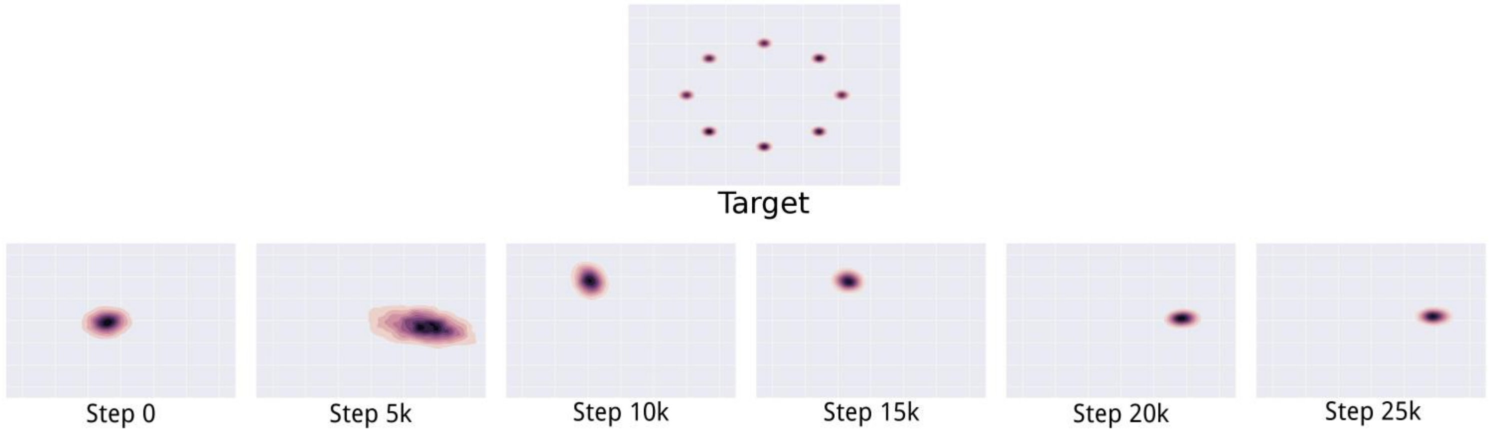
Discriminator converges to the correct distribution

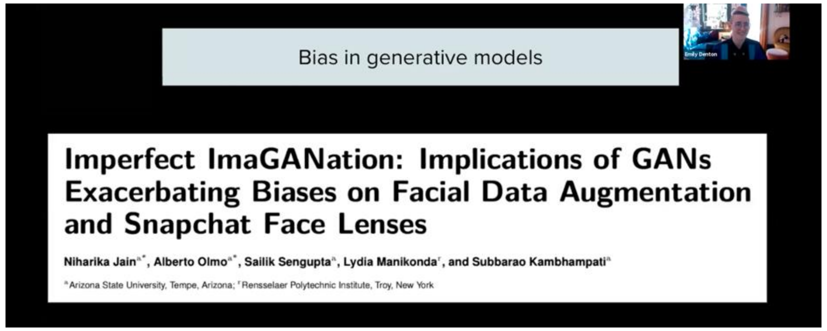
Generator however places all mass in the most likely point

All other modes are ignored (underestimating variance)

Low diversity in generating samples







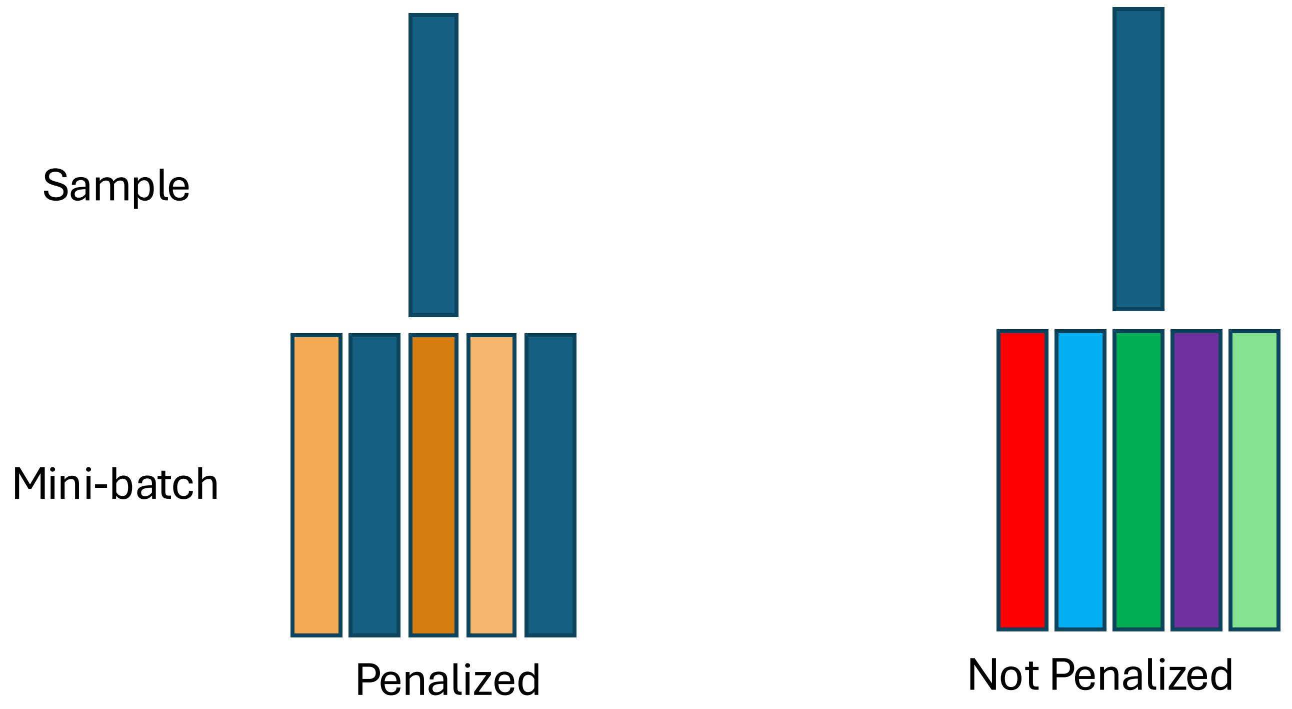
Imperfect ImaGANation: Implications of GANs Exacerbating Biases on Facial Data Augmentation and Snapchat Face Lenses

Niharika Jain[a*], Alberto Olmo[a*], Sailik Sengupta[a], Lydia Manikonda[r], and Subbarao Kambhampati[a]

[a] Arizona State University, Tempe, Arizona; [r] Rensselaer Polytechnic Institute, Troy, New York

Addressing mode collapse

A simple trick: compare each sample to others in a mini-batch and add a penalty for high similarities witihn a mini-batch.

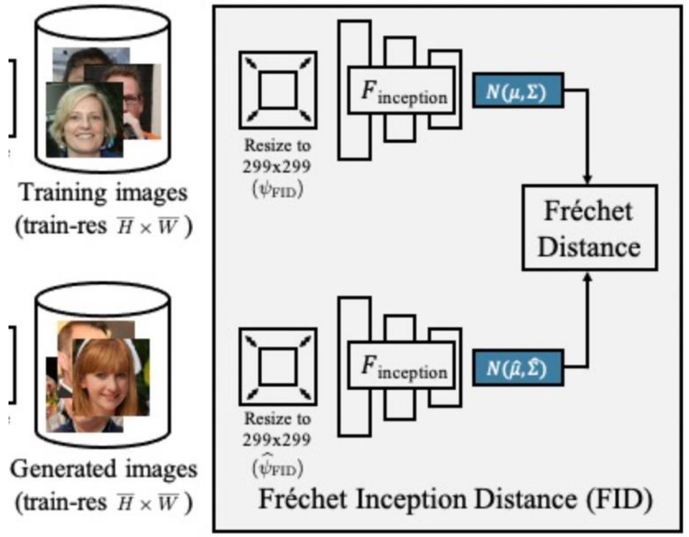


How do we actually evaluate GANs?

General problem for generative learning, how to know if your model is better?

There are image quality measures like FID, but these are naturally limited.

Best estimate: ask people to rank manually.


Training images
(train-res $\overline{H} \times \overline{W}$)
Generated images
(train-res $\overline{H} \times \overline{W}$)
Resize to 299x299 (ψ_{FID})
$F_{\text{inception}}$
$N(\mu, \Sigma)$
Resize to 299x299 ($\widehat{\psi}_{\text{FID}}$)
$F_{\text{inception}}$
$N(\hat{\mu}, \hat{\Sigma})$
Fréchet Distance
Fréchet Inception Distance (FID)

Slides inspired by Jumin Lee and the CVPR’22 Tutorial on Diffusion Models

Generative learning 3: The diffusion era

Back to autoencoders and Gaussians

VAEs try to directly learn a mapping form inputs to a Gaussian.

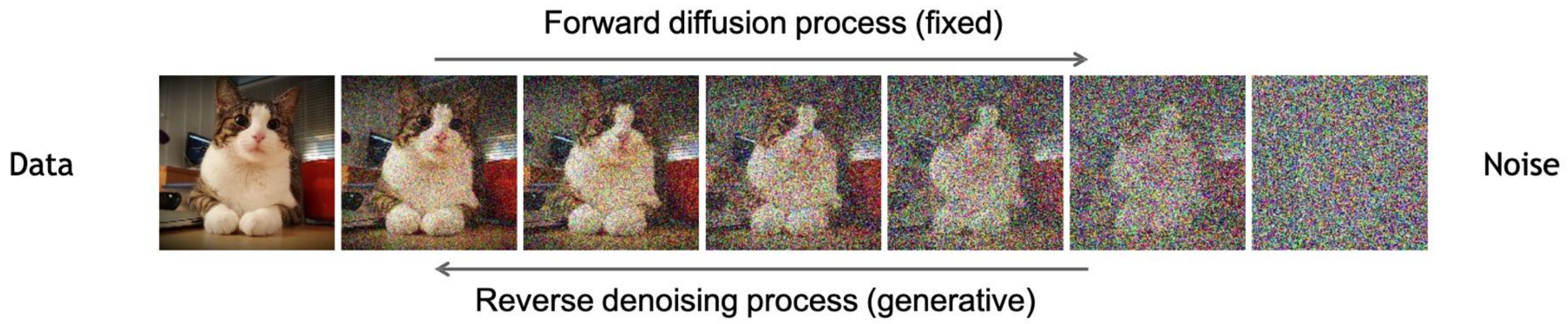
This seems like a big leap, what if we do this in a gradual manner?

I’ll give a short primer here, diffusion models come back in CV2 and FoMo.

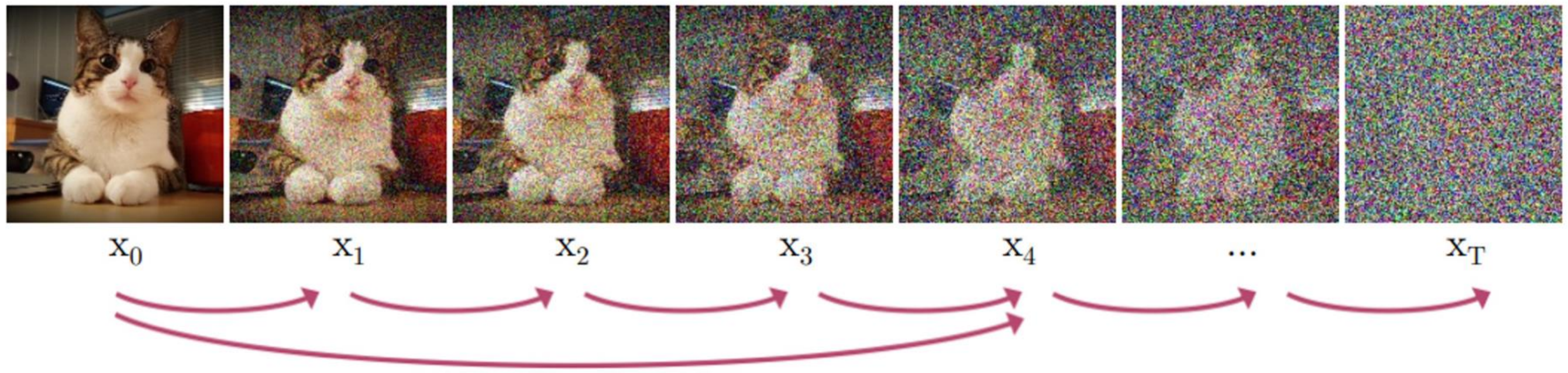
Denoising diffusion models

Forward process: take input can gradually add noise (encoder).

Reverse process: learn to generate data from noise (decoder).


Forward diffusion process (fixed)
Data
Noise
Reverse denoising process (generative)

Forward process of diffusion models



Scales down the input and adds noise.

Markov Property

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

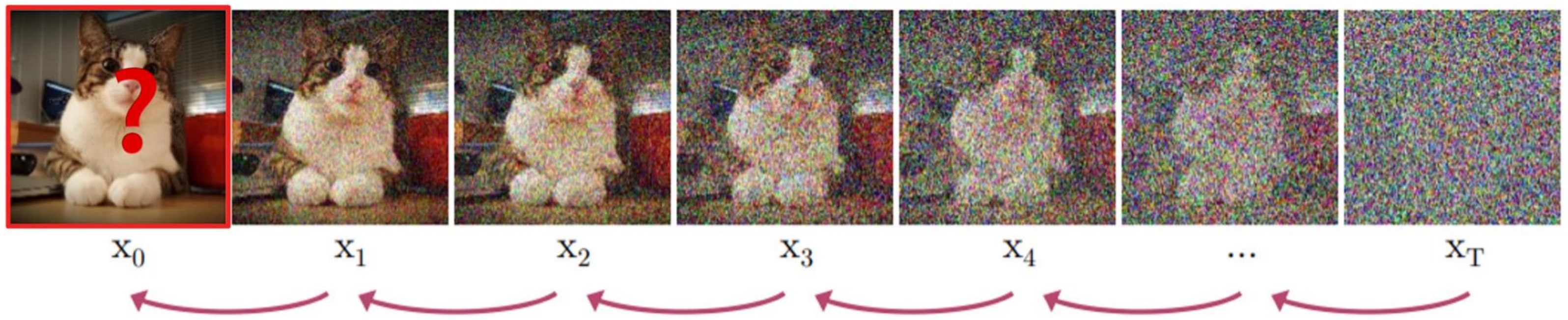
$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I})$$ ← Diffusion Kernel

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\,\epsilon \quad \text{where } \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\alpha_t := 1 - \beta_t \text{ and } \bar{\alpha}_t := \prod_{s=0}^{t} \alpha_s$$

Variance schedule.

Reverse process of diffusion models

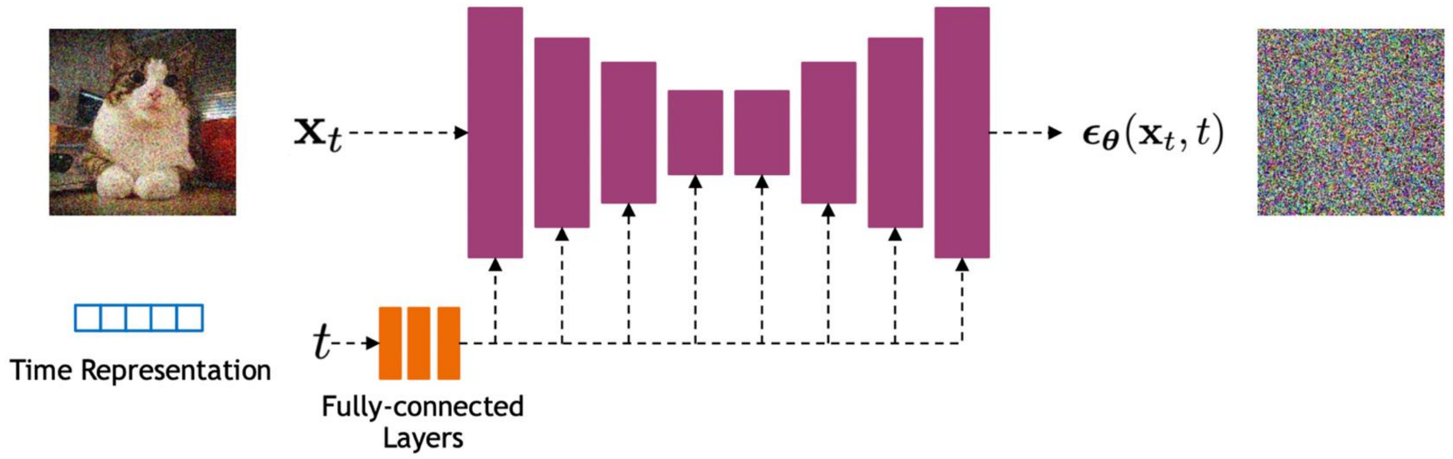


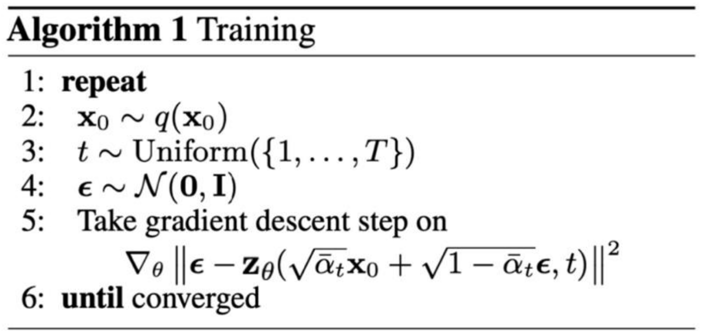
$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I})$$

$$\Big\downarrow \text{Model}$$

$$p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \boxed{\mu_\theta(x_t, t)}, \Sigma_\theta(x_t, t))$$

Training diffusion models

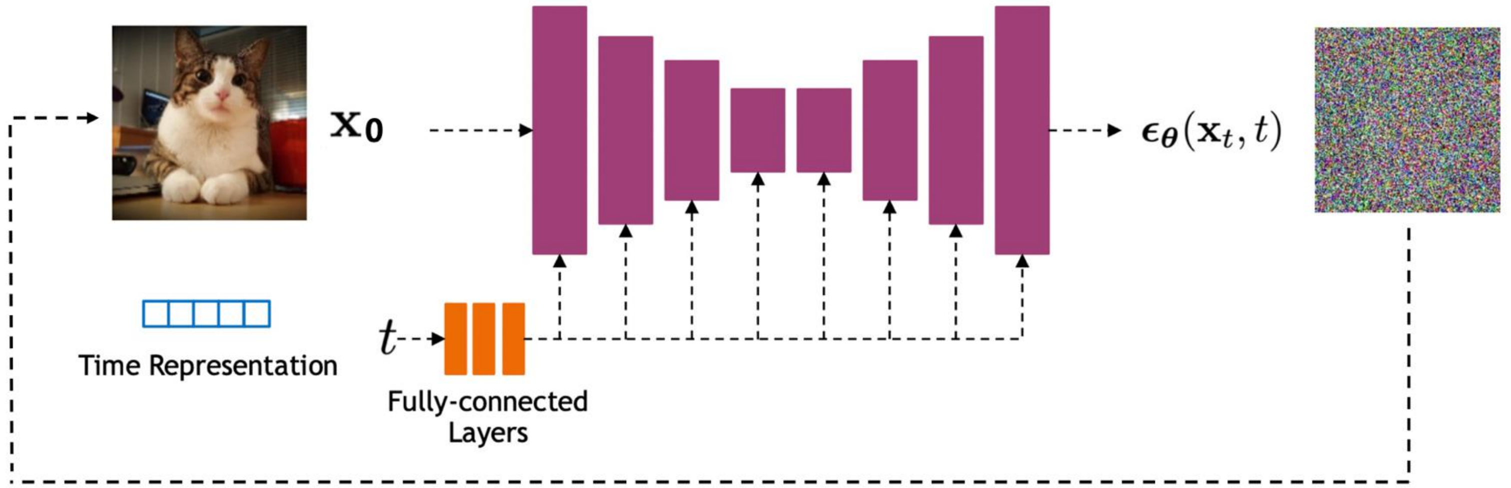




Algorithm 1 Training

```
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\| \boldsymbol{\epsilon} - \mathbf{z}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: until converged
```

Sampling form diffusion models



Algorithm 2 Sampling

```
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: for $t = T, \ldots, 1$ do
3:   $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4:   $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\mathbf{z}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: end for
6: return $\mathbf{x}_0$
```

Impact of diffusion models

Impact of diffusion models

Key behind state-of-the-art generative models:
Midjourney, Imagen, Sora, Dall-E, Stable Diffusion

Intuitive process, as we operate directly in pixel space.

Downside: as we directly operate in pixel space, memory is an issue, especially when dealing with higher resolutions.

Extensions: Latent diffusion models, conditional diffusion models…

Ethical side of generative learning

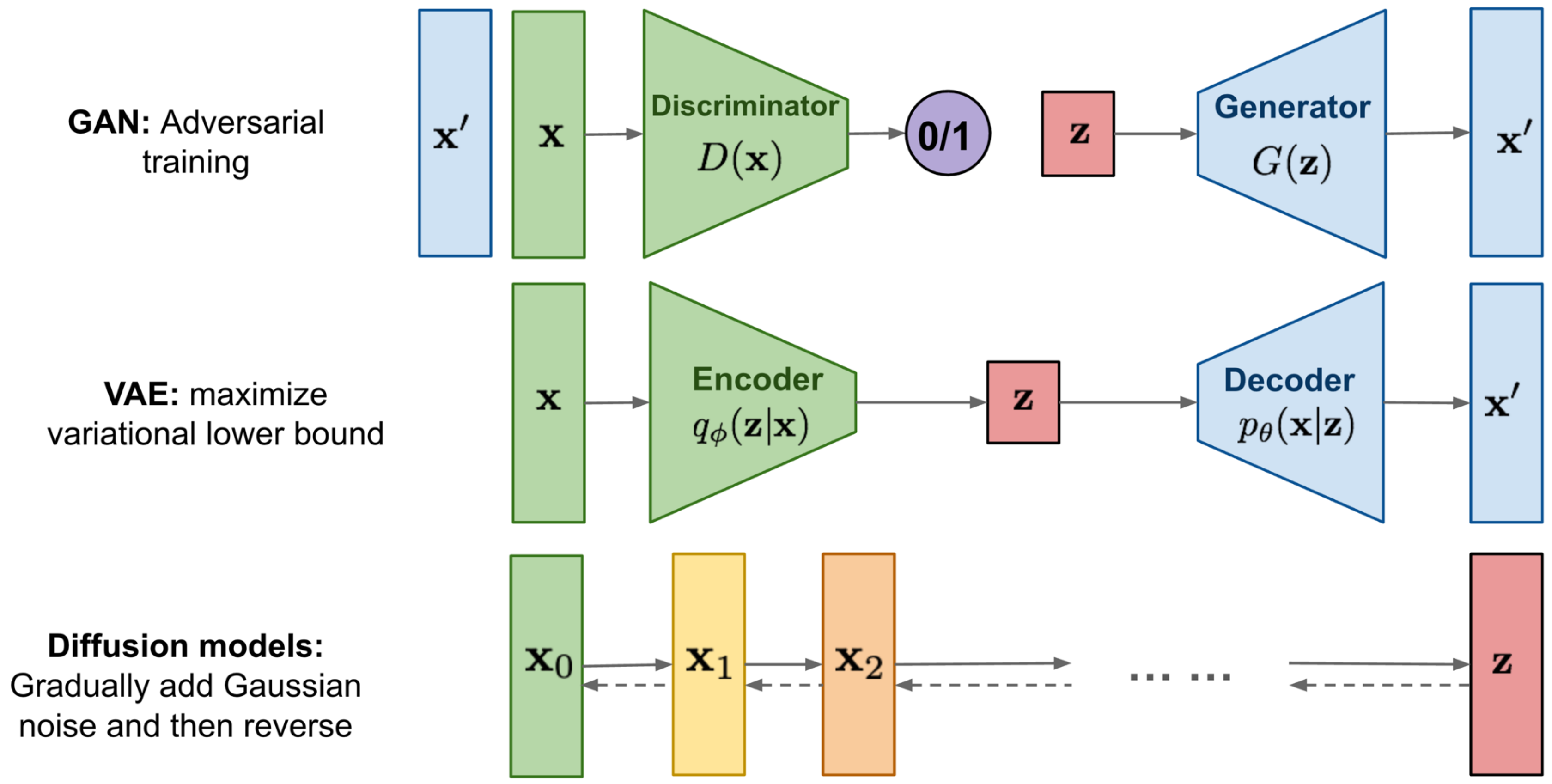
Line between real and generated data becomes blurry.

Data on which large models are trained break IP rights.

Fears for deepfakes, mass manipulation, massive changes in job markets.

AI is no longer only in the lab, we need to consider its real-world impact.

Models summarized


GAN: Adversarial training
$\mathbf{x}'$
$\mathbf{x}$
Discriminator $D(\mathbf{x})$
0/1
$\mathbf{z}$
Generator $G(\mathbf{z})$
$\mathbf{x}'$
VAE: maximize variational lower bound
$\mathbf{x}$
Encoder $q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{z}$
Decoder $p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$
Diffusion models: Gradually add Gaussian noise and then reverse
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
… …
$\mathbf{z}$

Picture by Lilian Wang

Next lecture

Lecture	Title
1	Intro and history of deep learning
3	Deep learning optimization I
5	Convolutional Neural Networks I
7	Attention
9	Self-supervised and vision-language learning
11	Deep learning for videos
13	The oddities of deep learning

Lecture	Title
2	Manually forward, automatically backward
4	Deep learning optimization II
6	Convolutional Neural Networks II
8	Graph Neural Networks
10	Auto-encoding and generation
12	Non-Euclidean deep learning
14	Q&A