Advanced Deep Learning:
- Generative Models
- Physics-Based Deep Learning

Sorbonne Université – Masters DAC et M2A. PatrickGallinari,
patrick.gallinari@sorbonne-universite.fr, https://pages.isir.upmc.fr/gallinari/

Year 2024-2025

Advanced Deep learning

- Generative models
  - Variational Auto-Encoders
  - Generative Adversarial Networks
  - Diffusion models

- AI4Science - Physics Based Deep Learning
  - Neural Nets and Ordinary Differential Equation
  - Neural Networks for modeling spatio-temporal dynamics
    - NNs as surrogate models for solving Partial Differential Equations
    - Incorporating physical knowledge in statistical dynamics models
    - Generalization for agnostic ML models for dynamics modeling
    - Foundation models for science

Generative models

Variational Auto-Encoders

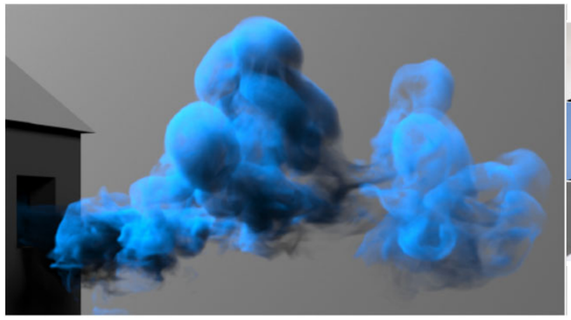
Generative Adversarial Networks

Diffusion models

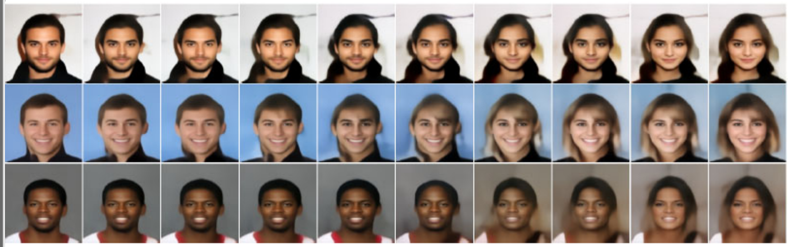
Generative models

- Objective
  - Learn a probability distribution model from data samples
    - Given $x^1, \dots, x^N \in R^n$ learn to approximate their underlying distribution $\mathcal{X}$
    - For complex distributions, there is no analytical form, and for large size spaces (R^n) approximate methods (e.g. MCMC) might fail
    - Deep generative models recently attacked this problem with the objective of handling large dimensions and complex distributions

https://en.wikipedia.org/wiki/Edmond_de_Belamy
432 k$ Christies in 2018



Xie et al. 2019
artificial smoke



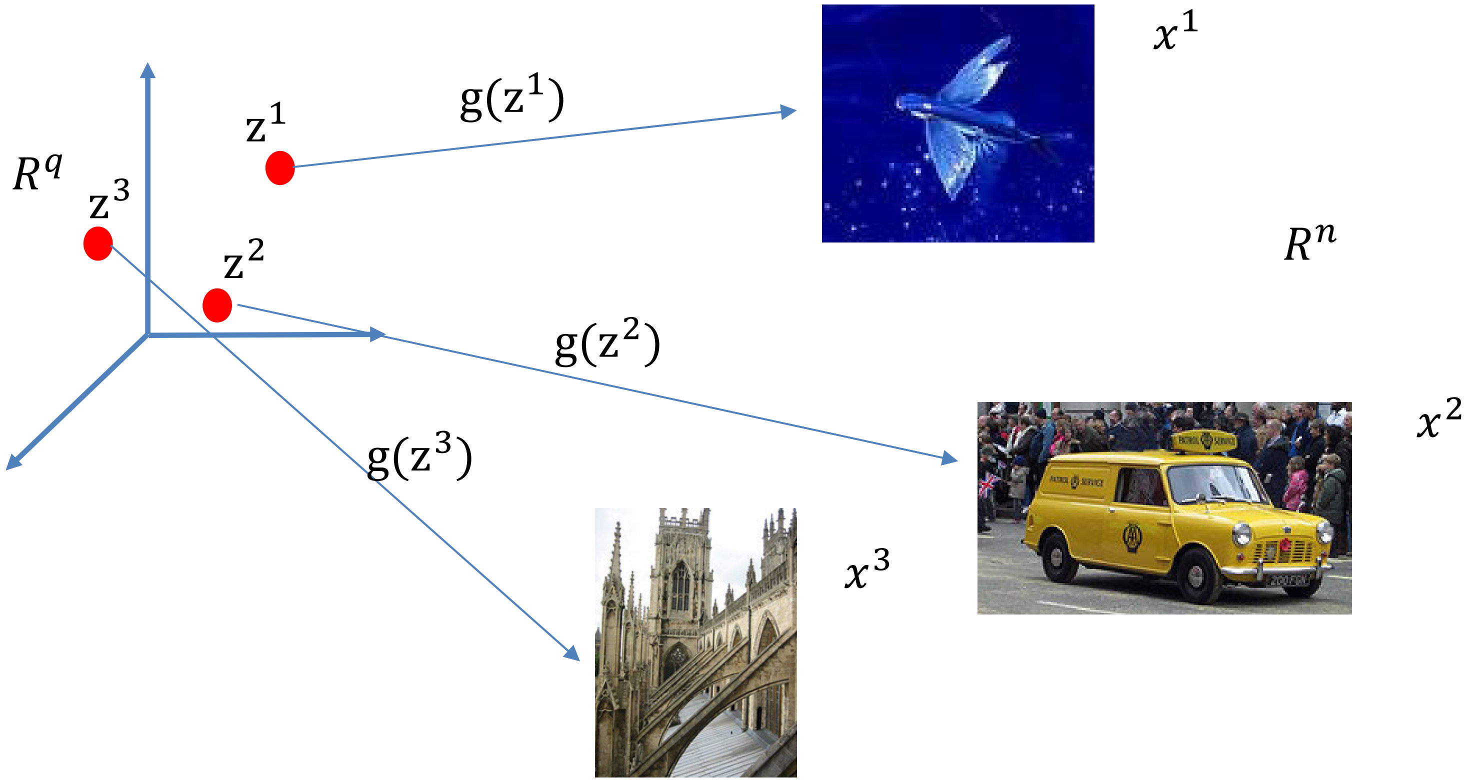
De Bezenac et al. 2021
Generating female images from male ones

Generative models

- Objective
  - General setup of deep generative models
    - Learn a generator network $g_\theta: R^q \to R^n$ that transforms a latent distribution $\mathcal{Z} \subset R^q$to match a target distribution $\mathcal{X}$
      - $\mathcal{Z}$ is usually a simple distribution e.g. Gaussian from which it is easy to sample, $q < n$
      - This is unlike traditional statistics where an analytic expression for the distribution is sought
    - Once trained the generator can be used for:
      - Sampling from the latent space:
        - $z \in R^q \sim \mathcal{Z}$ and then generate synthetic data via $g_\theta(.)$, $g_\theta(z) \in R^n$
      - When possible, density estimation $p_\theta(x) = \int \; p_\theta(x|z)p_Z(z)dz$
        - with $p_\theta(x|z)$ a function of g_θ

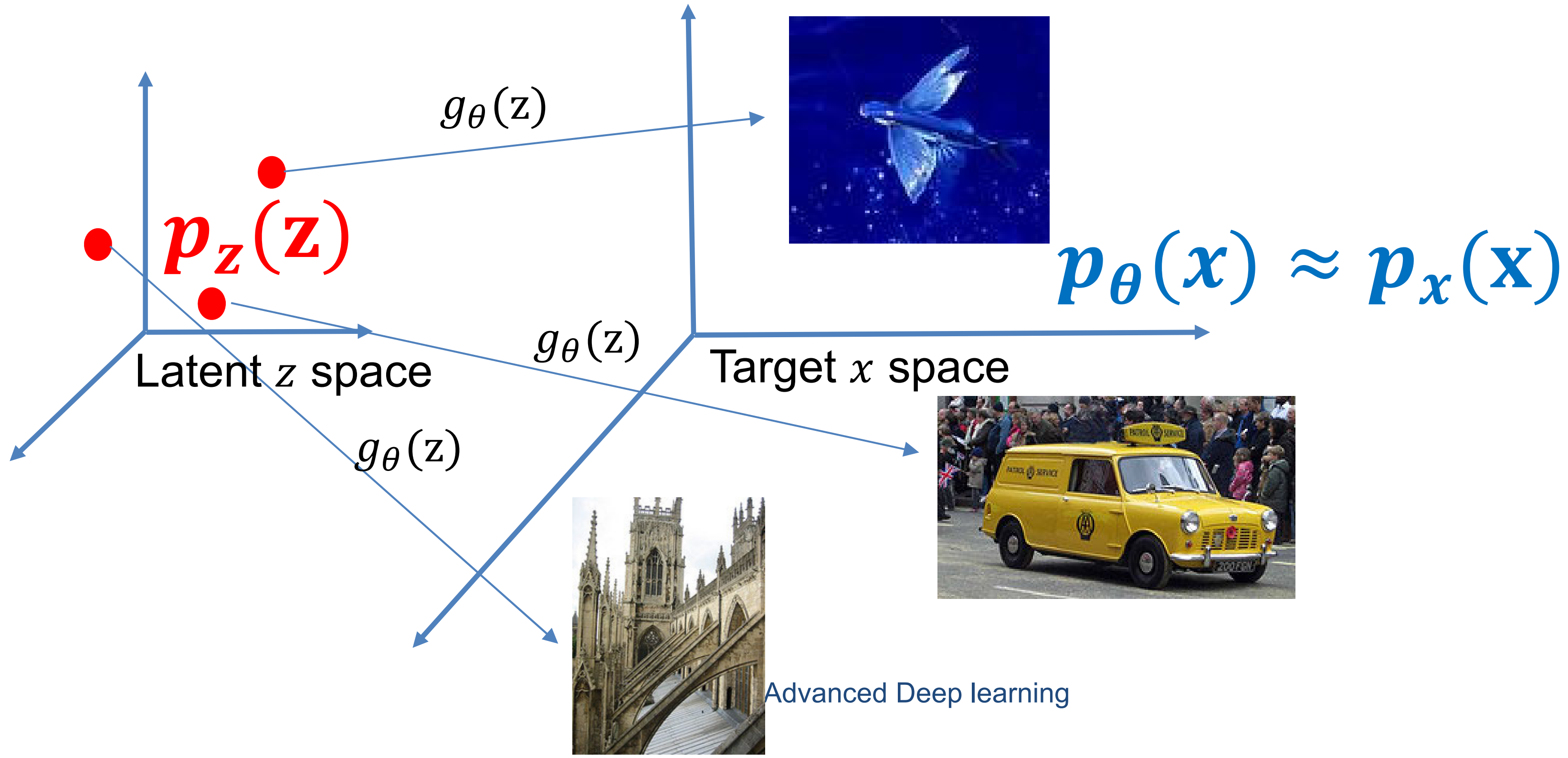
Generative models intuition

- Let $\{z^1, \ldots, z^N\}, z^i \in R^q$ and $\{x^1, \ldots, x^N\}, x^i \in R^n$, two sets of points in different spaces
  - Provided a sufficiently powerful model $g(x)$, it should be possible to learn complex deterministic mappings associating the two sets:



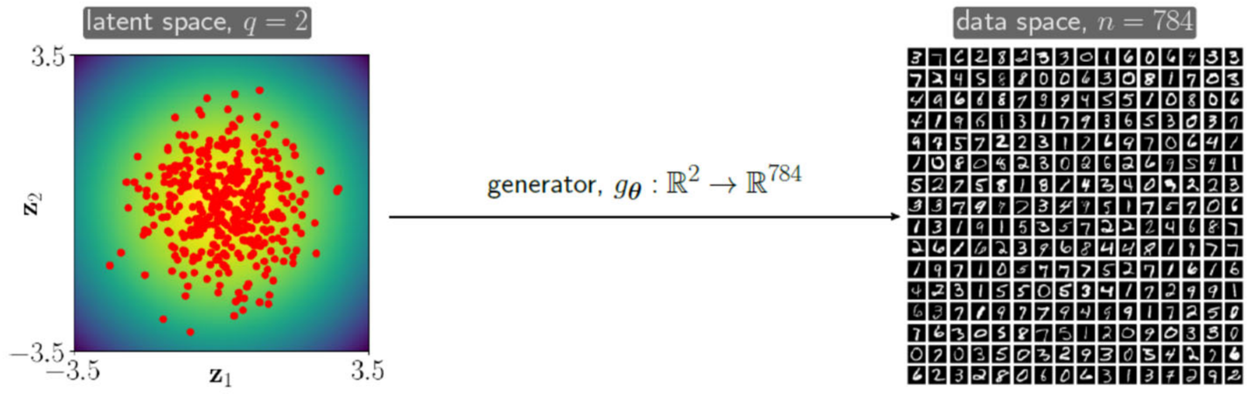
Generative models intuition

- Given distributions on a latent space $p_z(\mathrm{z})$, and on the data space $p_x(x)$, it is possible to map $p_z(\mathrm{z})$ onto $p_x(x)$?
  - g_θ defines a distribution on the target space $p_x\big(g_\theta(z)\big) = p_\theta(x)$
    - $p_\theta(x)$ is the generated data distribution, objective: $p_\theta(x) \approx p_x(x)$
  - Data generation: sample $\mathrm{z} \sim Z$, transform with g_θ, $g_\theta(z)$

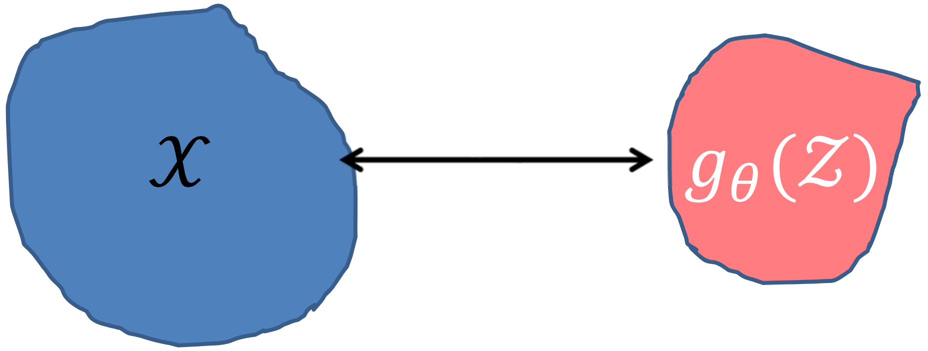


Generative models intuition

- Data generation: sample $z \sim Z$, transform with g_θ, $g_\theta(z)$



- Important issue
  - How to compare predicted distribution $p_\theta(x)$ and target distribution $p_X(x)$?



Course objective

- Introduce three popular families of generative models
  - Joint requirements
    - Learn a generator g_θ from samples so that distribution $g_\theta(\mathcal{Z})$ is close to data distribution $\mathcal{X}$, $p_\theta(x) \approx p_x(x)$
    - This requires measuring the similarity between $g_\theta(\mathcal{Z})$ and $\mathcal{X}$
      - Different similarities are used for each family
- Three families
  - Variational autoencoders
    - $g_\theta: R^q \to R^n, q \ll n$
    - Trained to maximize a lower bound of the samples' likelihood
    - Assumption: a density function explains the data
  - Generative Adversarial Networks
    - $g_\theta: R^q \to R^n, q \ll n$
    - Can approximate any distribution (no density assumption)
    - Similarity between generated and target distribution is measured via a discriminator or transport cost in the data space
  - Diffusion models
    - $g_\theta: R^q \to R^n, q \ll n$ is an iterative process based on a Markov chain
    - Assumption: a density function explains the data

Variational Auto-Encoders

After Kingma D., Welling M., Auto-Encoding Variational Bayes, ICLR 2014

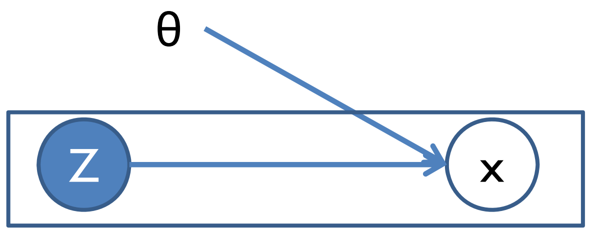
Plus some blogs – see the references

Prerequisite KL divergence

- Kullback Leibler divergence
  - Measure of the difference between two distributions p and q
  - Continuous variables
    - $D_{KL}(p(y)||q(y)) = \int_y (\log\frac{p(y)}{q(y)})p(y)dy$
  - Discrete variables
    - $D_{KL}(p(y)||q(y)) = \sum_i (\log\frac{p(y_i)}{q(y_i)})p(y_i)$
- Property
  - $D_{KL}(p(y)||q(y)) \geq 0$
  - $D_{KL}(p(y)||q(y)) = 0$ iff $p = q$
    - $D_{KL}(p(y)||q(y)) = -E_{p(y)}\left[log\frac{q(y)}{p(y)}\right] \geq -\log E_{p(y)}\left[\frac{q(y)}{p(y)}\right] = 0$
      - ☐ the first inequality is obtained via Jensen inequality:
      - ☐ For a convex function f, $f(E[x]) \leq E[f(x)]$, and $-\log x$ is a convex function
  - note: D_{KL} is asymmetric, symmetric versions exist, e.g. Jensen-Shannon divergence

Preliminaries – Variational methods

- Generative latent variable model
- Let us suppose available a joint model on the observed and latent variables $p_\theta(x, z)$



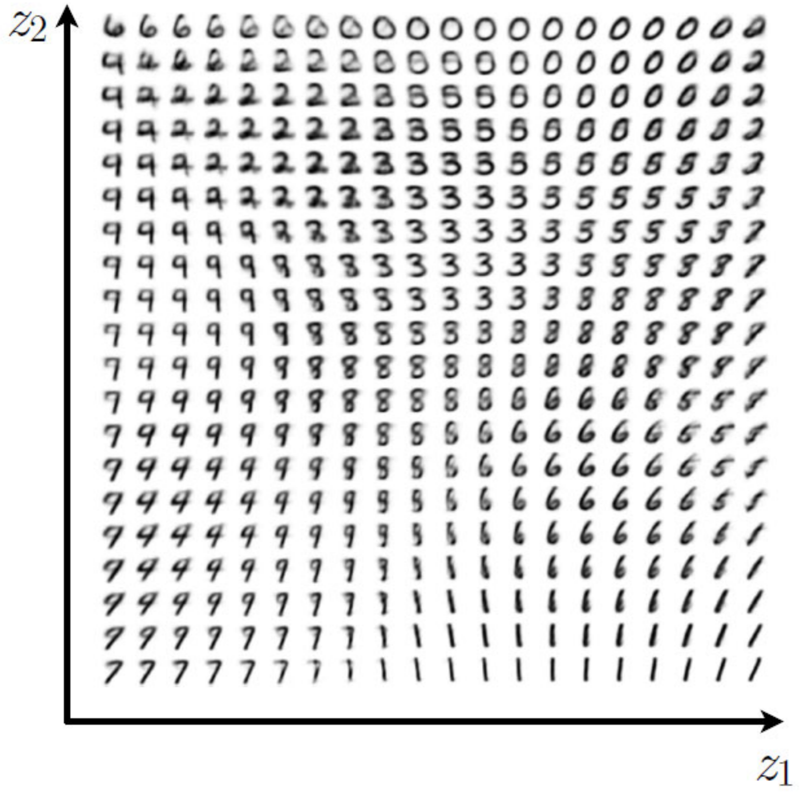
- The observations x are generated by the following process
  - Sample from $z \sim p_\theta(z)$ - $p_\theta(z)$ is the prior
  - generate $p_\theta(x|z)$ - $p_\theta(x|z)$ is the likelihood

- Training objective
  - We want to optimize the likelihood of the observed data
    - $p(x) = \int p(x|z)p(z)dz$ - $p(x)$is called the evidence
    - Computing the integral requires evaluating over all the configurations of latent variables,
    - This is often intractable
    - In order to narrow the sampling space, one may use importance sampling, i.e. sampling important z instead of sampling blindly from the prior
    - Let us introduce a sampling function $q_\Phi(z|x)$

VAEs - Intuition

- Intuitively, z might correspond to the factors conditioning the generation of the data

MNIST:



Frey Face dataset:

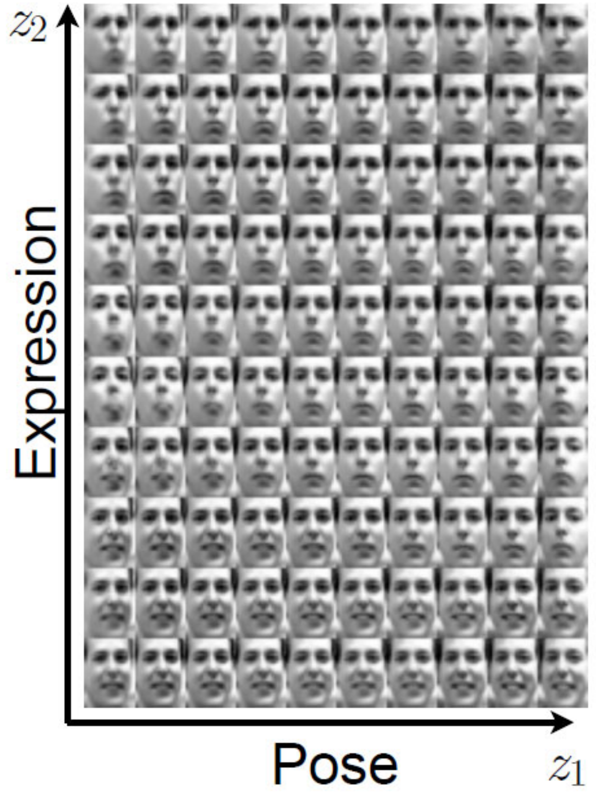


Fig. (Kingma 2015)

Generative models intuition

- What we want: organize the latent space according to some characteristics of the observations (images)

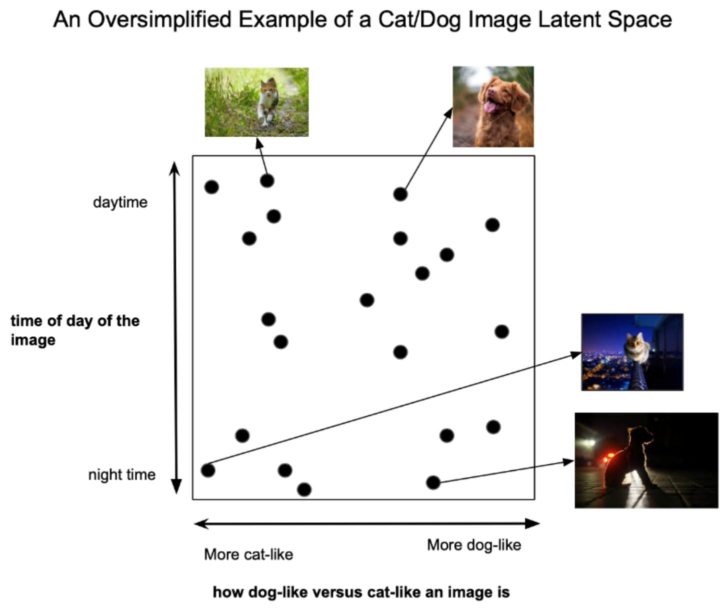

An Oversimplified Example of a Cat/Dog Image Latent Space
daytime
time of day of the image
night time
More cat-like
More dog-like
how dog-like versus cat-like an image is

Fig.: https://ml.berkeley.edu/blog/posts/vq-vae/

- See also the demos @
  - https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

VAE
Loss criterion – summary

- The log likelihood for data point x can be decomposed as
  - $\log p_\theta(x) = D_{KL}(q_\phi(z|x)||p_\theta(z|x)) + V_L(\theta,\phi;x)$
  - with
  - $V_L(\theta,\phi;x) = -D_{KL}(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[\log p_\theta(x|z)]$
- Why is it useful?
  - $D_{KL}(.||.) \geq 0$, then $V_L(\theta,\phi;x)$ is a lower bound of $\log p_\theta(x)$
  - in order to maximize $\log p_\theta(x)$, we will maximize $V_L(\theta,\phi;x)$
- $V_L(\theta,\phi;x)$ is called the ELBO: Evidence Lower Bound
  - With an appropriate choice of $q_\phi(z|x)$ this is amenable to a computationable form
  - $q_\phi(z|x)$ approximates the intractable posterior $p_\theta(z|x)$
  - This method is called variational inference
    - In general inference denotes the computations of hidden variables given observed ones (e.g. infering the class of an object)
- Note
  - Because each representation z is associated to a unique x, the loss likelihood can be decomposed for each point – this is what we do here
  - The global log likelihood is then the summation of these individual losses

VAE
Loss criterion – summary

- Variational lower bound:
  - $V_L(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - Remarks
    - $E_{q_\phi(z|x)}[\log p_\theta(x|z)]$ is a **reconstruction** term
      - Measures how well the datum x can be reconstructed from latent representation z
    - $D_{KL}(q_\phi(z|x)||p(z))$ is a **regularization** term:
      - Forces the learned distribution $q_\phi(z|x)$ to stay close to the prior $p(z)$
        - Otherwise a trivial solution would be to learn a Dirac distribution for $q_\phi(z|x)$
        - We want the z to be close in the latent space for similar xs
      - $p(z)$ has usually a simple form e.g. $\mathcal{N}(0, I)$, then $q_\phi(z|x)$ is also forced to remain simple

VAE details
Derivation of the loss function

- $\boldsymbol{\log p_\theta (x) = \mathrm{D}_{KL}(q_\phi(z|x)||p_\theta(\mathrm{z|x})) + V_L(\theta, \phi; \mathrm{x})}$

Proof

- $\log p_\theta (x) = \int_z (\log p(x)) q(z|x)\, dz \qquad (\int_z q(z|x)\, dz = 1)$
- $\log p_\theta (x) = \int_z (\log \frac{p(x,z)}{p(z|x)}) q(z|x)\, dz$
- $\log p_\theta (x) = \int_z (\log \frac{p(x,z)}{q(z|x)} \frac{q(z|x)}{p(z|x)}) q(z|x)\, dz$
- $\log p_\theta (x) = \int_z (\log \frac{p(x,z)}{q(z|x)}) q(z|x)\, dz + \int_z (\log \frac{q(z|x)}{p(z|x)}) q(z|x)\, dz$
- $\log p_\theta (x) = E_{q(z|x)}[\log \mathrm{p(x, z)} - \log q(z|x)] + D_{KL}(q(z|x)||p(z|x))$

$$\log p_\theta (x) = V_L(\theta, \phi; x) + \mathrm{D}_{KL}(q_\phi(z|x)||p_\theta(z|x))$$

with

$$V_L(\theta, \phi; x) = E_{q(z|x)}[\log \mathrm{p}_\theta(x, z) - \log q_\phi(z|x)]$$

- Maximizing $\log p_\theta (x)$ is equivalent to maximizing $V_L(\theta, \phi; x)$ (and minimizing $\mathrm{D}_{KL}(q_\phi(z|x)||p_\theta(z|x))$
- $V_L(\theta, \phi; x)$ is called an Evidence Lower Bound (ELBO)

VAE details
Derivation of the loss function

- $\boldsymbol{V_L(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[log\, p_\theta(x|z)]}$

Proof:

- $V_L(\theta, \phi; x) = \mathrm{E}_{q_\phi(z|x)}[log p_\theta(x, z) - \log q_\phi(z|x)]$
- $V_L(\theta, \phi; x) = \mathrm{E}_{q_\phi(z|x)}[log p_\theta(x|z) + \log p_\theta(z) - \log q_\phi(z|x)]$
- $V_L(\theta, \phi; x) = -\mathrm{D}_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z)) + \mathrm{E}_{q_\phi(z|x)}[log p_\theta(x|z)]$

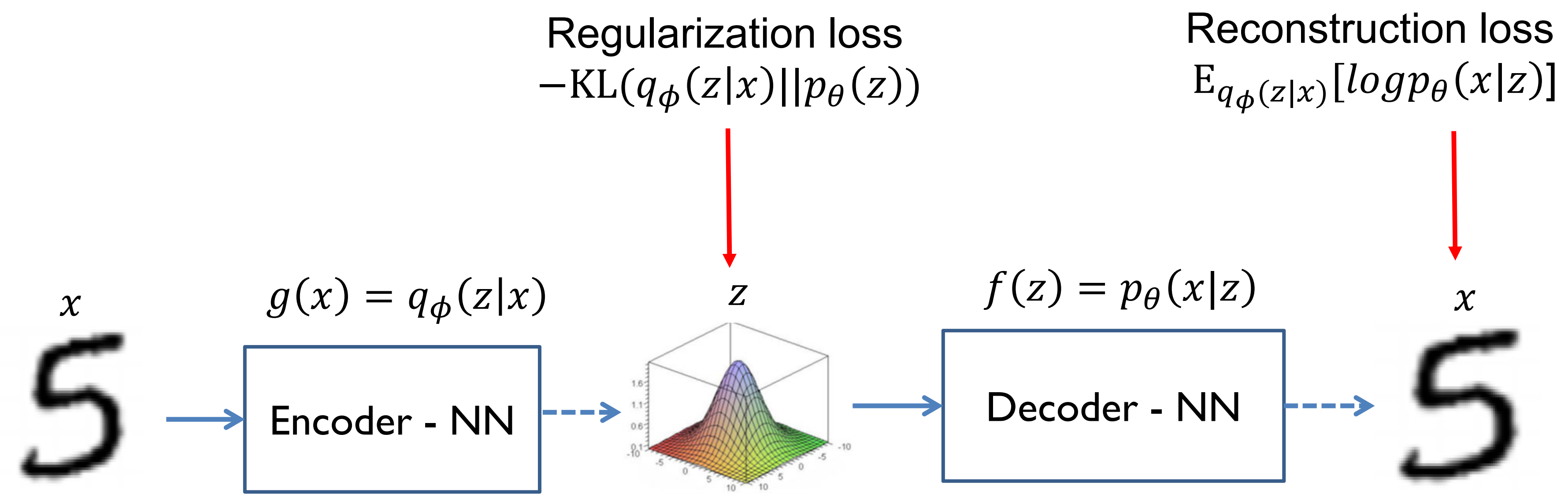
VAE
Loss criterion – summary

- Variational lower bound:
  - $V_L(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[\log p_\theta(x|z)]$

  - This form provides a link with a NN implementation
    - The generative $p_\theta(x|z)$ and inference $q_\phi(z|x)$ modules are implemented by NNs
    - They will be trained to maximize the reconstruction error for each (z, x): $E_{q_\phi(z|x)}[\log p_\theta(x|z)]$ term
    - The inference module $q_\phi(z|x)$ will be constrained to remain close to the prior $p(z)$: $-\mathrm{D_{KL}}(q_\phi(z|x)||p_\theta(z)) \approx 0$

VAE
Loss - summary

- Loss function in the NN model



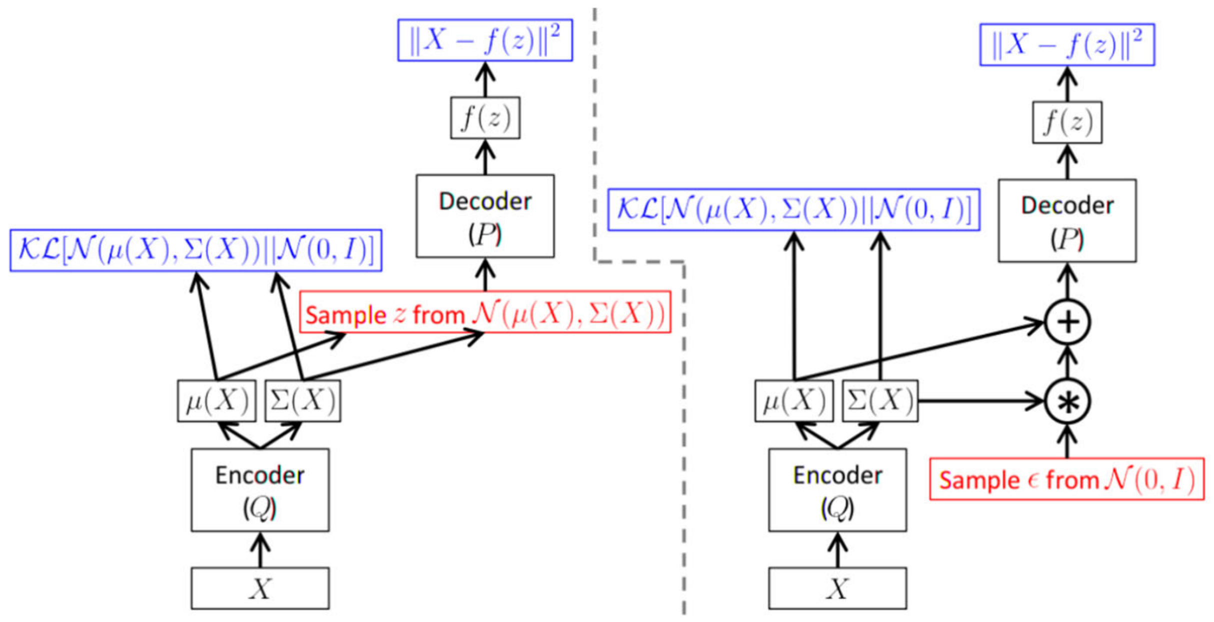
- Training performed via Stochastic gradient
  - This requires an analytical expression for the loss functions and for gradient computations
    - $\dashrightarrow$ Sampling
    - $\longrightarrow$ deterministic

VAE- reparametrization trick

- Training with stochastic units: reparametrization trick
  - Not possible to propagate the gradient through stochastic units (the zs and xs are generated via sampling)
  - Solution
    - Parametrize z as a deterministic transformation of a random variable ϵ: $z = g_\phi(x, \epsilon)$ with $\epsilon \sim p(\epsilon)$ independent of ϕ, e.g. $\epsilon \sim N(0,1)$
    - Example
      - If $z \sim \mathcal{N}(\mu, \sigma)$, it can be reparameterized by $z = \mu + \sigma \odot \epsilon$, with $\epsilon \sim \mathcal{N}(0,1)$, with $\odot$ pointwise multiplication (μ, σ are vectors here)
      - For the NN implementation we have: $z = \mu_z(x) + \sigma_z(x) \odot \epsilon_z$
    - This will allow the derivatives to « pass » through the z
      - With this expression, one may compute the gradients of the ELBO with to the NN parameters of $\mu_z(x)$ and $\sigma_z(x)$
      - For the derivative, the sampling operation is regarded as a deterministic operation with an extra input ϵ_z, whose distribution does not involve variables needed in the derivation

VAE - reparametrization trick

- Reparametrization (fig. from D. Kingma)



$\|X - f(z)\|^2$
$f(z)$
Decoder (P)
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X)) \| \mathcal{N}(0, I)]$
Sample z from $\mathcal{N}(\mu(X), \Sigma(X))$
$\mu(X)$
$\Sigma(X)$
Encoder (Q)
X
$\|X - f(z)\|^2$
$f(z)$
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X)) \| \mathcal{N}(0, I)]$
Decoder (P)
$+$
$*$
$\mu(X)$
$\Sigma(X)$
Sample ϵ from $\mathcal{N}(0, I)$
Encoder (Q)
X

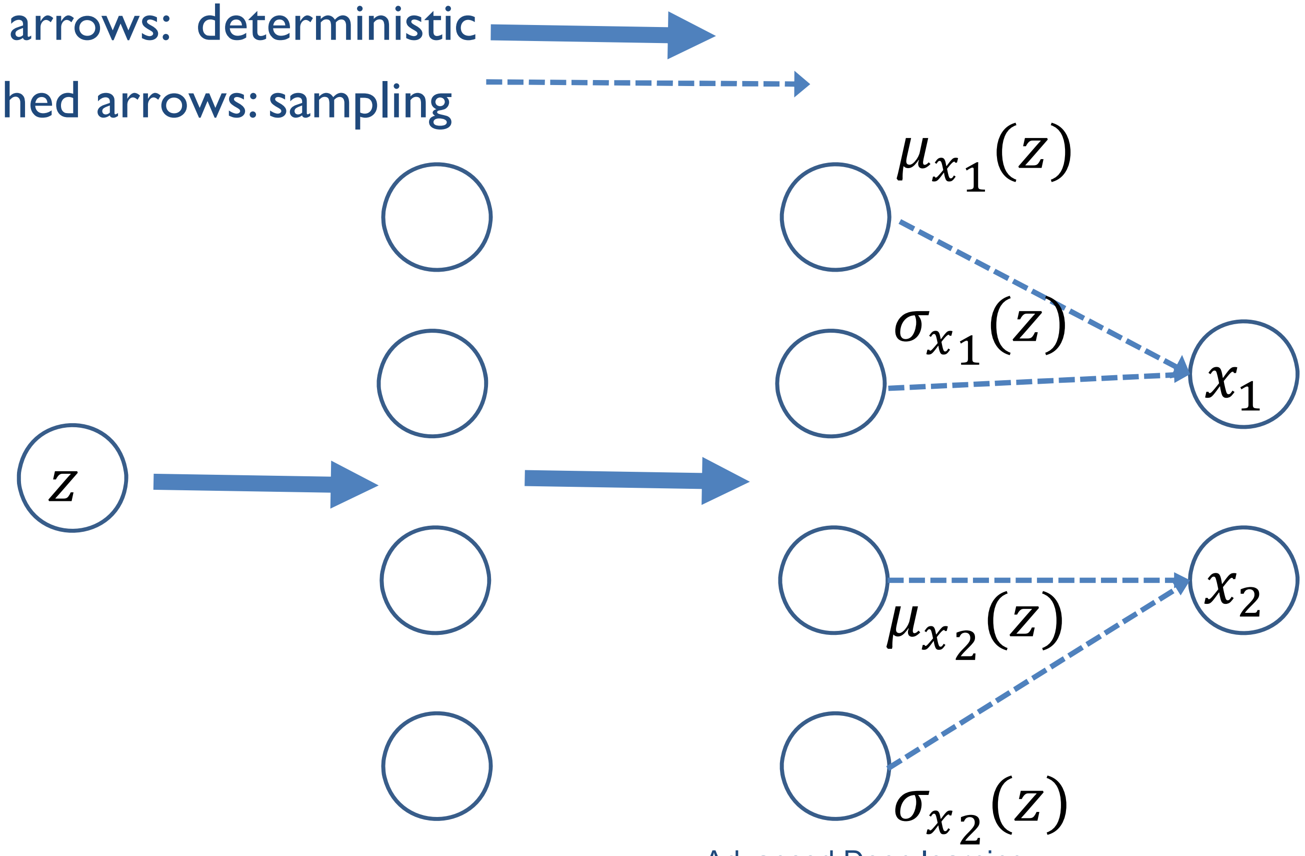
Figure 4: A training-time variational autoencoder implemented as a feed-forward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network.

VAE
Exemple: Gaussian priors and posteriors

- Special case: gaussian priors and posteriors
- Hyp:
  - $p(z) = \mathcal{N}(0, I)$
  - $p_\theta(x|z) = \mathcal{N}(\mu(z), \sigma(z)), \sigma(z)$ diagonal matrix, $x \in R^D$
  - $q_\phi(z|x) = \mathcal{N}(\mu(x),\ \sigma(x)), \sigma(x)$ diagonal matrix, $z \in R^J$

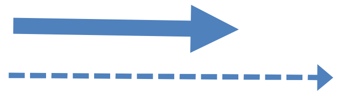
VAE
Exemple: Gaussian priors and posteriors - illustration

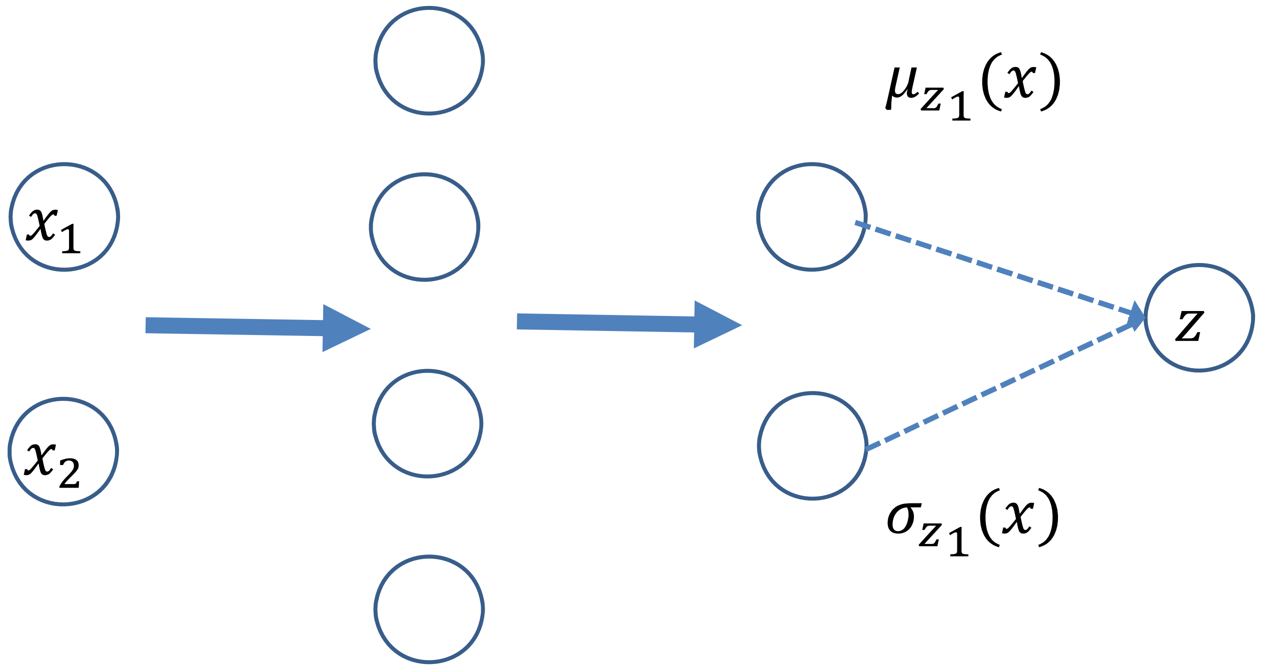
- Decoder:
  - in the example z is 1 dimensional and x is 2 dimensional, f is a 1 hidden layer MLP with gaussian output units and tanh hidden units
  - full arrows: deterministic
  - dashed arrows: sampling



VAE
Gaussian priors and posteriors - illustration

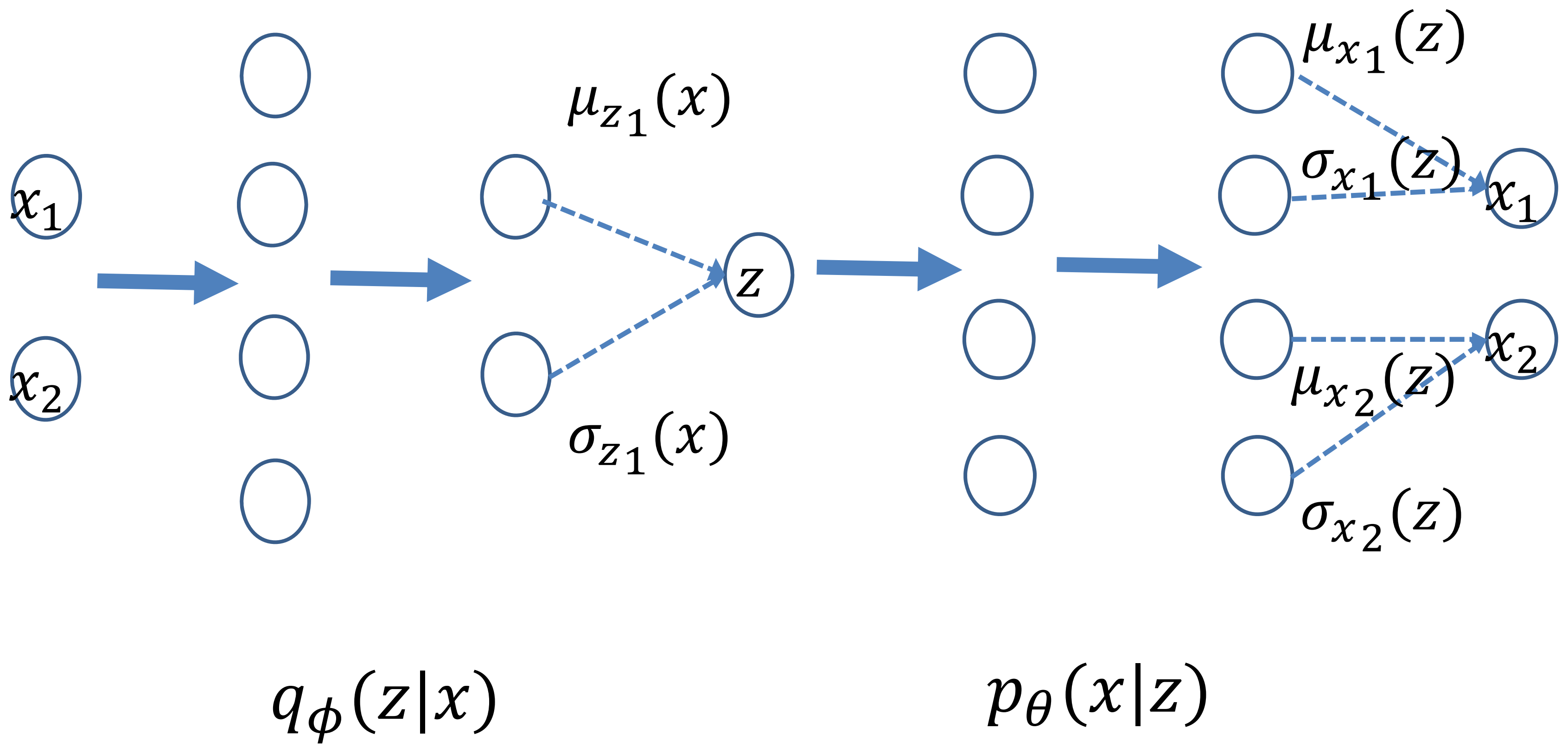
- Encoder
  - in the example z is 1 dimensional and x is 2 dimensional, g is a 1 hidden layer MLP with gaussian output units and tanh hidden units
  - full arrows: deterministic
  - dashed arrows: sampling





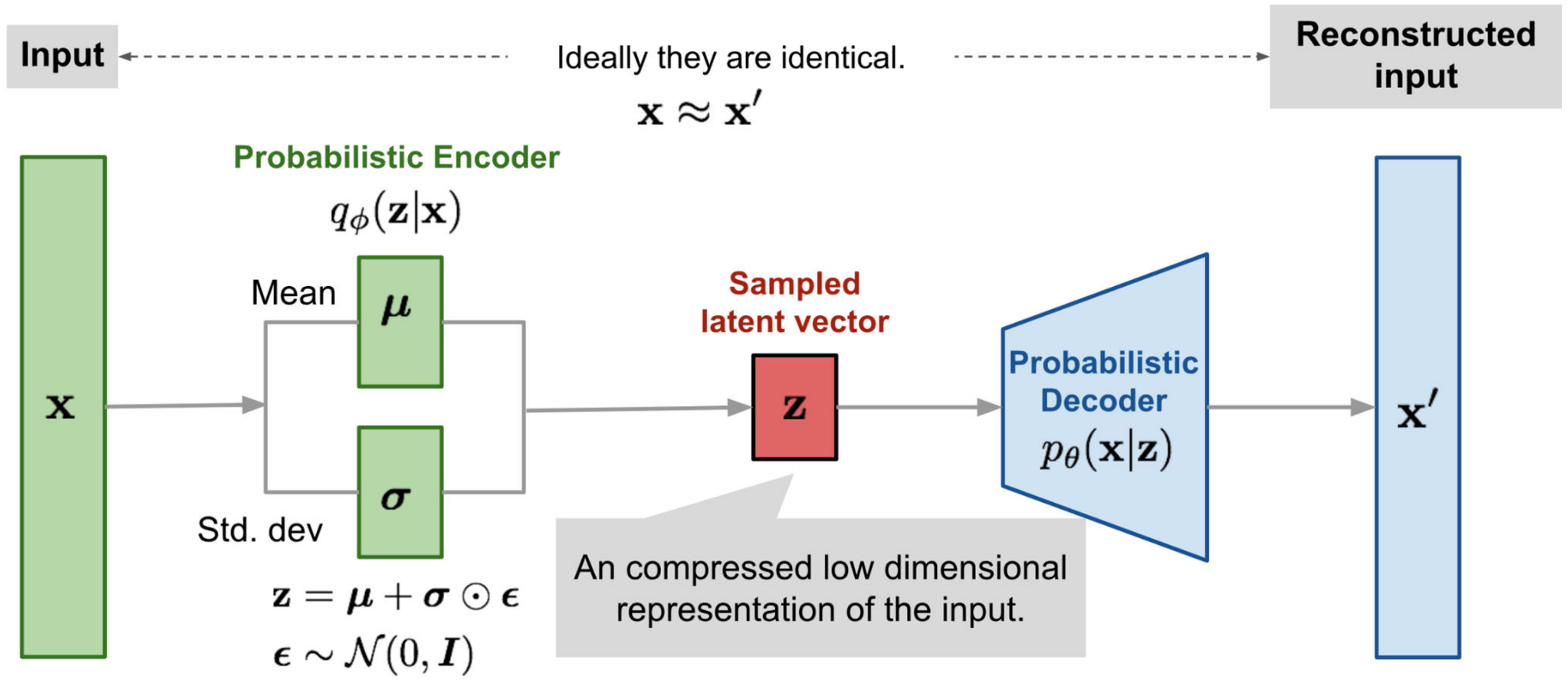
VAE
Gaussian priors and posteriors

- Putting it all together



VAE
Gaussian priors and posteriors

- Additional illustration


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
Mean
$\boldsymbol{\mu}$
Std. dev
$\boldsymbol{\sigma}$
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
Sampled latent vector
$\mathbf{z}$
An compressed low dimensional representation of the input.
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

https://lilianweng.github.io/posts/2018-08-12-vae/

VAE details
for Gaussian priors and posteriors

VAE – instanciation example
Gaussian priors and posteriors

- Special case: gaussian priors and posteriors
- Hyp:
  - $p(z) = \mathcal{N}(0, I)$
  - $p_\theta(x|z) = \mathcal{N}(\mu(z), \sigma(z)), \sigma(z)$diagonal matrix, $x \in R^D$
  - $q_\phi(z|x) = \mathcal{N}(\mu(x),\ \sigma(x)), \sigma(x)$ diagonal matrix, $z \in R^J$

- Variational lower bound
  - $V_L(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p(z)) + E_{q_\phi(z|x)}[\log p_\theta(x|z)]$
  - In this case, $\mathrm{D}_{KL}(q_\phi(z|x)||p(z))$ has an analytic expression (see next slide)
    - $-D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log\left(\left(\sigma_{z_j}\right)^2\right) - \left(\mu_{z_j}\right)^2 - \left(\sigma_{z_j}\right)^2)$

  - $E_{q_\phi(z|x)}[\log p_\theta(x|z)]$ is estimated using Monte Carlo sampling
    - $E_{q_\phi(z|x)}[\log p_\theta(x|z)] \simeq \frac{1}{L}\sum_{l=1}^{L}\log(p_\theta(x|z^{(l)})$
    - $\log(p_\theta\left(x|z^{(l)}\right) = -(\sum_{j=1}^{D}\frac{1}{2}\log\left(\sigma_{x_j}^2\left(z^{(l)}\right)\right) + \frac{(x_j - \mu_{x_j}(z^{(l)}))^2}{2\sigma_{x_j}^2(z^{(l)})})$
    - i.e. L samples with $z^{(l)} = g_\phi\left(x, \epsilon^{(l)}\right)$

VAE - instanciation example
Gaussian priors and posteriors (demos on next slides)

- If $z \in R^J : -D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log\left(\left(\sigma_j\right)^2\right) - \left(\mu_j\right)^2 - \left(\sigma_j\right)^2)$
- proof
  - $D_{KL}(q_\phi(z)\big|\big|p(z)\big) = \int \quad q_\phi(z) \log \frac{q_\phi(z)}{p(z)} dz$
- Consider the 1 dimensional case
  - $\int q_\phi(z) \log p(z) dz = \int \mathcal{N}(z; \mu, \sigma) \log \mathcal{N}(z; 0{,}1) dz$
  - $\int q_\phi(z) \log p(z) dz = -\frac{1}{2}\log(2\pi) - \frac{1}{2}(\mu^2 + \sigma^2)$
    - Property of 2nd order moment of a Gaussian
  - $\int q_\phi(z) \log q_\phi(z) dz = \int \mathcal{N}(z; \mu, \sigma) \log \mathcal{N}(z; \mu, \sigma) dz$
  - $\int q_\phi(z) \log q_\phi(z) dz = -\frac{1}{2}\log(2\pi) - \frac{1}{2}(1 + \log \sigma^2)$
  - ……

  - Since both ddps are diagonal, extension to J dimensions is straightforward, hence the result

VAE - instanciation example
Gaussian priors and posteriors – demos for the 1 dimensional case

- Remember $q_\phi(z|x) = \mathcal{N}(\mu(x),\ \sigma(x))$
- Then $\int q_\phi(z)\log p(z)dz = \int \mathcal{N}(z;\mu,\sigma)\log\mathcal{N}(z;0{,}1)dz$
- $= E_{q_\Phi}[\log\mathcal{N}(z;0{,}1)]$

  $= E_{q_\Phi}[\log(\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{z^2}{2}\right))]$

  $= E_{q_\Phi}\left[-\frac{1}{2}\log 2\pi - \frac{z^2}{2}\right]$

  $= -\frac{1}{2}\log 2\pi - \frac{1}{2}E_{q_\Phi}\ [z^2]$
- What is the value of $E_q[z^2]$?
  - $E_{q_\Phi}[(z-\mu)^2] = \sigma^2$
  - $E_{q_\Phi}[z^2] - 2E_{q_\Phi}[z\mu] + \mu^2 = \sigma^2$
    - $E_{q_\Phi}[z\mu] = \mu^2$
  - $E_{q_\Phi}[z^2] = \ \mu^2 + \sigma^2$
- Then $\boldsymbol{\int q_\phi(z)\log p(z)dz = -\frac{1}{2}\log 2\pi - \frac{1}{2}(\mu^2 + \sigma^2)}$

VAE - instanciation example
Gaussian priors and posteriors – demos for the 1 dimensional case

- $\int q_\phi(z) \log q_\phi(z) dz = \int \mathcal{N}(z;\mu,\sigma) \log \mathcal{N}(z;\mu,\sigma) dz$

$$= E_{q_\Phi}[\log(\frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right))]$$

$$= -\frac{1}{2}\log 2\pi - \log\sigma - E_{q_\Phi}[\frac{(z-\mu)^2}{2\sigma^2}]$$

$$= -\frac{1}{2}\log 2\pi - \frac{1}{2}\log\sigma^2 - \frac{1}{2}$$

$$= -\frac{1}{2}\log 2\pi - \frac{1}{2}(\log\sigma^2 + 1)$$

VAE - instanciation example
Gaussian priors and posteriors

- Loss
- Regularization term
  - $-D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log\left((\sigma_j)^2\right) - (\mu_j)^2 - (\sigma_j)^2)$
- Reproduction term
  - $\log(p(x|z)) = \sum_{j=1}^{D}\frac{1}{2}\log(\sigma_j^2(z)) + \frac{(x_j-\mu_j(z))^2}{2\sigma_j^2(z)}$
- Training
  - Mini batch or pure stochastic
    - Repeat
      - $x\leftarrow$ random point or minibatch
      - $\epsilon \leftarrow$ sample from $p(\epsilon)$ for each x
      - $\theta\leftarrow \nabla_\theta V_L(\theta, \phi; x, g(\epsilon, \phi))$
      - $\phi\leftarrow \nabla_\phi V_L(\theta, \phi; x, g(\epsilon, \phi))$
    - Until convergence

Learning discrete distributions: VQ-VAE (highlights)

- So far we considered continuous latent distributions
- There are several instances were discrete distributions are more appropriate
  - Text data, objects in images (color, size, orientation,…), etc
  - There are several algorithms, e.g. transformers designed to work with discrete data
  - Teaser: Dall-e – makes use of a discrete VAE together with transformers in order to generate diverse images
    - https://openai.com/blog/dall-e/, https://openai.com/dall-e-2/
    - https://gpt3demo.com/apps/openai-dall-e
    - https://www.craiyon.com/ (mini version of Dall-e)

Learning discrete distributions: VQ-VAE

- What is a discrete latent distribution?

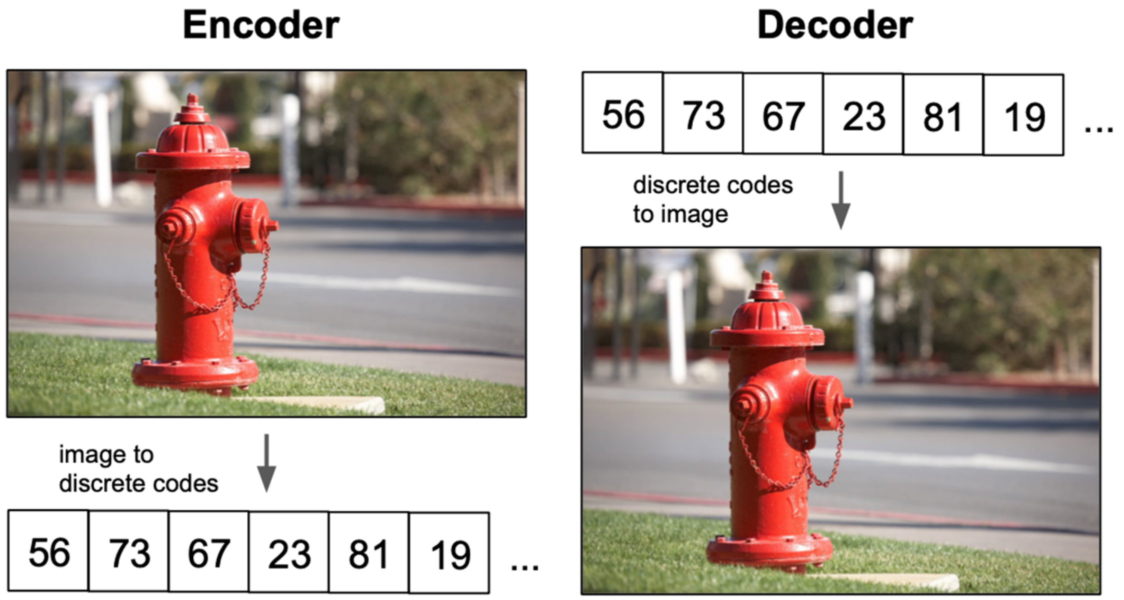


Fig: https://ml.berkeley.edu/blog/posts/vq-vae/

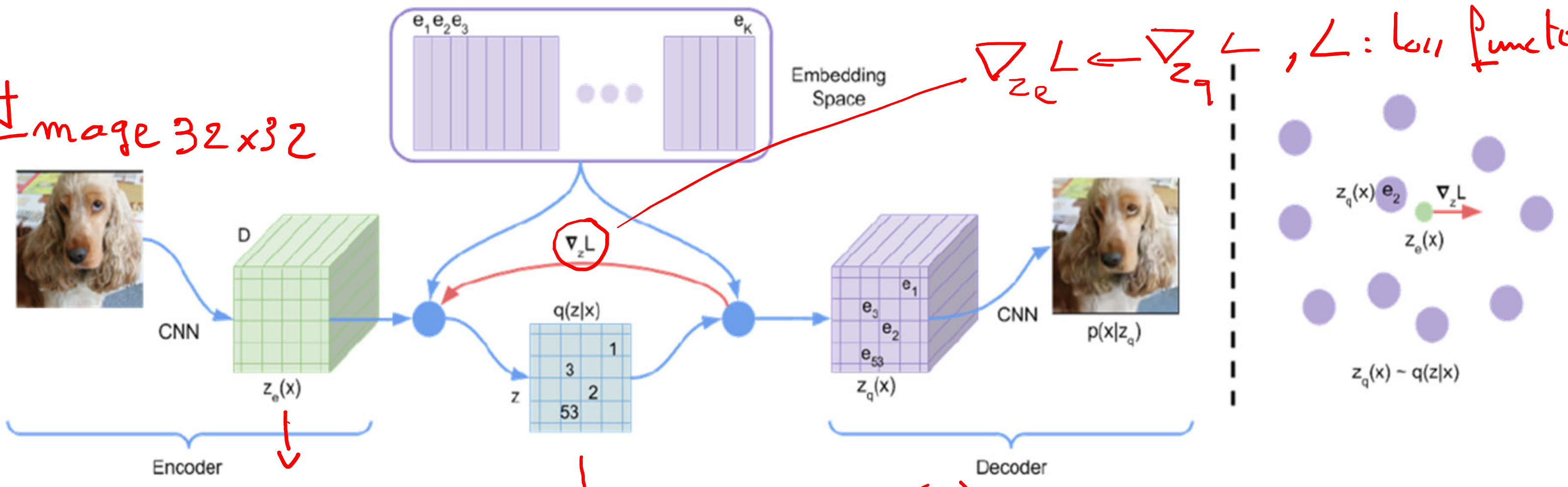
Learning discrete distributions: VQ-VAE

- VQ-VAE modifies the vanilla VAE by adding a discrete codebook of vectors to the VAE - It is used to quantize the VAE bottleneck
  - General scheme: VQ-VAE paper - https://arxiv.org/pdf/1711.00937.pdf

codebook: $e_i \in \mathbb{R}^D$, e.g $K = 512$ (# of code vectors)

$\nabla_{z_e} L \leftarrow \nabla_{z_q} L$, L: loss function

Image 32x32



$z_e(x)$ $\downarrow$ e.g $30 \times 30 \times D$ tensor

indexes: V.Q. $30 \times 30 \times 1$ indicator of codebook vectors closest to $z_e(x)$

$z_q(x)$ $30 \times 30 \times D$ tensor

$\rightarrow z_i$ is the index of the closest e_k to the code vector of $z_e(x)$ in position i in $z(x)$

Learning discrete distributions: VQ-VAE

- Loss function
  - $L = \| x - Dec\left(z_q(x)\right) \|^2 + \| sg(z_e(x)) - z_q(x) \|^2 + \beta \| z_e(x) - sg\left(z_q(x)\right) \|^2$
  - With $sg(z)$ stop gradient, i.e. do not back-propagate through z
    - $\| x - Dec\left(z_q(x)\right) \|^2$: train **decoder** and **encoder**
    - $\| sg(z_e(x)) - z_q(x) \|^2$: train the **codebook** $e = z_q(x)$
    - $\| z_e(x) - sg\left(z_q(x)\right) \|^2$: train **encoder**, forces $z_e(x)$ to stay close to $e = z_q(x)$
      - This is because the codebook does not train as fast as the encoder and the decoder
        - Prevents the encoder values to diverge

- Gradients
  - Since it is not possible to compute the gradient through the VQ component, it is proposed to simply copy the gradient w.r.t. z_q to z_e
  - $\nabla_{z_e(x)} \| x - Dec\left(z_q(x)\right) \|^2 = \nabla_{z_q(x)} \| x - Dec\left(z_q(x)\right) \|^2$
  - This is called straight-through gradient
- Note
  - This is an incomplete description, the model requires additional steps
  - Dall-e makes use of a slightly different discrete VAE (called dVAE)

- References
  - Nice blogs explaining VAEs
    - https://lilianweng.github.io/posts/2018-08-12-vae/
    - https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
    - https://www.fenghz.xyz/vector-quantization-based-generative-model/
    - Luo, C. (2022). Understanding Diffusion Models: A Unified Perspective. http://arxiv.org/abs/2208.11970 - positions hierarchical VAEs w.r.t diffusion models
  - Blogs introducing variational inference
    - https://blog.evjang.com/2016/08/variational-bayes.html
    - https://towardsdatascience.com/bayesian-inference-problem-mcmc-and-variational-inference-25a8aa9bce29
  - Papers
    - Kingma, D. P., & Welling, M. (2014). Auto-Encoding Variational Bayes. *ICLR (2014)*, *Ml*, 1–14. http://arxiv.org/abs/1312.6114

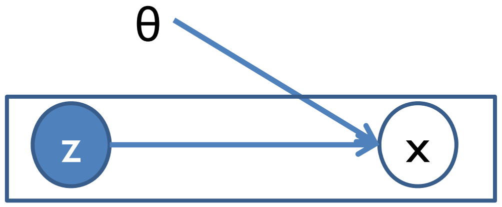
Generative Adversial Networks - GANs

Ian J. Goodfellow, et al. 2014

There has been a strong hype for GANs for several years - O(1000) GAN papers on Arxiv

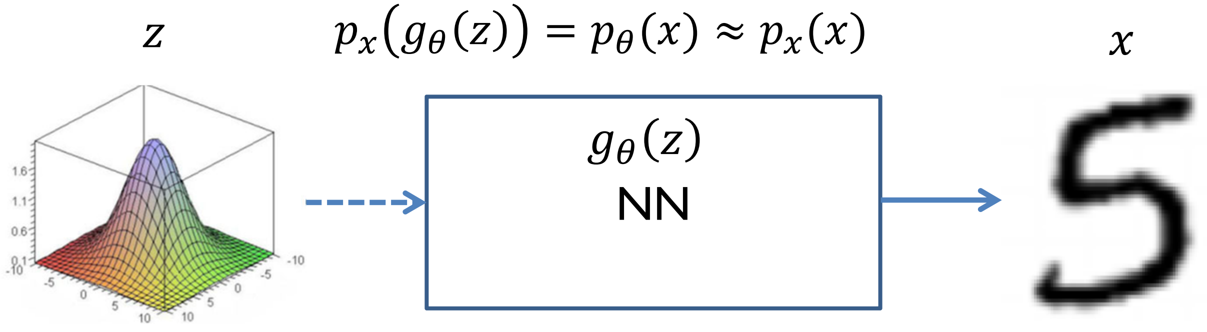
GANs

- Generative latent variable model



  - Given Samples $x^1, \dots, x^N \in R^n$, with $x \sim \mathcal{X}$, latent space distribution $z \sim \mathcal{Z}$ e.g $z \sim \mathcal{N}(0, \mathrm{I})$, use a NN to learn a possibly complex mapping $g_\theta : R^q \to R^n$ such that:

$$p_x\big(g_\theta(z)\big) = p_\theta(x) \approx p_x(x)$$



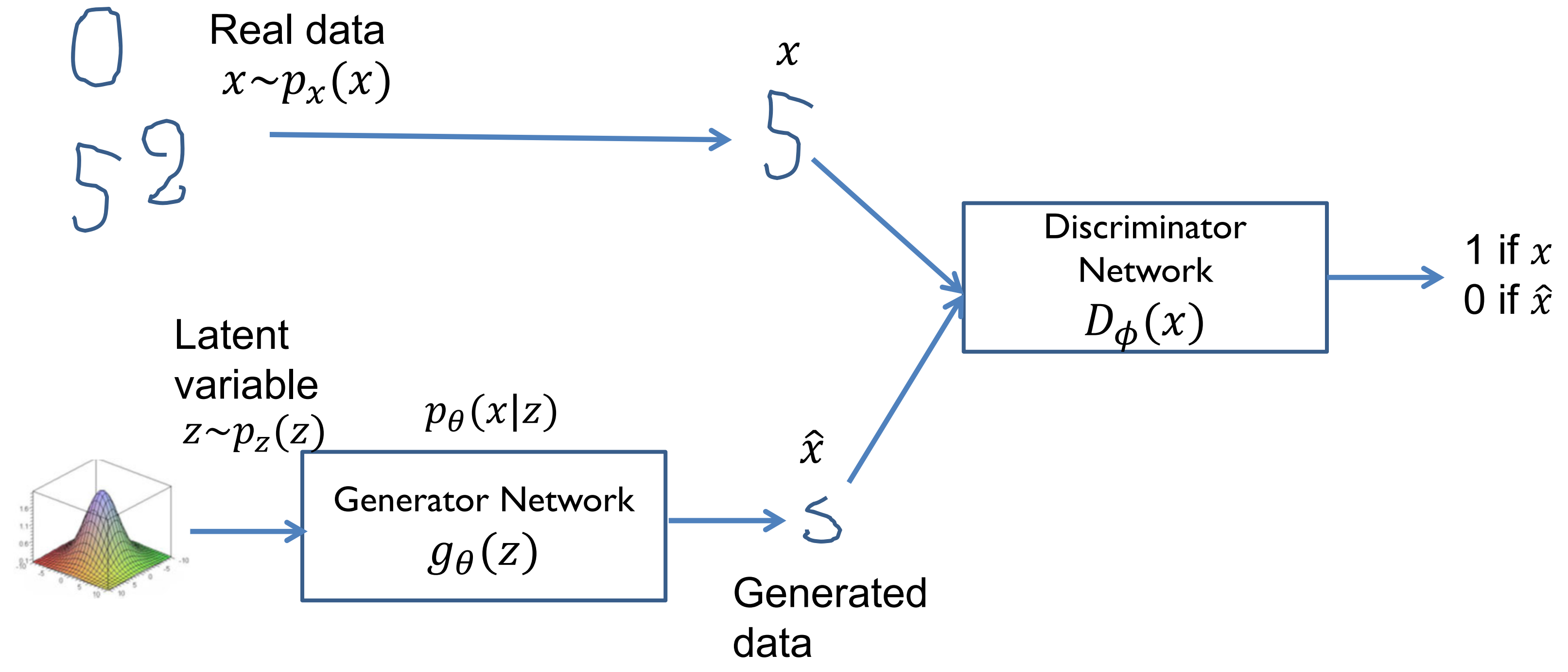
- Different solutions for measuring the similarity between $p_\theta(x)$ and $p_x(x)$
  - In this course: binary classification
- Note:
  - Once trained, sample from z directly generates the samples $g_\theta(z)$
  - Different from VAEs and Flows where the NN $g_\theta(.)$ generate distribution parameters

GANs – Adversarial training as binary classification

- Principle
  - A **generative** network generates data after sampling from a latent distribution
  - A **discriminant** network tells if the data comes from the generative network or from real samples
    - The discriminator will be used to measure the distance between the distributions $p_\theta(x)$ and $p_x(x)$
  - The two networks are trained together
    - The generative network tries to fool the discriminator, while the discriminator tries to distinguish between true and artificially generated data
    - The problem is formulated as a MinMax game
    - The Discriminator will force the Generator to be « clever » and learn the data distribution
- Note
  - No hypothesis on the existence of a density function
    - i.e. no density estimate (Flows), no lower bound (VAEs)

GANs – Adversarial training as binary classification Intuition - Training

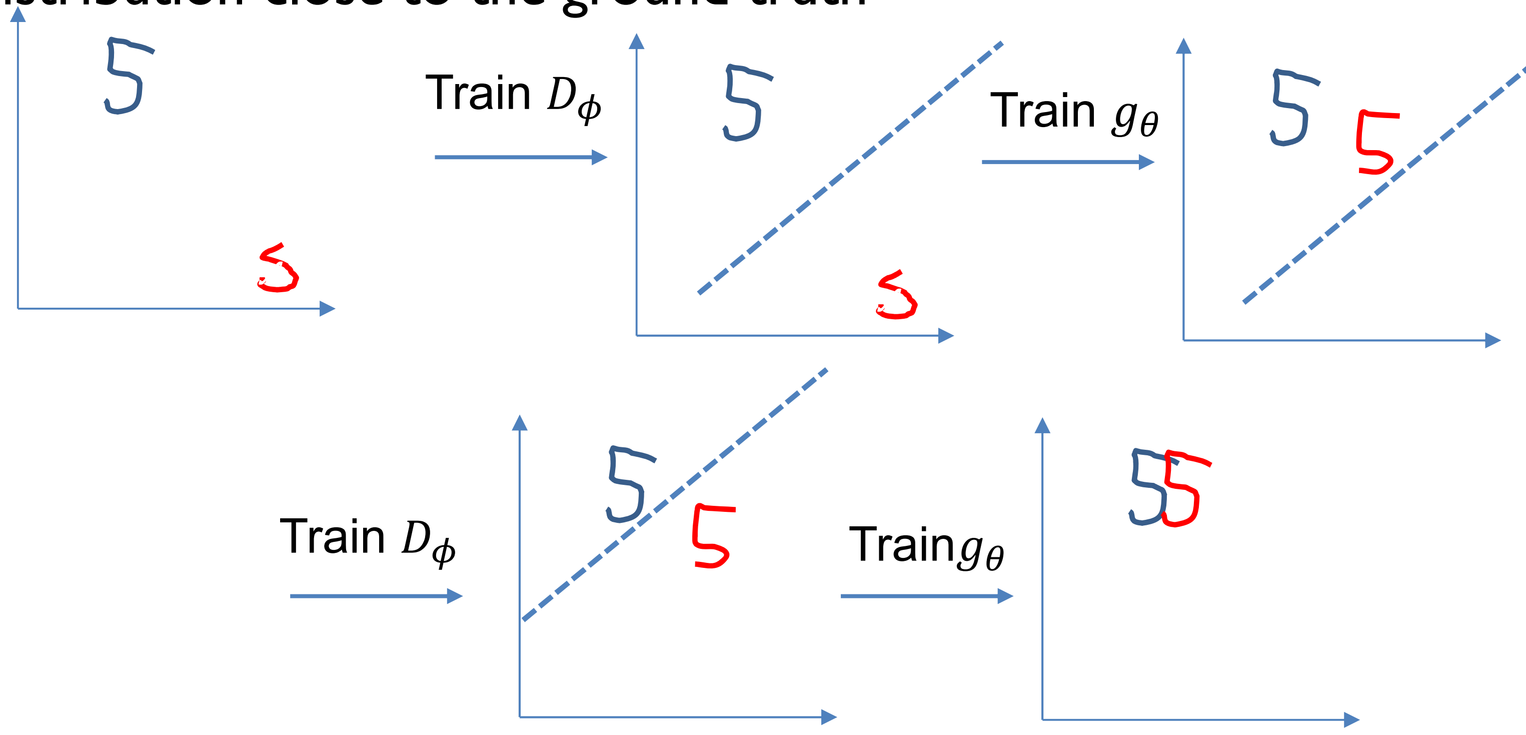
- Discriminator is presented alternatively with true (x) and fake ($\hat{x} = g_\theta(z)$)data



D_ϕ and g_θ are typically MLPs/Deep CNNs/…

GAN – Adversarial training as binary classification
Intuition - Training

- Algorithm alternates between optimizing D_ϕ (separate true and generated data) and g_θ (generate data as close as possible to true examples) – Once trained, G should be able to generate data witha distribution close to the ground truth



GANs - Adversarial training as binary classification
Loss function (Goodfellow et al. 2014)

- $x \sim p_x(x)$ distribution over data x
- $z \sim p_z(z)$ prior on z, usually a simple distribution (e.g. Normal distribution)
- Loss
  - $\min_{\theta} \max_{\phi} L(D_\phi, g_\theta) = E_{x\sim p_x(x)}[logD_\phi(x)] + E_{z\sim p_z(z)}[\log\left(1 - D_\phi\left(g_\theta(z)\right)\right)]$
    - $g_\theta : R^q \rightarrow R^n$ mapping from the latent (z) space to the data (x) space
    - $D_\phi : R^n \rightarrow [0,1]$ probability that x comes from the data rather than from the generator g_θ
    - If g_θ is fixed, $L(D_\phi, g_\theta)$ is a classical binary cross entropy for D_ϕ, distinguishing real and fake examples
  - Note:
    - Training is equivalent to find D_{ϕ^*}, g_{θ^*} such that
      - $D_{\phi^*} \in \ arg \max_{\phi} L(D_\phi, g_{\theta^*})$ and $g_{\theta^*} \in \ arg \min_{\theta} L(D_{\phi^*}, g_\theta)$
      - Saddle point problem
        - instability
- Practical training algorithm
    - Alternates optimizing (maximizing) w.r.t. D_ϕ optimizing (minimizing) w.r.t. g_θ

Adversarial training as binary classification
Training GANs

- Training alternates optimization (SGD) on D_ϕ and g_θ
  - In the alternating scheme, g_θ usually requires more steps than D_ϕ+ different batch sizes
- It is known to be highly unstable with two pathological problems
  - Oscillation: no convergence
  - Mode collapse: g collapses on a few modes only of the target distribution (produces the same few patterns for all z samplings)
  - Low dimensional supports (Arjovsky 2017): $p_x$and p_θ may lie on low dimensional manifold that do not intersect.
    - It is then easy to find a discriminator, without p_θ close to p_x
  - Lots of heuristics, lots of theory, but
    - Behavior is still largely unexplained, best practice is based on heuristics

GAN- Adversarial training as binary classification
Equilibrium analysis (Goodfellow et al. 2014)

- The seminal GAN paper provides an analysis of the solution that could be obtained at equilibrium
- Let us define
  - $L(D_\phi, g_\theta) = E_{x \sim p_x(x)}[log D_\phi(x)] + E_{x \sim p_\theta(x)}[\log\left(1 - D_\phi(x)\right)]$
    - □ with $p_x(x)$ the true data distribution and $p_\theta(x)$ the distribution of generated data
      - □ Note that this is equivalent to the $L(D, G)$ definition on the slide before
- If g_θ and D_ϕ have sufficient capacity
  - Computing $\underset{\theta}{argmin}\ g^* = \underset{\theta}{argmin} \max_\phi L\left(D_\phi, g_\theta\right)$
  - Is equivalent to compute
    - □ $g^* = argmin_\theta D_{JS}(p_x, p_\theta)$ with $D_{JS}(,)$ the Jenson-Shannon dissimilarity measure between distributions
    - □ The loss function of a GAN quantifies the similarity between the real sample distribution and the generative data distribution by JSD when the discriminator is optimal

GAN- Adversarial training as binary classification
Equilibrium analysis (Goodfellow et al. 2014)

- If the optimum is reached
  - $D_\phi(x) = \frac{1}{2}$ for all $x \rightarrow$ Equilibrium
- In practice equilibrium is never reached
- Note
  - Maximize $\log\left(D_\phi\left(g_\theta(z)\right)\right)$ instead of minimizing $\log\left(1 - D_\phi\left(g_\theta(z)\right)\right)$ provides stronger gradients and is used in practice, i.e. $\log\left(1 - D_\phi\left(g_\theta(z)\right)\right)$ is replaced by $-\log\left(D_\phi\left(g_\theta(z)\right)\right)$

GAN equilibrium analysis (Goodfellow et al. 2014)
Prerequisite KL divergence

- Kullback Leibler divergence
  - Measure of the difference between two distributions p and q
  - Continuous variables
    - $D_{KL}(p(y)||q(y)) = \int_y \ (\log\frac{p(y)}{q(y)})p(y)dy$
  - Discrete variables
    - $D_{KL}(p(y)||q(y)) = \sum_i \ (\log\frac{p(y_i)}{q(y_i)})p(y_i)$
- Property
  - $D_{KL}(p(y)||q(y)) \geq 0$
  - $D_{KL}(p(y)||q(y)) = 0$ iff $p = q$
    - $D_{KL}(p(y)||q(y)) = -E_{p(y)}\left[log\frac{q(y)}{p(y)}\right] \geq -\log E_{p(y)}\left[\frac{q(y)}{p(y)}\right] \geq 0$
      - where the first inequality is obtained via Jensen inequality
  - note: D_{KL} is asymmetric, symmetric versions exist, e.g. Jensen-Shannon divergence

GAN equilibrium analysis (Goodfellow et al. 2014) - proof

- For a given generator g, the optimal discriminator is
  - $\mathrm{D}^*(x) = \frac{p_{\mathcal{X}}(x)}{p_{\mathcal{X}}(x)+p_\theta\,(x)}$
    - Let $f(y) = a\ log(y) + b\ log(1-y)$, with $a, b, y > 0$
    - $\frac{df}{dy} = \frac{a}{y} - \frac{b}{1-y}, \frac{df}{dy} = 0 \Leftrightarrow y = \frac{a}{a+b}$ and this is a max
    - $Max_D\ L(D,G) = E_{x\sim p_{\mathcal{X}}(x)}[logD(x)] + E_{x\sim p_\theta(x)}[\log\big(1-D(x)\big)]$ is then obtained for:
      - $\mathrm{D}^*(x) = \frac{p_{\mathcal{X}}(x)}{p_{\mathcal{X}}(x)+p_\theta\,(x)}$

GAN equilibrium analysis (Goodfellow et al. 2014) - proof

- Let $C(g) = \max_D L(g, D) = L(g, D^*)$
- It si easily verified that:
  - $C(g) = -\log 4 + KL\left(p_\chi(x); \frac{p_\chi(x)+p_\theta(x)}{2}\right) + KL\left(p_\theta(x); \frac{p_\chi(x)+p_\theta(x)}{2}\right)$
  - Since $KL(p;q) \geq 0$ and $KL(p;q) = 0$ iff $p = q$
    - $C(g)$ is minimum for $p_\theta = p_\chi$ with $D^*(x) = \frac{1}{2}$
    - At equilibrium, GAN training optimises Jenson-Shannon Divergence, $JSD(p;q) = \frac{1}{2}KL\left(p; \frac{p+q}{2}\right) + \frac{1}{2}KL\left(q; \frac{p+q}{2}\right)$ between p_θ and p_χ
- Summary
  - The loss function of a GAN quantifies the similarity between the real sample distribution and the generative data distribution by JSD when the discriminator is optimal
- Note
  - $\frac{p_\chi(x)}{p_\theta(x)} = \frac{p(x|y=1)}{p(x|y=0)} = k\frac{p(y=1|x)}{p(y=0|x)} = k\frac{D^*(x)}{1-D^*(x)}$ with $k = \frac{p(y=0)}{p(y=1)}$
  - The discriminator is used to implicitely measure the discrepancy between the distributions

Training GANs

- Training alternates optimization on D and G
  - In the alternating scheme, G usually requires more steps than D
- It is known to be highly unstable with two pathological problems
  - Oscillation: no convergence
  - Mode collapse: G collapses on a few modes only of the distribution (produces the same few patterns for all z samplings)
  - Low dimensional supports (Arjovsky 2017): p_{data} and p_g may lie on low dimensional manifold that do not intersect. It is then easy to find a discriminator, without training p_g to be close to p_{data}
  - Very large number of papers offering tentative solutions to these problems
    - e.g. recent developments concerning Wasserstein GANs (Arjovsky 2017)
  - This remain difficult and heuristic although various explanation heve been developped (e.g. stability of the generator – related to optimal transport or dynamics of the network – see course on ODE)
- Evaluation
  - What could we evaluate?
  - No natural criterion
    - Very often beauty of the generated patterns!

Objective functions

- A large number of alternative objective functions have been proposed, we will present two examples
  - Least Square GANs
  - Wasserstein GANs

Objective functions – Least Square GANS (Mao et al. 2017)

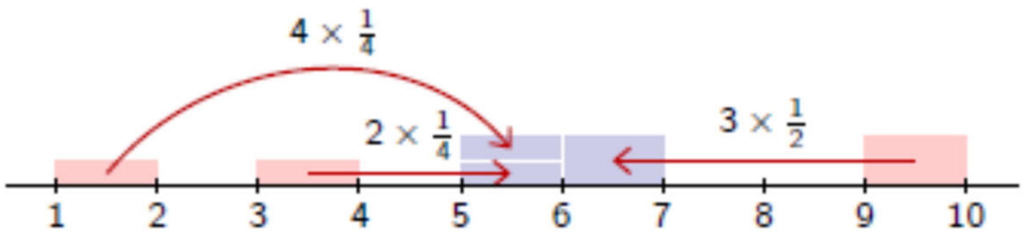
- If a generated sample is well classified but far from the real data distribution, there is no reason for the generator to be updated
- LS-GAN replaces the cross entropy loss with a LS loss which penalizes generated examples by moving them close to the real data distribution.
- The objective becomes
  - $L(D) = E_{x\sim p_X(x)}[(D(x) - b)^2] + E_{z\sim p_z(z)}[(D(g(z)) - a)^2]$
  - $L(g) = \ E_{z\sim p_z(z)}\left[\left(D\left(g(z)\right) - c\right)^2\right]$
  - Where a, b are constants respectively associated to generated and real data and c is a value that g wants D to believe for the generated data.
  - They use for example $a = 0, b = c = 1$

Objective functions – Wasserstein GANs (Arjovski et al. 2017)

- Arjovski advocates that D_{KL} (or D_{JS}) might not be appropriate
- They suggest using the Wasserstein distance between the real and generated distributions (also known as Earth Moving Distance or EMD)
  - Intuitively, this is the minimum mass displacement to transform one distribution to the other
- Wassertein distance is defined as
  - $W(p_{\mathcal{X}}, p_\theta) = \inf_{\gamma \in \Pi(p_{\mathcal{X}}, p_\theta)} E_{(x,x') \sim \gamma}[\| x - x' \|]$
    - where $\Pi(p_{\mathcal{X}}, p_\theta)$ is the set of distributions over X^2 , with $X \subset R^n$ the space of data, whose marginals are respectively $p_{\mathcal{X}}(x)$ and $p_\theta(x)$, $\| x - x' \|$ is the Euclidean norm.
  - Intuitively,
    - $W(,)$ is the minimum amount of work required to transform $p_{\mathcal{X}}(x)$ to $p_\theta(x)$ – see next slide
    - it makes sense to learn a generator g minimizing this metric
      - $g^* = argmin_G W(p_{\mathcal{X}}, p_\theta)$

Wasserstein GANs (Arjovski et al. 2017)

- Earth Mover distance illustration
  - 2 distributions (pink (μ) and blue (μ'))
  - An elementary rectangle weights ¼
  - The figure illustrates the computation of $W(\mu, \mu')$, the Wasserstein distance between pink and blue: this is the earth mover distance to transport pink on blue. This is denoted as $\mu' = \#\mu$, μ' is the push forward of μ



$$\mu = \frac{1}{4}\mathbf{1}_{[1,2]} + \frac{1}{4}\mathbf{1}_{[3,4]} + \frac{1}{2}\mathbf{1}_{[9,10]} \qquad \mu' = \frac{1}{2}\mathbf{1}_{[5,7]}$$

$$W(\mu, \mu') = 4 \times \frac{1}{4} + 2 \times \frac{1}{4} + 3 \times \frac{1}{2} = 3$$

Fig. from F. Fleuret 2018

Objective functions – Wasserstein GANs (Arjovski et al. 2017)

- Let x and y respectively denote the variables from the source and the target distributions
- $p_{\mathcal{X}}(x) = \int_y \gamma(x, y) dy$ is the amount of mass to move from x,

  $p_\theta(y) = \int_y \gamma(x, y) dx$ is the amount of mass to move to y
- Transport is defined as the amont of mass multiplied by the distance it moves, then the transport cost is: $\gamma(x, y). \parallel x - y \parallel$ and the minimum transport cost is $\inf_{\gamma \in \Pi(p_{\mathcal{X}}, p_\theta)} E_{(x,x') \sim \gamma}[\parallel x - x' \parallel]$

Wasserstein GANs (Arjovski et al. 2017)
Optimal Transport interpretation

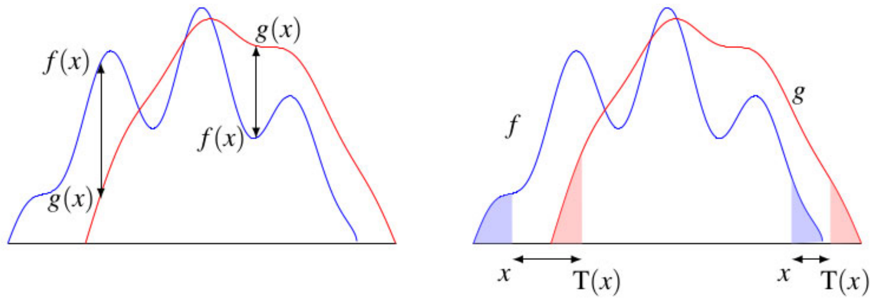


Fig. Santambrogio, 2015

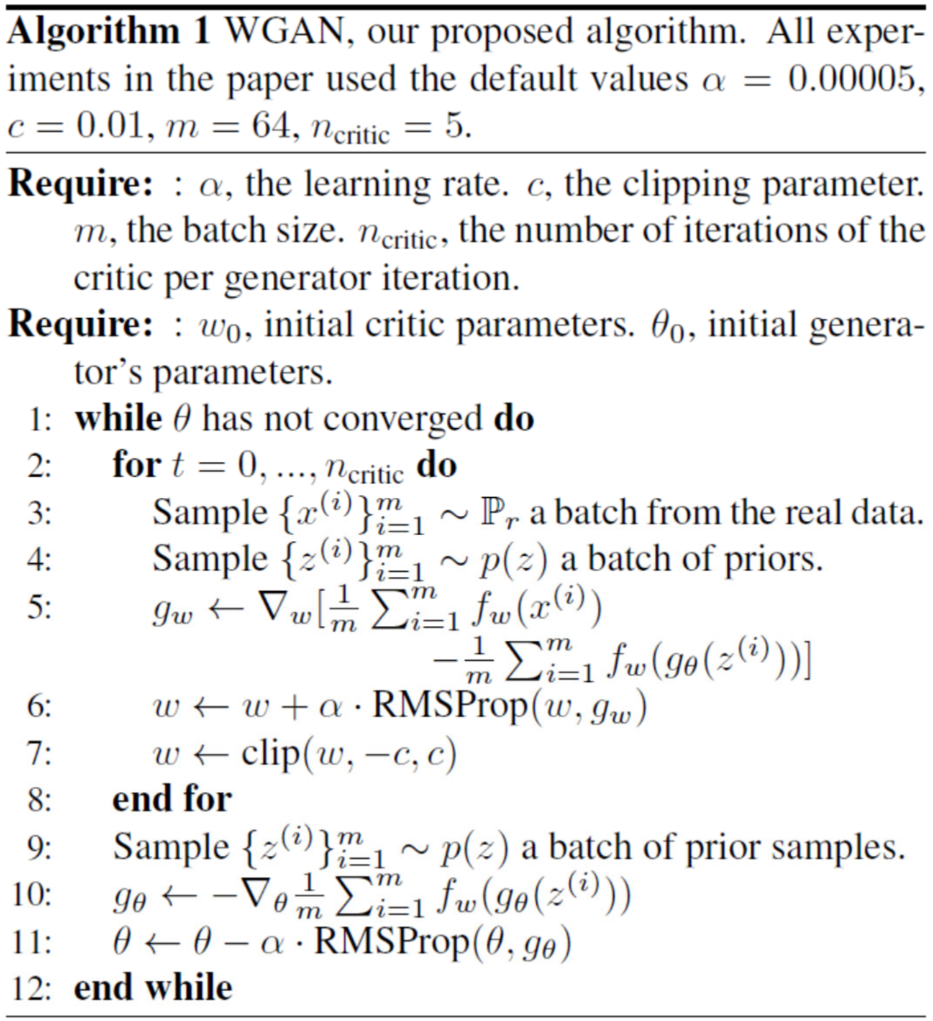
- Left: standard ways to compute distance between functions (point distance)
- Right: Optimal Transport way
  - Seek the best map T which transports the blue distribution on the red one.
  - The smaller T , the closest f and g.
- Wasserstein distance is defined as $W(f,g) = \inf_{T|T\#f=g} \int_x |T(x) - x| dx$
- Which can be translated in:
  - "You look at all the ways to transport f on g with a map T (denoted $T\#f = g$).
  - For a given such transport map T, you look at the total distance you traveled on the x axis , that is $\int_x |T(x) - x| dx$.
  - Among all these transport maps, you look at the one which achieves the optimal (i.e. minimal) distance traveled. This minimal distance is called the Wasserstein distance between f and g."

Wasserstein GANs (Arjovsky et al. 2017)

- The $W(,)$ definition does not provide an operational way for learning G
- Arjovsky uses a duality theorem from Kantorovitch and Rubinstein, stating the following result:
  - $W(p_X, p_\theta) = \sup_{\|f\|_L \le 1} E_{x\sim p_X}|f(x)| - E_{x\sim p_\theta}|f(x)|$
  - Where $f: X \to R$ is 1-Lipchitz, i.e. $|f(x) - f(y)| < 1 \parallel x - y \parallel, \forall\, x, y \in X$
    - i.e. $\parallel f \parallel_L \le 1$ denotes the 1-Lipchitz functions
- Implementation
  - Using this result, one can look for a generator g and a critic f_w:
    - $g^* = argmin_g W(p_X, p_\theta)$
    - $g^* = argmin_g \sup_{\|f\|_L} E_{x\sim p_X}|f_w(x)| - E_{x\sim p_\theta}|f_w(x)|$
    - $g^* = argmin_g \sup_{\|f\|_L} E_{x\sim p_X}|f_w(x)| - E_{z\sim p_z}|f_w(G(z))|$
    - f_w is implemented via a NN with parameters w, it is called a critic because it does not classify but scores its inputs
    - In the original WGAN, f_w is made 1-Lipchitz by clipping the weights (Arjovski et al. 2017)
      - Better solutions were developed later

Wasserstein GANs (Arjovski et al. 2017)

- Algorithm
  - Alternate
    - Optimize f_w
    - Optimize g_θ



Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

```
Require: : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size. $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
Require: : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.
 1: while $\theta$ has not converged do
 2:     for $t = 0, ..., n_{\text{critic}}$ do
 3:         Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
 4:         Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of priors.
 5:         $g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))]$
 6:         $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
 7:         $w \leftarrow \text{clip}(w, -c, c)$
 8:     end for
 9:     Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
10:     $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
11:     $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
12: end while
```

From Arjovski 2017

GANs examples
Deep Convolutional GANs (Radford 2015) - Image generation

- LSUN bedrooms dataset - over 3 million training examples

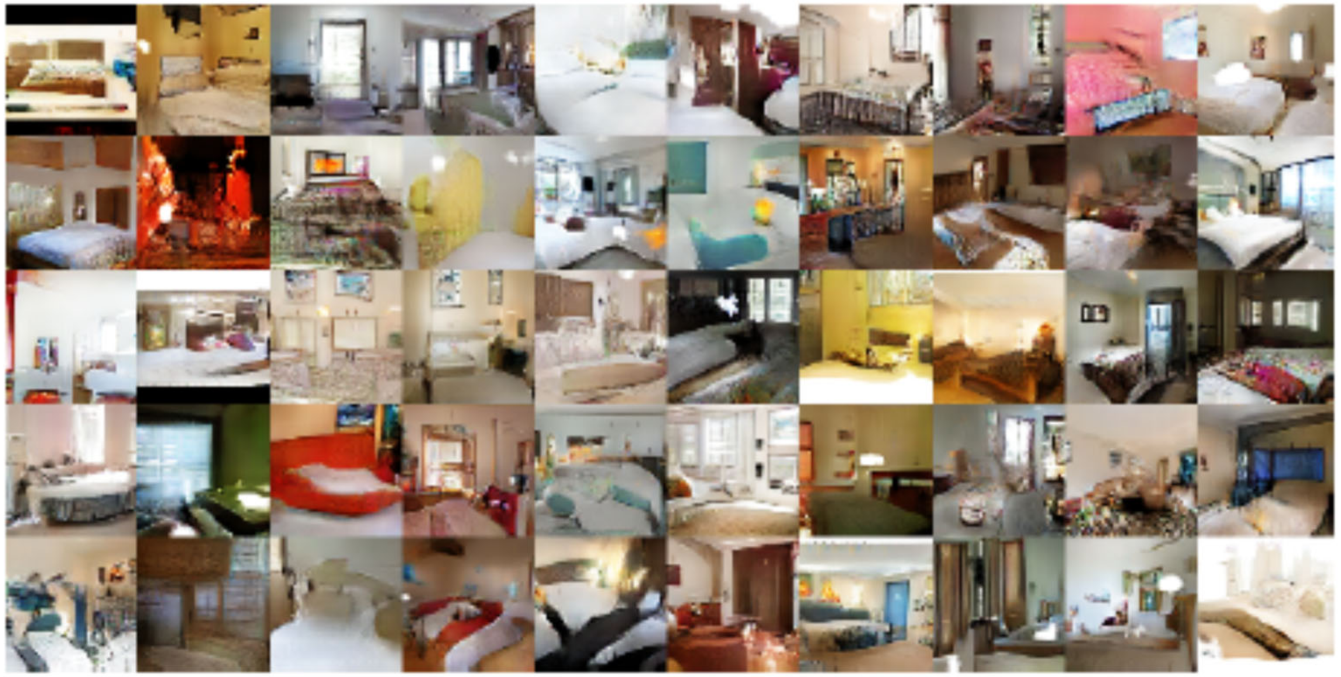
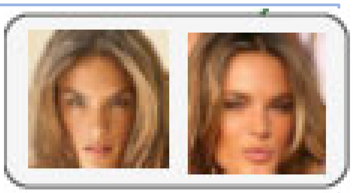


Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

Fig. Radford 2015

Gan example
MULTI-VIEW DATA GENERATION WITHOUT VIEW SUPERVISION (Chen 2018 - Sorbonne)



- Objective
  - Generate images by disantangling content and view
    - Eg. Content 1 person, View: position, illumination, etc
  - 2 latent spaces: view and content
    - Generate image pairs: same item with 2 different views
    - Learn to discriminate between generated and real pairs

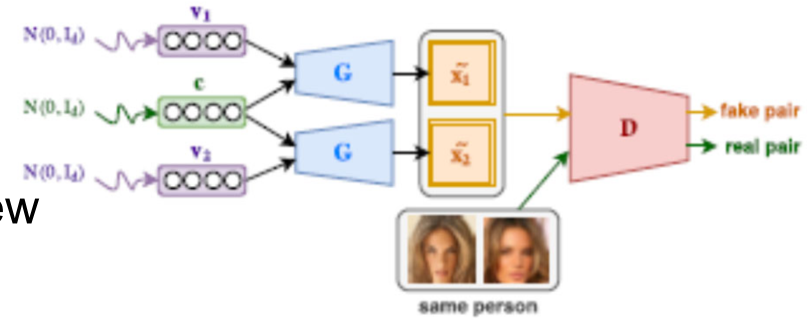
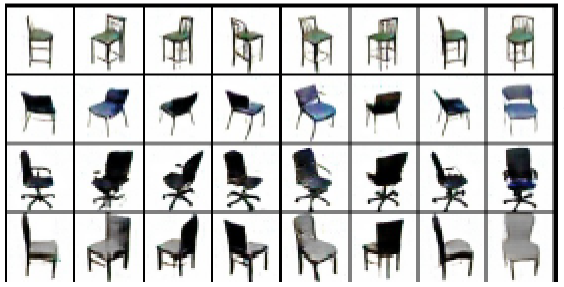


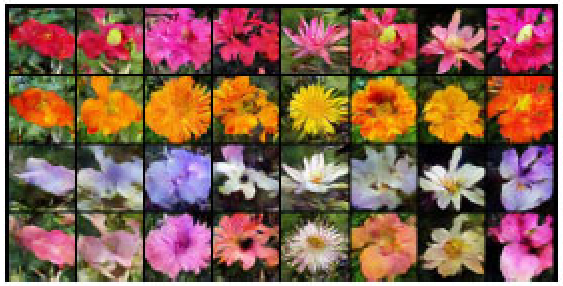
Fig. Chen 2018

Column = view

Column = view



1 row = 1 content



Conditional GANs (Mirza 2014)

- The initial GAN models distributions by sampling from the latent Z space
- Many applications require to condition the generation on some data
  - e.g.: text generation from images, in-painting, super-resolution, etc
- (Mirza 2014) proposed a simple extension of the original GAN formulation to a conditional setting:
  - Both the generator and the discriminator are conditioned on variable y – corresponding to the conditioning data

$$\min_g \max_D L(D, g) = E_{x \sim p_X(x)}[log D(x|y)] + E_{z \sim p(z)}[\log\left(1 - D\left(g(z|y)\right)\right)]$$

Conditional GANs (Mirza 2014)

$$\min_{g}\max_{D} L(D,g) = E_{x\sim p_{\chi}(x)}[logD(x|y)] + E_{z\sim p(z)}[\log\left(1 - D\left(g(z|y)\right)\right)]$$

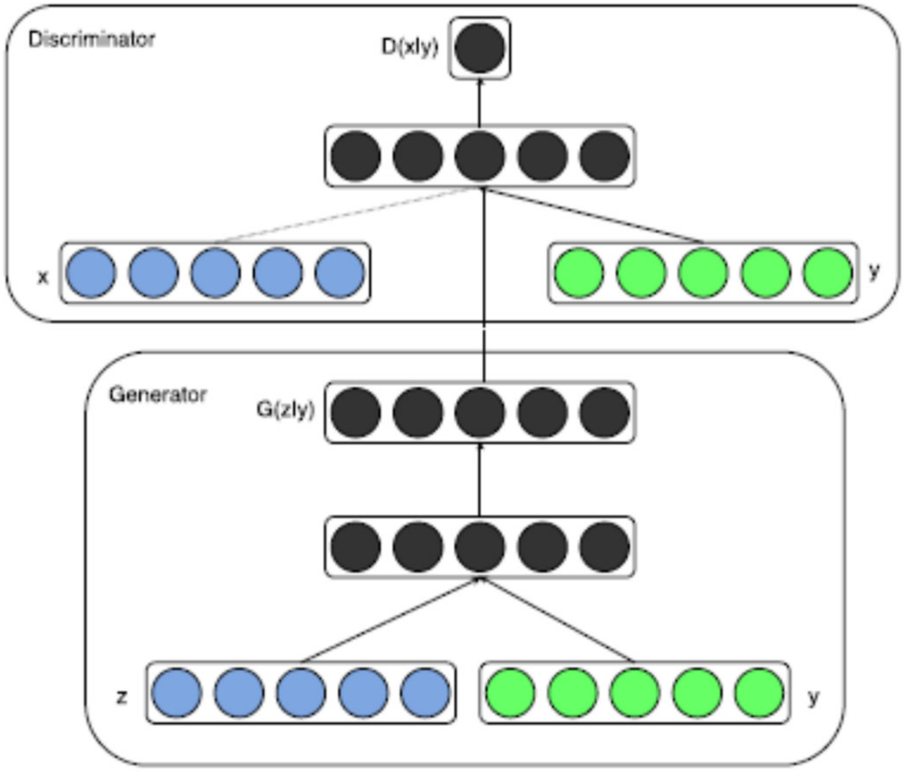


Fig. (Mirza 2014)

Conditional GANs example
Generating images from text (Reed 2016)

- Objective
  - Generate images from text caption
  - Model: GAN conditioned on text input
- Compare different GAN variants on image generation
- Image size 64x64

Fig. from Reed 2016

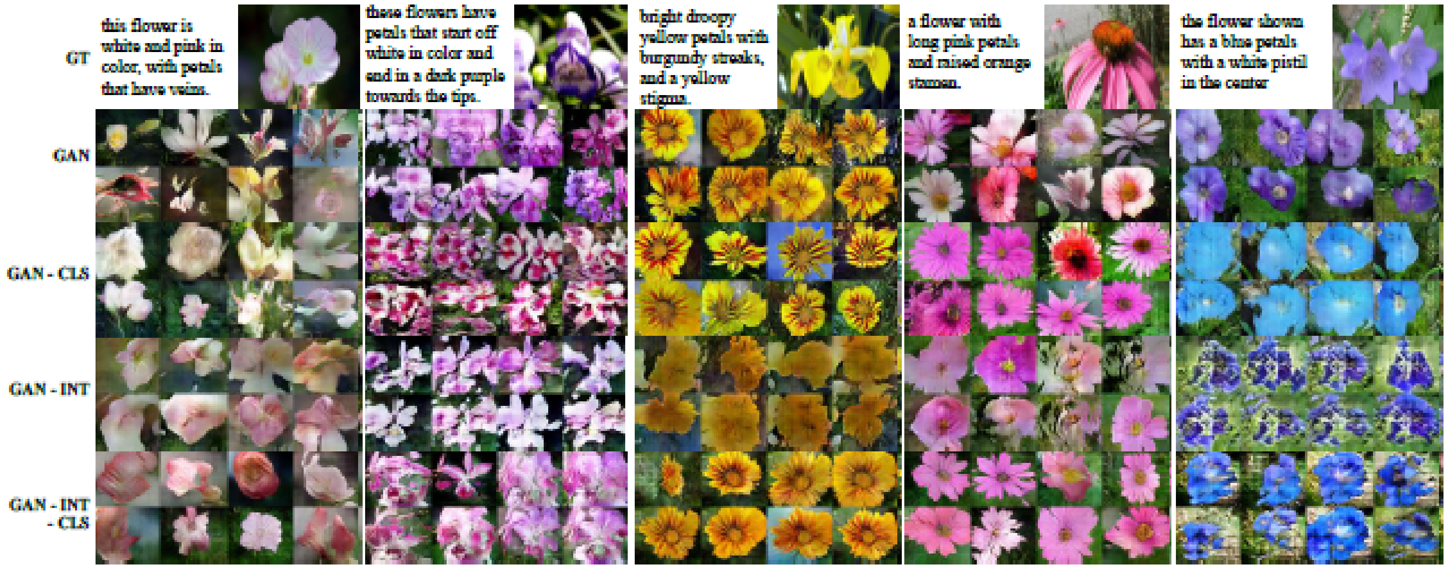
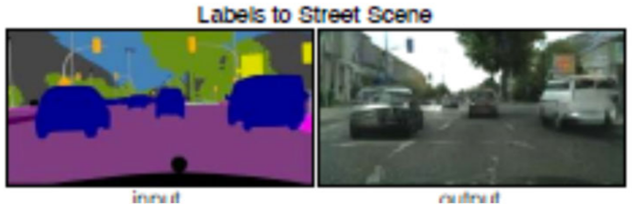
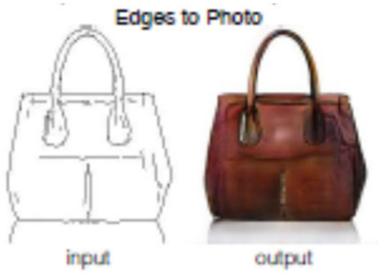

GT
this flower is white and pink in color, with petals that have veins.
these flowers have petals that start off white in color and end in a dark purple towards the tips.
bright droopy yellow petals with burgundy streaks, and a yellow stigma.
a flower with long pink petals and raised orange stamen.
the flower shown has a blue petals with a white pistil in the center
GAN
GAN - CLS
GAN - INT
GAN - INT - CLS

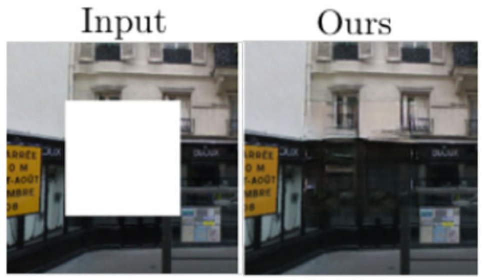
Figure 4. Zero-shot generated flower images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. All variants generated plausible images. Although some shapes of test categories were not seen during training (e.g. columns 3 and 4), the color information is preserved.

Conditional GANs example – Pix2Pix
Image translation with cGANs (Isola 2016)

- Objective
  - Learn to « translate » images for a variety of tasks using a common framework
    - i.e. no task specific loss, but only adversarial training + conditioning
  - Tasks: semantic labels -> photos, edges -> photos, (inpainting) photo and missing pixels -> photos, etc







Conditional GANs example – Pix2Pix
Image translation with cGANs (Isola 2016)

- Loss function
  - Conditional GAN
- $\min_g \max_D L(D,g) = E_{\substack{x\sim p_X(x)\\ y\sim p(y)}}[logD(x,y)] + E_{\substack{z\sim p(z)\\ y\sim p(y)}}[\log(1 - D(g(z,y),y))]$
  - Note: the formulation is slightly different from the conditional GAN model of (Mirza 2014): it makes explicit the sampling on y , but this is the same loss.
- This loss alone does not insure a correspondance between the conditioning variable y and the input data x
  - They add a loss term, its role is to keep the generated data $\mathrm{g}(z,y)$ « close » to the conditioning variable y
  - $L_{L^1}(g) = E_{x,y,z}\|x - g(y,z)\|_1$
    - Where $\|.\|_1$ is the L^1 norm
- Final loss
  - $\min_g(\max_D L(D,g) + \lambda L_{L^1}(g))$

Conditional GANs example – Pix2Pix
Image translation with cGANs – Examples (Isola 2016)

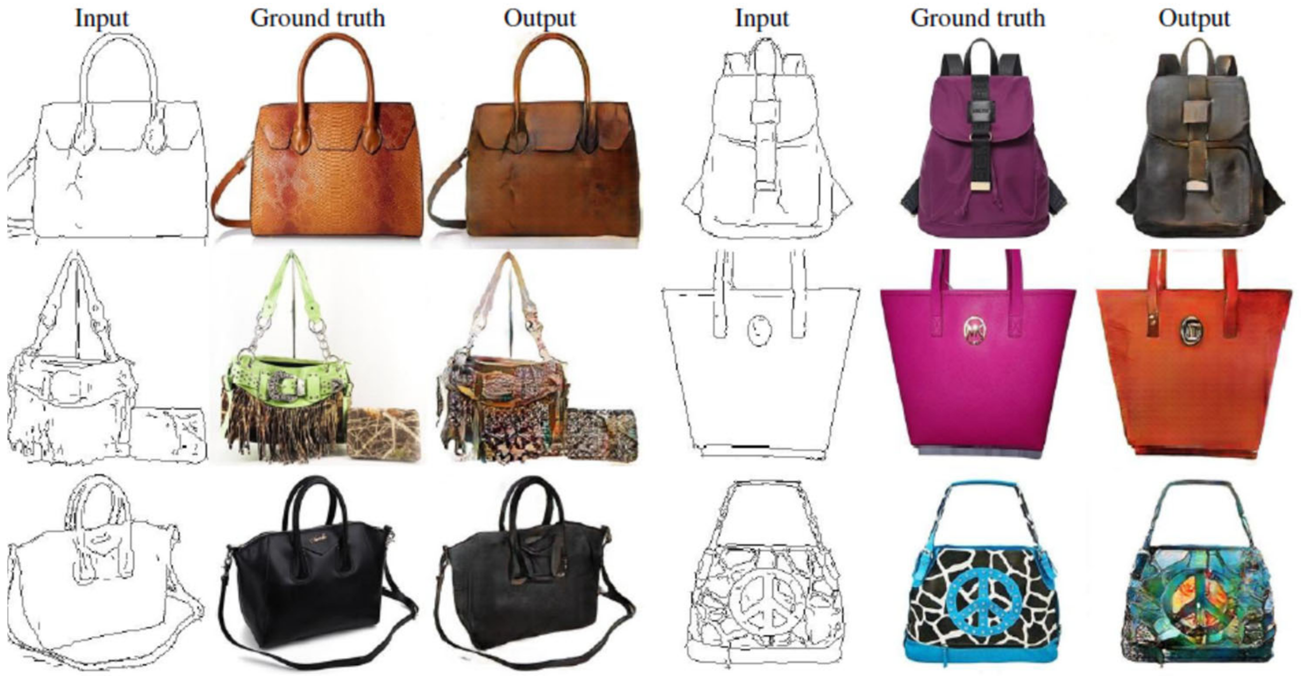


Figure 15: Example results of our method on automatically detected edges→handbags, compared to ground truth.

Fig. (Isola 2016)

Conditional GANs example – Pix2Pix
Image translation with cGANs - Examples - (Isola 2016)

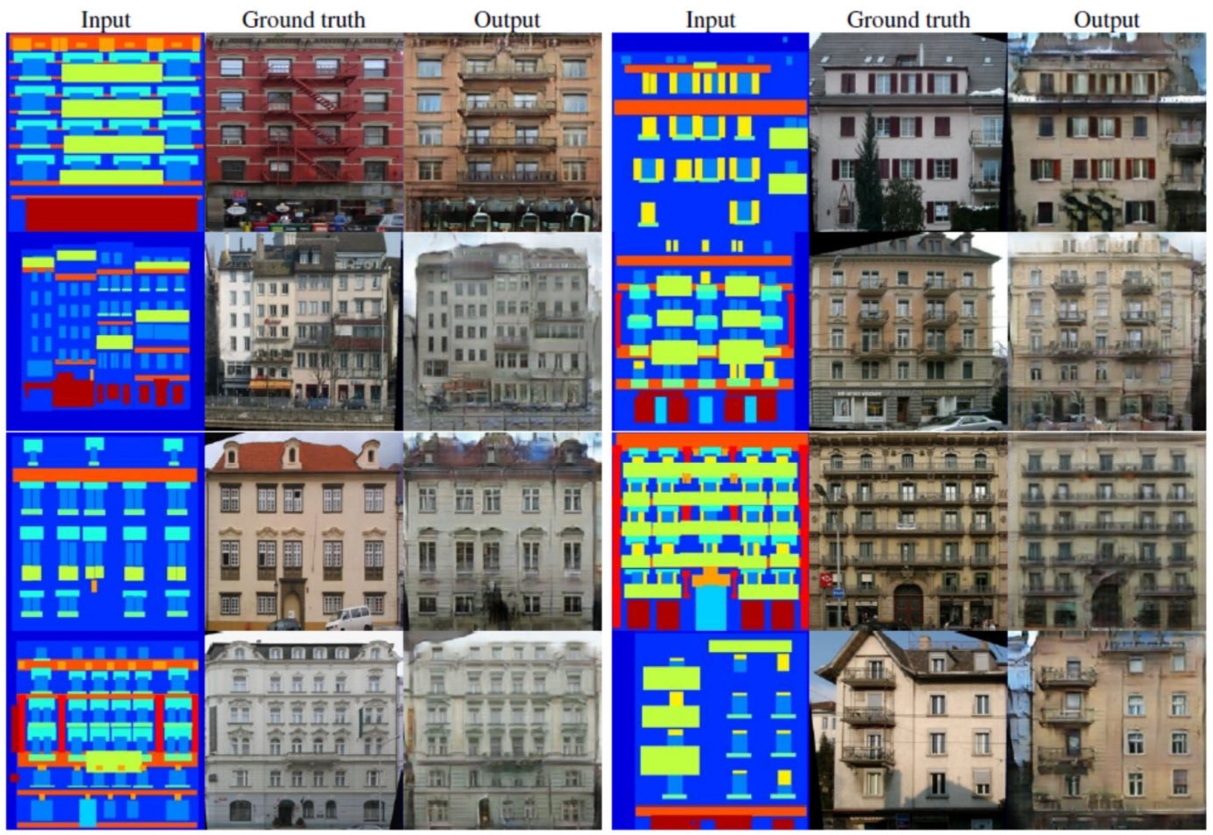


Figure 13: Example results of our method on facades labels→photo, compared to ground truth.

Fig. (Isola 2016)

Conditional GANs example – Pix2Pix
Image translation with cGANs – Examples - (Isola 2016)

- Failure examples

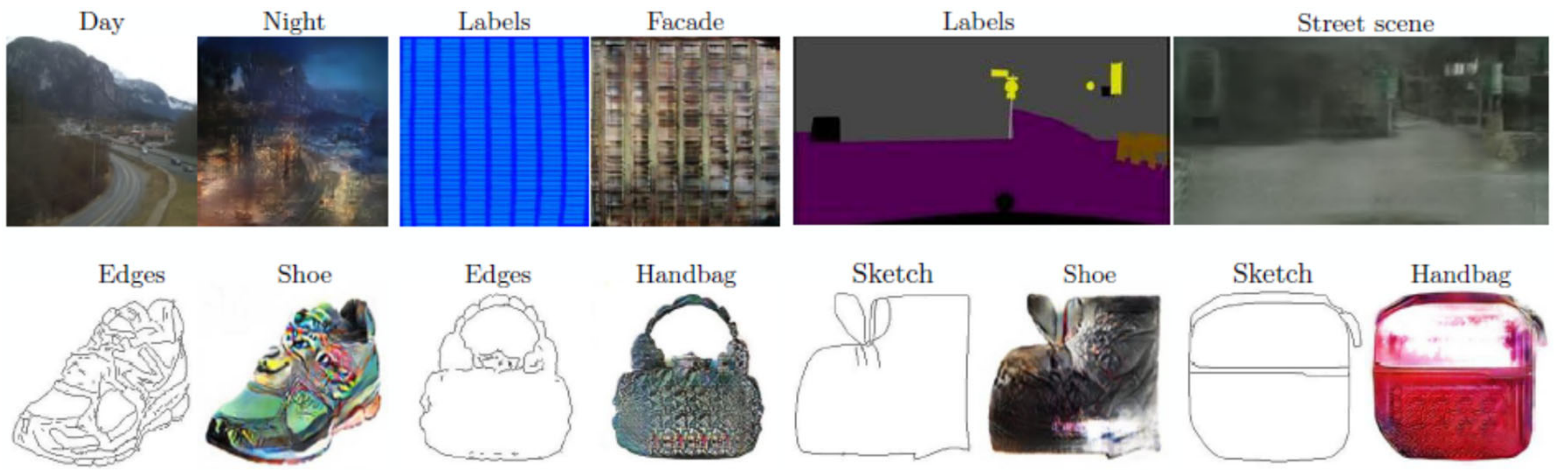


Figure 20: Example failure cases. Each pair of images shows input on the left and output on the right. These examples are selected as some of the worst results on our tasks. Common failures include artifacts in regions where the input image is sparse, and difficulty in handling unusual inputs. Please see `https://phillipi.github.io/pix2pix/` for more comprehensive results.

Fig. (Isola 2016)

Cycle GANs (Zhu 2017)

- Objective
  - Learn to « translate » images without aligned corpora
    - 2 corpora available with input and output samples, but no pair alignment between images
  - Given two unaligned corpora, a conditional GAN can learn a correspondance between the two distributions (by sampling the two distributions), however this does not guaranty a correspondance between input and output
- Approach
  - (Zhu 2017) proposed to add a « consistency » constraint similar to back translation in language
    - This idea has been already used for vision tasks in different contexts
    - Learn two generative mappings
      - $g: X \rightarrow Y$ and $f: Y \rightarrow X$ such that:
      - $f \circ g(x) \simeq x$ and $g \circ f(y) \simeq y$
    - and two discriminant functions D_Y and D_X

Cycle GANs (Zhu 2017)

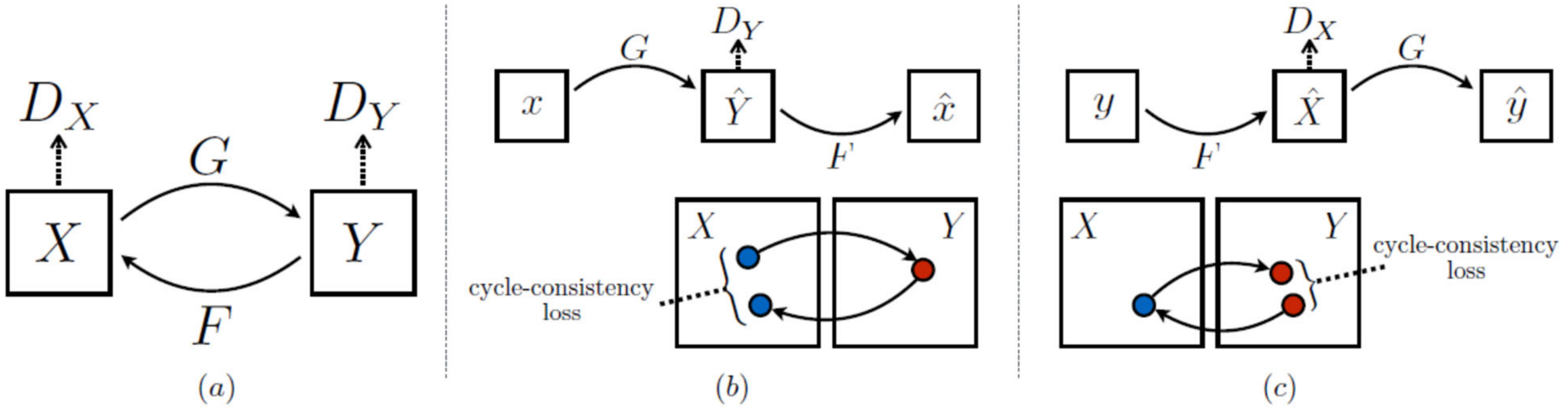


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X, F, and X. To further regularize the mappings, we introduce two "cycle consistency losses" that capture the intuition that if we translate from one domain to the other and back again we should arrive where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

Fig (Zhu 2017)

Cycle GANs (Zhu 2017)

- Training
  - The loss combines two conditional GAN losses (g, D_Y) and (f, D_X) and a cycle consistency loss
  - $L_{cycle}(f, g) = E_{p_X(x)}[\|f(g(x)) - x)\|_1] + E_{p_{data}(y)}[\|g(f(y)) - y)\|_1]$
  - $L(g, D_Y, f, D_X) = L(g, D_Y) + L(f, D_X) + L_{cycle}(f, g)$
  - Note: they replaced the usual $L(g, D_Y)$ and $L(f, D_X)$ term by a mean square error term, e.g.:
    - $L(g, D_Y) = E_{p_Y(y)}[(D_Y(y) - 1)^2] + E_{p_X(x)}[D_Y(G(x))]$

Cycle GANs (Zhu 2017)

- Examples

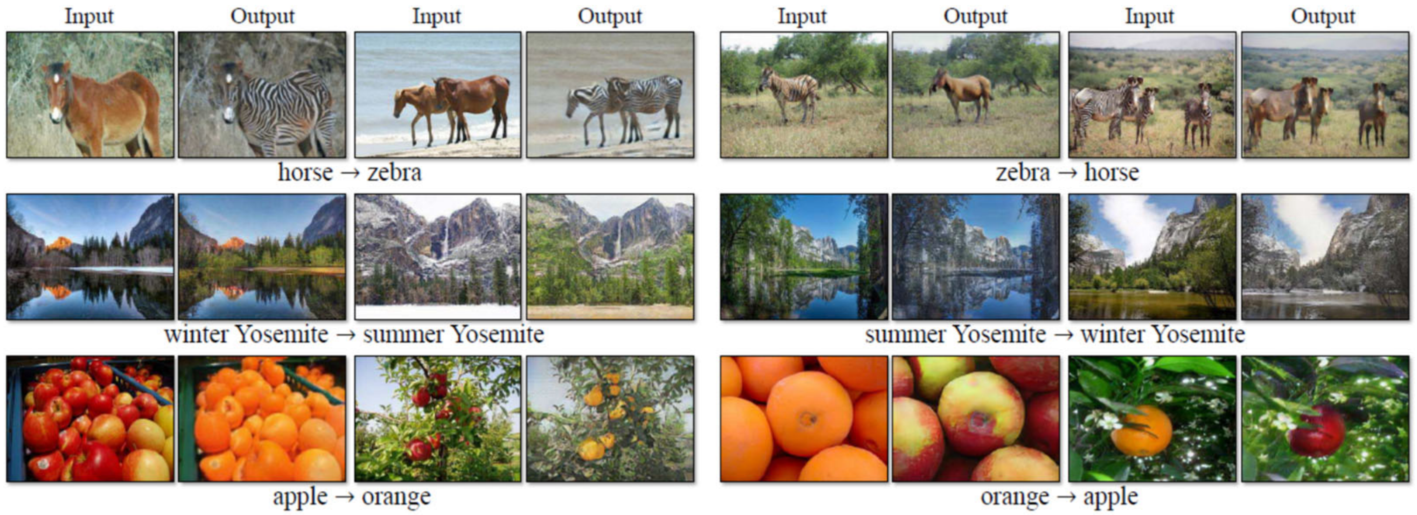


Figure 7: Results on several translation problems. These images are relatively successful results – please see our website for more comprehensive results.

Input Monet Van Gogh Cezanne Ukiyo-e

- Failures

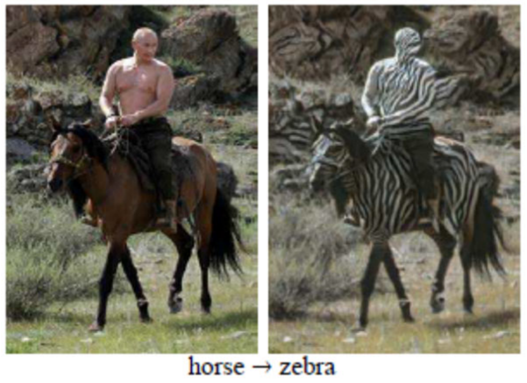
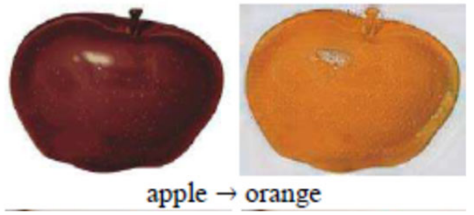
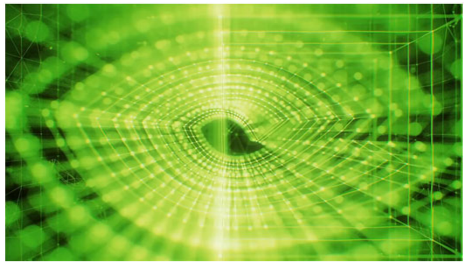


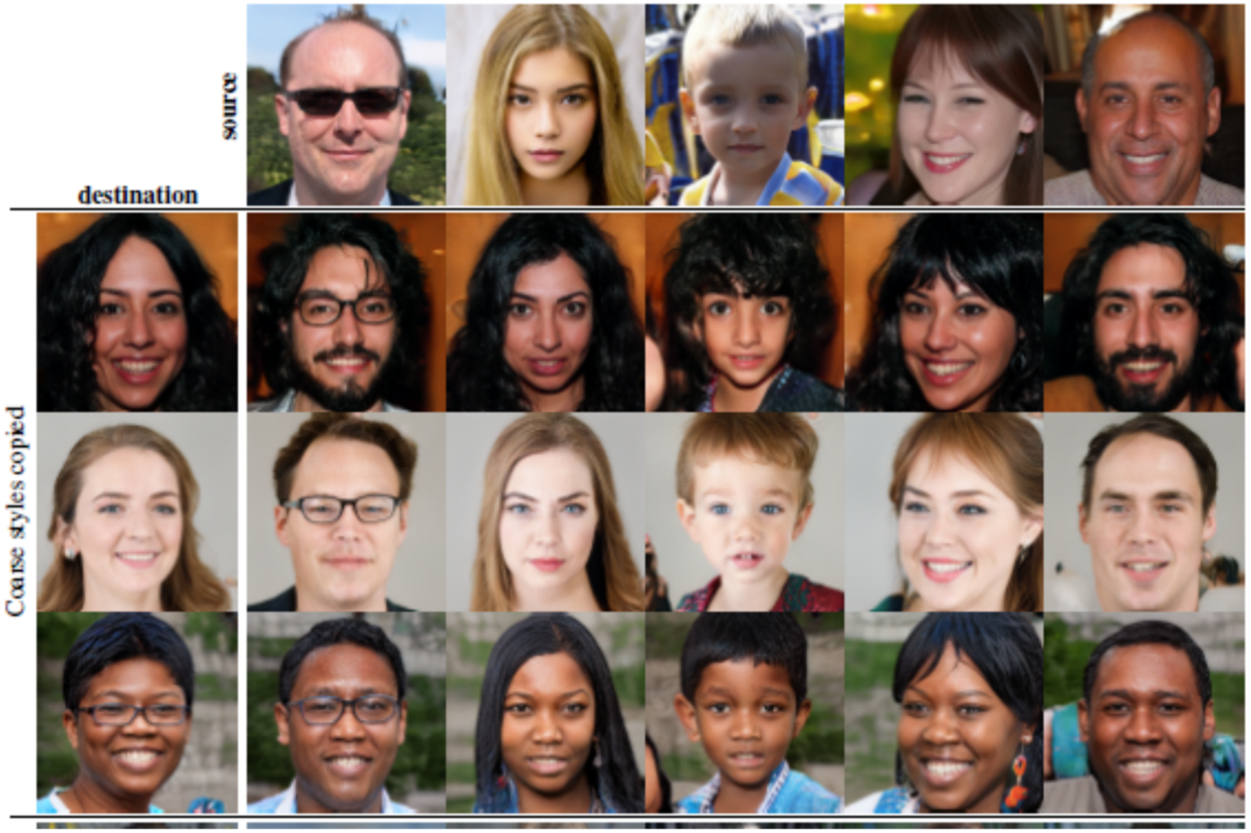
Fig (Zhu 2017)



(Karras et al. 2019) – Style GAN

- (Karras et al. 2019) – Style GAN
- Noyte: now (2020) StyleGAN3: https://nvlabs.github.io/stylegan3/
- https://nvlabs.github.io/stylegan2/versions.html



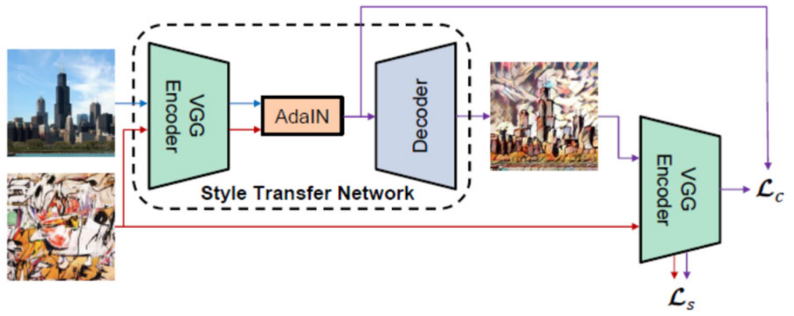


Style Gan
Preliminary: Adaptive Instance Normalization (AdaIN)

- Recall batch normalization
  - $BN(x) = \gamma\left(\frac{x-\mu(x)}{\sigma(x)}\right) + \beta$, here all the quantities are vectors (or tensors) of the appropriate size
  - The mean for channel c is computed as:
    - $\mu_c(x) = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$
    - With N the number of images in the batch, H the height and W the width, i.e. x is of shape $[N, C, H, W]$
    - γ and β are trainable parameters that are different for each channel
    - BN averages over all the images in the batch
      - i.e. all the images in the batch are averaged around a single « style »

Style Gan
Preliminary: Adaptive Instance Normalization (AdaIN)

- Adaptive Instance Normalization (Huang 2017)
  - Idea: inject through the linear transformation defined by γ, β the feature statistics from another image (e.g. its style)
  - Let x (content) and y (style) two images or image transformations
    - $AdaIN(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$
    - This simply replaces the the channel-wise statistics of x by those of y
    - AdaIN can normalize the style of each individual sample to a target style

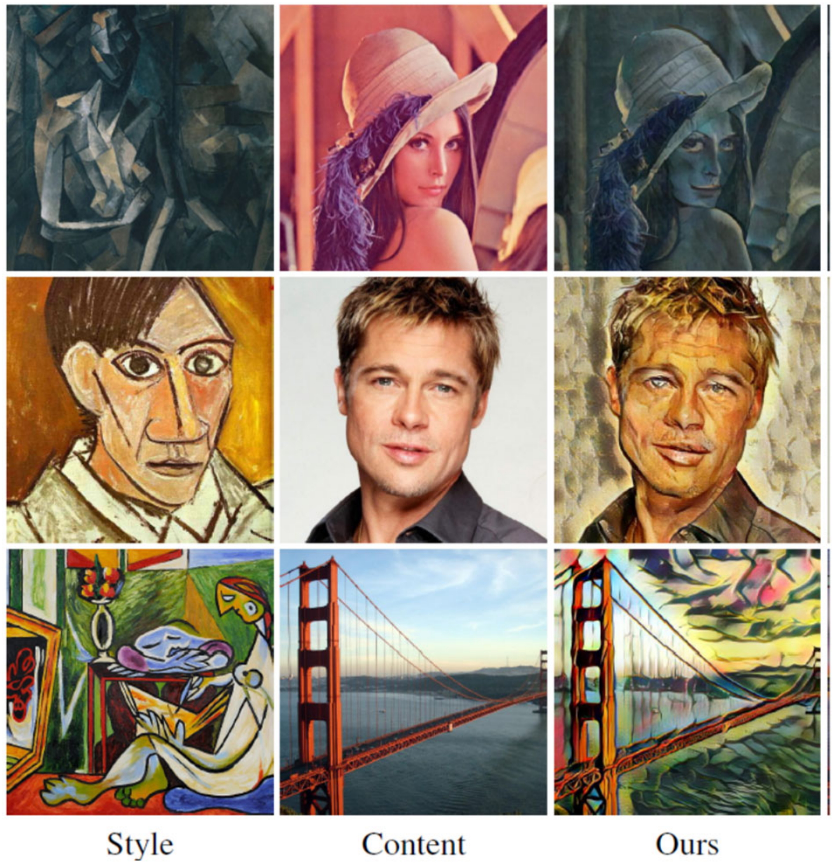


(Huang 2017)

Figure 2. An overview of our style transfer algorithm. We use the first few layers of a fixed VGG-19 network to encode the content and style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is learned to invert the AdaIN output to the image spaces. We use the same VGG encoder to compute a content loss $\mathcal{L}_c$ (Equ. 12) and a style loss $\mathcal{L}_s$ (Equ. 13).

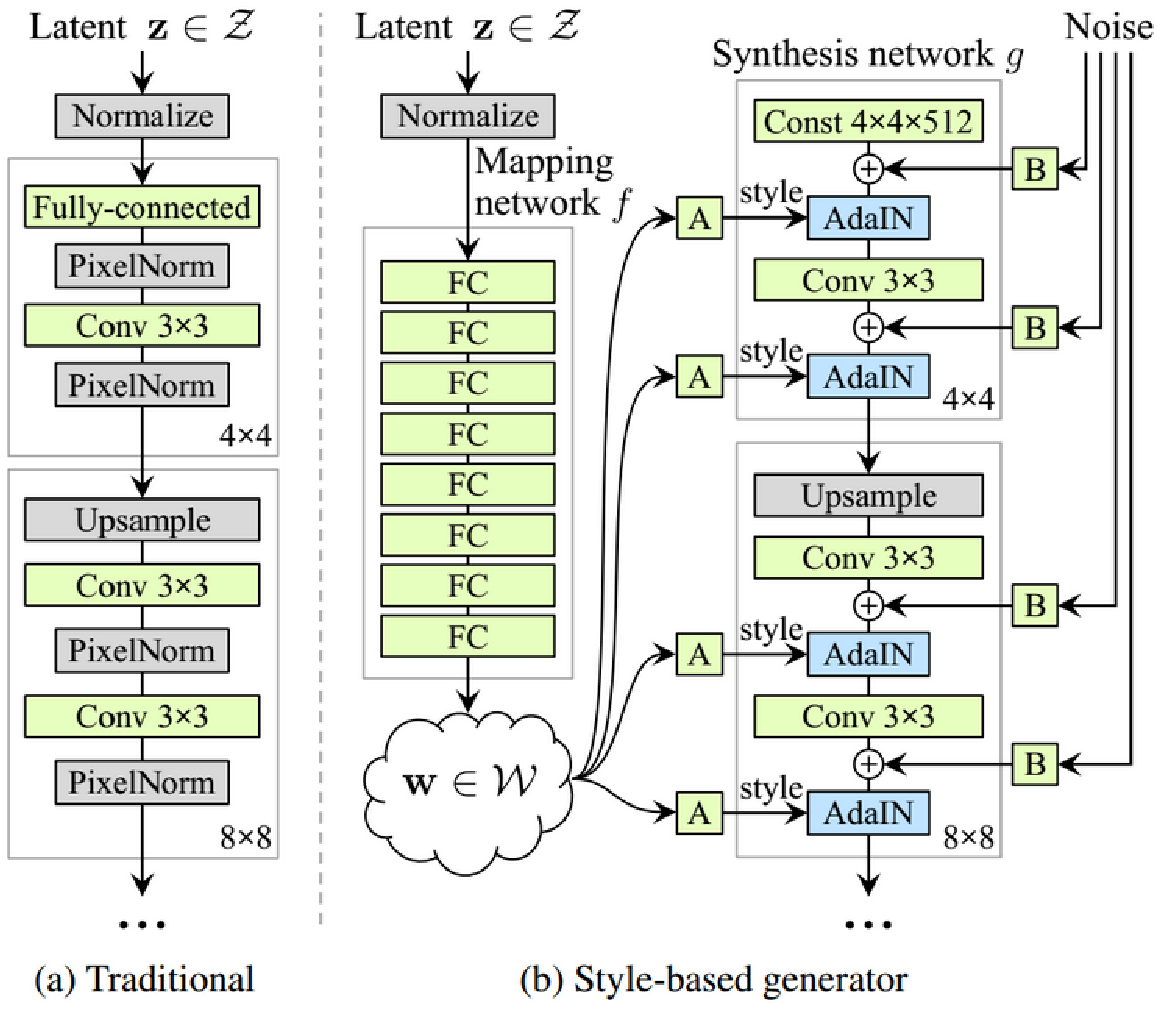
Style Gan
Preliminary: Adaptive Instance Normalization (AdaIN)

- (Huang 2017) examples



Architecture of Style Gan

Karras et al. 2019

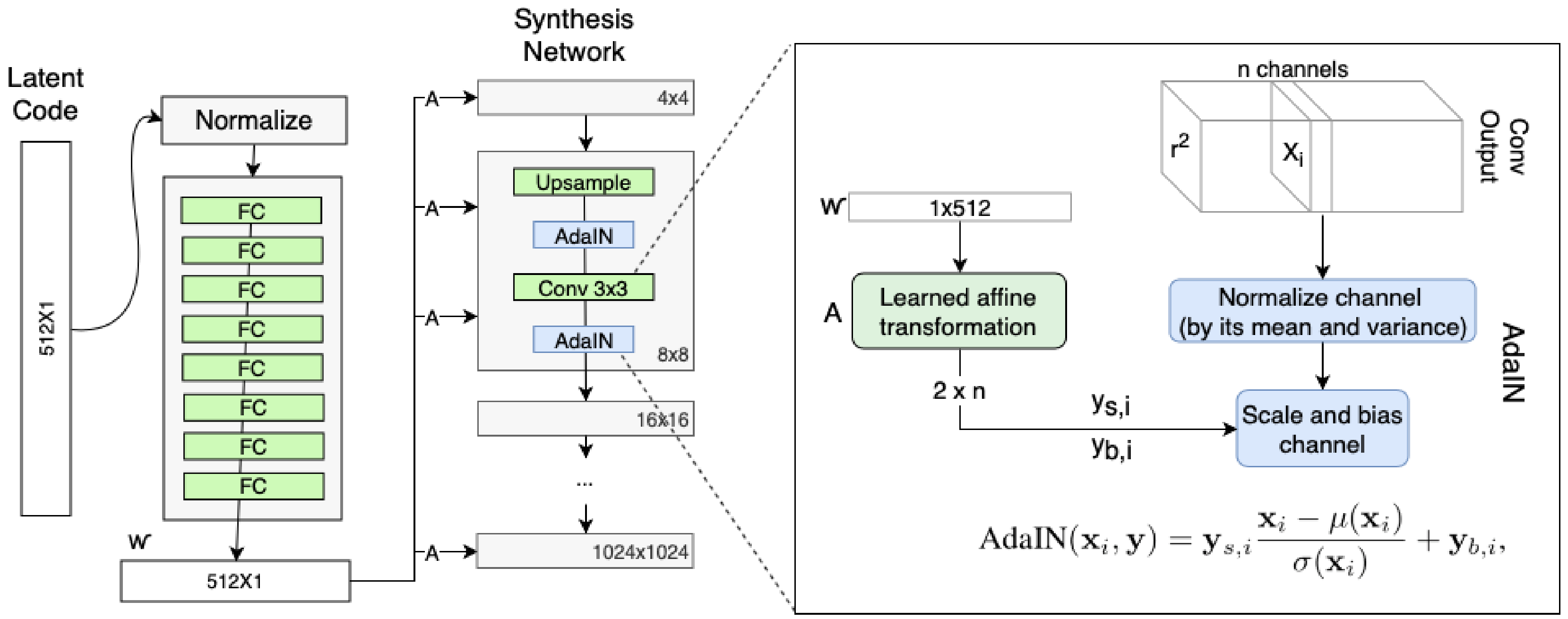


(a) Traditional

(b) Style-based generator

- A mapping network generates a representation vector w
- Affine transformations (A) are trained to compute λ and β vectors for different resolution of the image generator from w – this induces different styles for each resolution
- Noise input are single channel images consisting of uncorrelated Gaussian noise – a single noise image is broadcasted to all the feature maps – this induces stochastic variations

Architecture of Style Gan

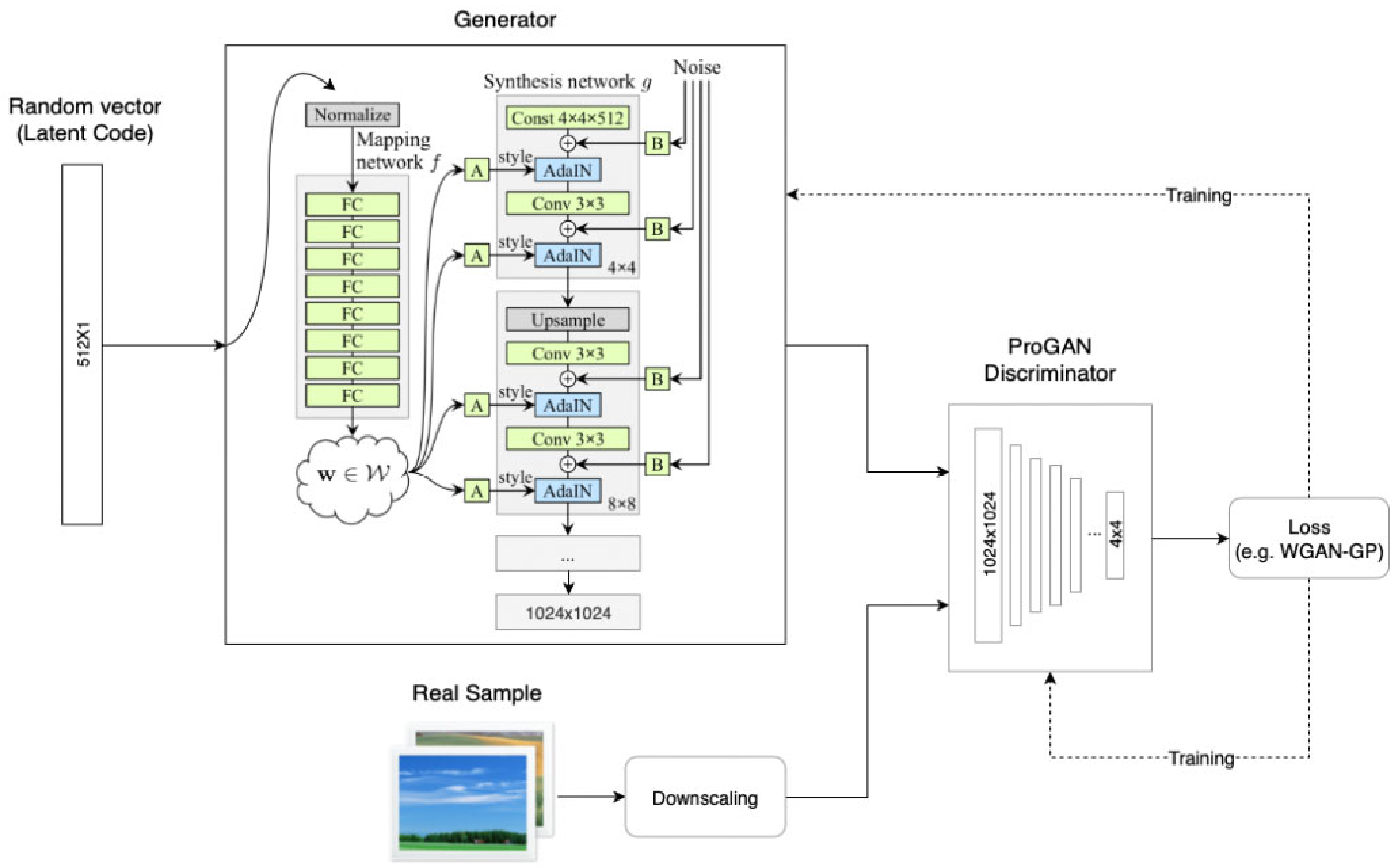


$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

- Affine transformations computed from w

https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431

Architecture of Style Gan



- Global architecture of StyleGAN

https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431

GANs

- Making GANs work is usually hard
- All papers are full of technical details, choices (architecture, optimization, etc.), tricks, not easy to reproduce.

Diffusion models

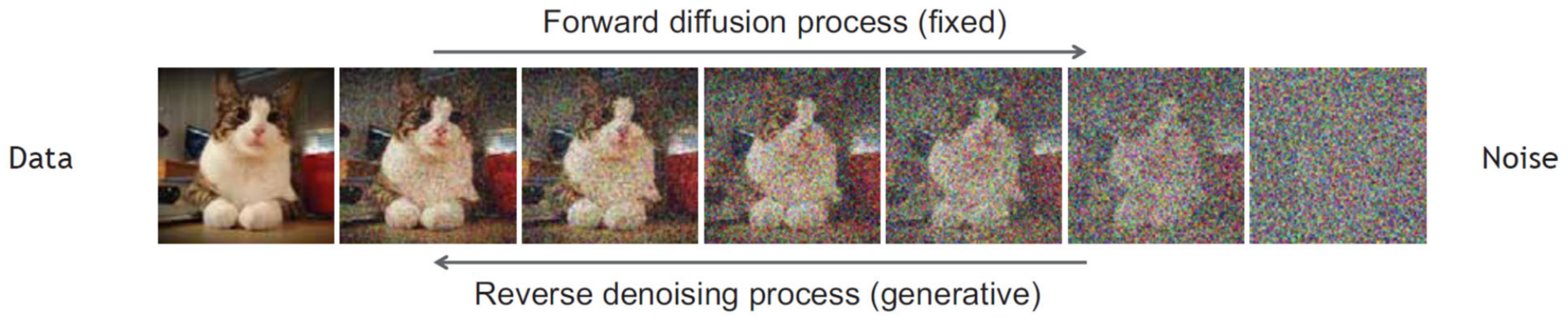
Diffusion models

- Diffusion models emerged in 2019, gained momentum in 2021
- As in 2023, diffusion models are used in several popular large scale models for text to image generation
  - e.g. Imagen https://imagen.research.google/, stable diffusion https://stablediffusionweb.com/, Dall-e-2 https://openai.com/dall-e-2/
  - Generative modeling tasks
    - Continuous space models: Image generation, super resolution, image editing, segmentation; etc.
    - Discrete space models, e.g. applications to text generation
- Several approaches including
  - Discrete time models
    - Denoising Diffusion Probabilistic Models (DDPMs)
    - Score based Generative Models (SGM)
  - Time continuous models
    - Score Based Models with Differential Equations (SGMdiffeq)

Diffusion models

- Diffusion models implement the following idea
  - Forward diffusion

    Gradually add noise to an input image until one get a fully noisy image
  - Reverse denoising
    - Generate data from the target distribution
    - Sample from the noise space and reverse the forward process


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

  - Forward and reverse processes are used for training
  - At inference, generation is performed via the rewverse process

Fig. Kreis et al. 2022

Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models - DDPM

- DDPM are based on two Markov chains
  - A forward chain that adds noise to data –> Forward process
    - Hand designed: transforms any data distribution into a simple prior distribution – here we will use a standard Gaussian for the prior
  - A reverse chain that converts noise to data –> Reverse process
    - The forward chain is reversed by learning **transition kernels** parameterized by neural networks
    - New data are generated by sampling from the simple prior, followed by ancestral sampling through the reverse Markov chain

Denoising Diffusion Probabilistic Models
Forward (diffusion) process

- Data distribution $x_0 \sim q(x_0)$
- The forward MC generates a sequence of random variables $x_1, x_2, \ldots, x_T$ starting at x_0 with **transition kernel** $q(x_t|x_{t-1})$
- Given sufficient steps, $q(x_T)$ will be close to a prior distribution $\pi(x)$, e.g. gaussian distribution with fixed mean and variance

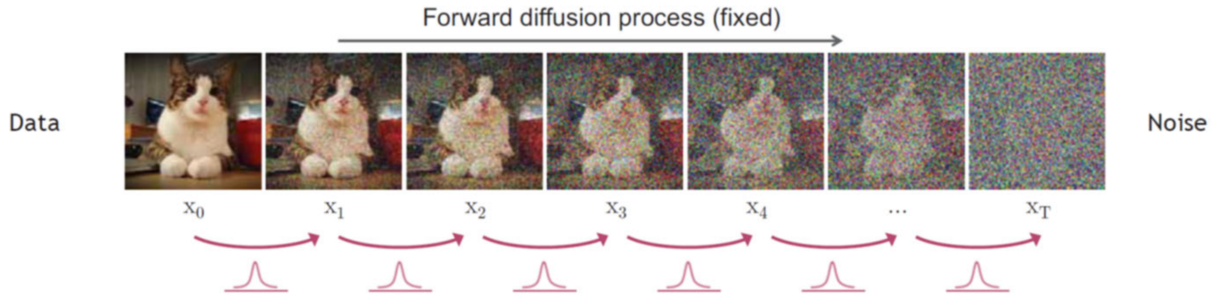


Fig. Kreis et al. 2022

- A typical design for the kernel is a gaussian perturbation $q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}; \beta_t I\right)\ \forall t \in \{1, \ldots, T\}$
  - I is the identity matrix, with the same size as image x_0, $\beta_t \in (0,1)$ is a variance parameter hand fixed or learned, we consider it hand fixed here.
    - β_t is chosen so that $\beta_t < \cdots < \beta_T$, e.g. $T = 2000, \beta_1 = 10^{-4}, \beta_T = 10^{-2}$ with a linear increase
  - Other types of kernels (than gaussians) could be used

Denoising Diffusion Probabilistic Models
Forward (diffusion) process

- The forward diffusion process is then defined as

- $\mathbf{x_0 \sim q(x_0)},$
- $\boldsymbol{q(x_1, \dots, x_T | x_0) = \prod_{t=1}^{T} q(x_t | x_{t-1})},$
- $\mathbf{q(x_t | x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1 - \beta_t} x_{t-1}; \beta_t I\right)} \; \forall \boldsymbol{t} \in \{\mathbf{1}, \dots, \boldsymbol{T}\}$
  - $x_t = \sqrt{\mathbf{1 - \beta_t}}\mathbf{x_{t-1}} + \boldsymbol{\beta_t \epsilon}$ with $\epsilon \sim \mathcal{N}(0, \mathrm{I})$
- $\beta_t \in [0,1]$ is a variance hyperparemeter, $\beta_t < \dots < \beta_T$

Denoising Diffusion Probabilistic Models
Forward process – Diffusion kernel

- Property: the forward process can be sampled at any time t in closed form (derivation next slides)
  - For the gaussian transition kernel
  - $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_O, (1-\bar{\alpha}_t)I)$ – this is called the **diffusion kernel**
  - with $\alpha_t = 1 - \beta_t, \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$
- This allows us to sample $x_t \sim p(x_t)$ using the reparametrization trick
  - Sample $x_0 \sim q(x_0)$ and then sample $\mathrm{x_t} \sim q(x_t|x_0)$ (this is called ancestral sampling)
  - $x_t = \sqrt{\bar{\alpha}_t} x_O + \sqrt{(1-\ \bar{\alpha}_t)}\epsilon$, with $\epsilon \sim \mathcal{N}(0, I), \forall t \sim \mathcal{U}(\{1, \dots, T\})$
  - The schedule for β_t is defined so that $q(x_T|x_0\ ) \approx \mathcal{N}(x_T; 0, I)$

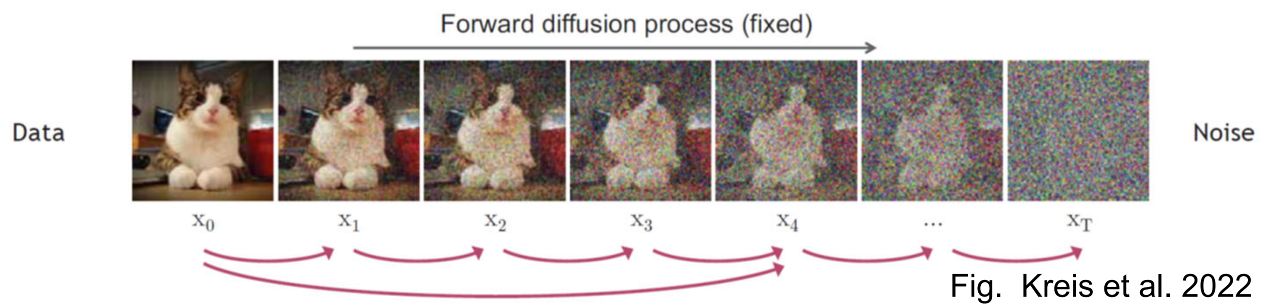


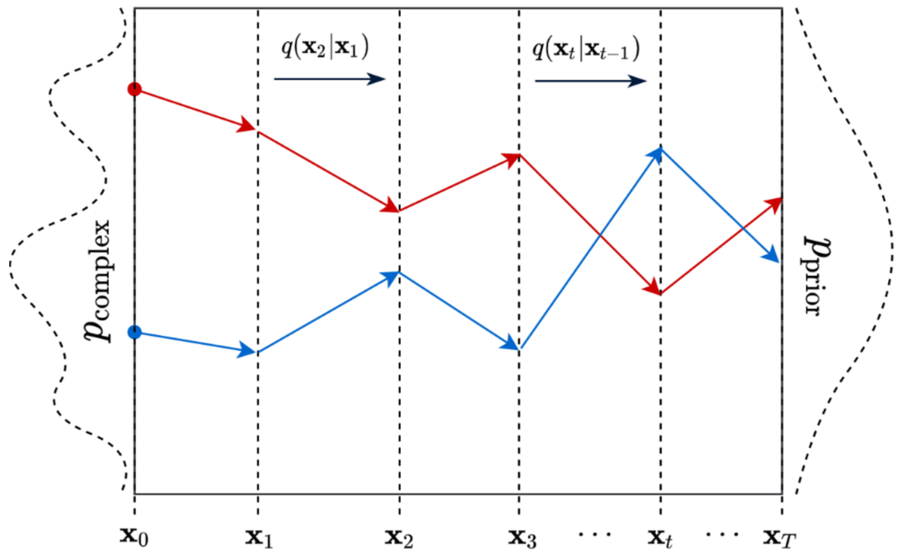
Fig. Kreis et al. 2022

Denoising Diffusion Probabilistic Models
Forward process - Illustration

- Illustration of the forward diffusion process – discrete trajectories in the x space

Fig. Ayan Das 2021

Samples
$x_0 \sim q(x_0)$



Samples
$x_T \sim q(x_T|x_0)$

Denoising Difusion Probabilistic Models – forward process
Diffusion kernel $q(x_t|x_0)$ - derivations

- Closed form for $q(x_t|x_0)$
  - $q(x_t \mid x_0) = N(x_t; \sqrt(\bar{\alpha}_t)x_0, (1 - \bar{\alpha}_t)I)$ with $\alpha_t = 1 - \beta_t, \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$
    - $x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1 - \alpha_t}\epsilon$
      - $x_{t-1} = \sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1 - \alpha_{t-1}}\epsilon$
    - $x_t = \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1 - \alpha_{t-1}}\epsilon) + \sqrt{1 - \alpha_t}\epsilon$
    - $x_t = \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{\alpha_t(1 - \alpha_{t-1})}\epsilon + \sqrt{1 - \alpha_t}\epsilon$
    - $x_t = \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1 - \alpha_t\alpha_{t-1}}\epsilon$ (*)
    - ……
    - $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$

  - (*) Sum of two Gaussians
    - Let x and y two Gaussian random variables with the same dimensionality, $p(x) = \mathcal{N}(\mu_x, \Sigma_x)$ and $p(y) = \mathcal{N}(\mu_y, \Sigma_y)$, then their sum is also Gaussian: $\mathrm{p(x + y)} = \mathcal{N}(\mu_x + \mu_y, \Sigma_x + \Sigma_y)$

Denoising Diffusion Probabilistic Models

- Other quantities related to the forward process
- Marginal distribution $q(x_t)$
- $q(x_t) = \int q(x_t|x_0)q_0(x)dx$
  - Cannot be written in closed form but can be sampled by ancestral sampling: sample from $q_0(x)$ and then trasform by the diffusion kernel $q(x_t|x_0)$
- Conditional distribution $q(x_{t-1}|x_t)$
  - $q(x_{t-1}|x_t)$ is intractable
- Conditional diffusion distribution $q(x_{t-1}|x_t, x_0)$
  - $q(x_{t-1}|x_t, x_0)$ is amenable to a closed form – and will be used for training the decoder – see later

Denoising Diffusion Probabilistic Models
Reverse denoising process

- The reverse MC requires the inversion of the Markov chain
  - Sample x_T from a prior distribution $\mathrm{x_T} \sim p(x_T) = \mathcal{N}(x_T; 0, I)$
  - Iteratively sample $\mathrm{x_t} \sim q(x_{t-1}|x_t)$
- In general, $q(x_{t-1}|x_t)$ is untractable
  - One will learn $p_\theta(x_{t-1}|x_t)$ a parametric approximation of $q(x_{t-1}|x_t)$

Denoising Diffusion Probabilistic Models
Reverse denoising process

- The true reverse distribution $q(x_{t-1}|x_t)$ are complex multimodal distributions, they are approximated as normal disctributions
- The reverse MC is then parameterized by
  - A prior distribution $p(x_T) = \mathcal{N}(x_T; 0, I)$
  - A learnable transition kernel $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$
    - $\mu_\theta(x_t, t)$ is typically implemented via a U-Net, $\mu_\theta(x_t, t)$ is the same size as x_t
    - σ_t^2 can be learned, but in (Ho et al. 2020) it is set to β_t

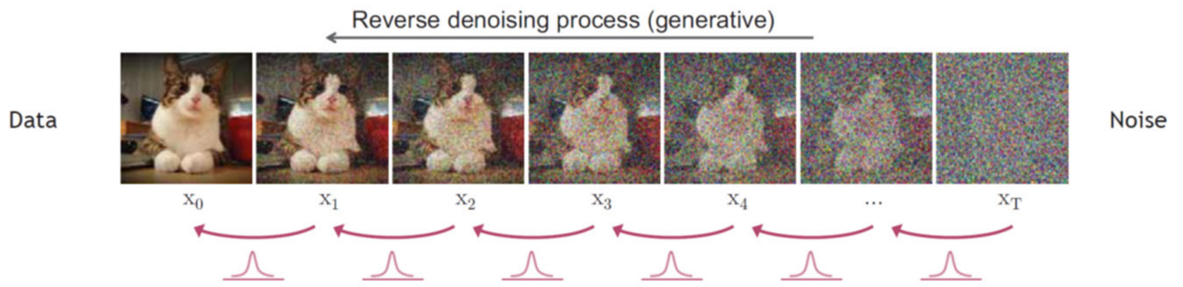


Fig. Kreis et al. 2022

- Reverse factorization: $p_\theta(x_0, \dots, x_T) = p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$
  - We can then generate a data sample x_0 by first sampling a noise vector $x_T \sim p(x_T)$ and then iteratively sampling from the learnable transition kernel $x_{t-1} \sim p_\theta(x_{t-1}|x_t)$ until $t = 1$ where we get $p_\theta(x \quad |x_1)$

Denoising Diffusion Probabilistic Models
Training

- Training amounts at learning the θ parameters:
  - $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)\ t = T, \dots, 1$
  - Ideally, we would like θ so that the probability assigned by the model to each training sample $p_\theta(x_0)$ is maximized, a.k.a. by maximizing the likelihood $E_{q(x_0)}[p_\theta(x_0)]$
    - However this would require marginalizing over all possible (reverse) trajectories to compute it
    - $p_\theta(x_0) = E_{p_\theta(x_1,\dots,x_T)}[p_\theta(x_O, x_1, \dots, x_T)]$

Denoising Diffusion Probabilistic Models
Training

- Instead, one adjusts the parameter θ so that
  - the joint distribution of the reverse MC:
    - $p_\theta(x_0, \dots, x_T) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$
  - matches the distribution of the forward process:
    - $q(x_0, \dots, x_T) = q(x_0) \prod_{t=1}^{T} q(x_t|x_{t-1})$
  - This is achieved by minimizing the Kullback-Leibler divergence between the two distributions
    - $D_{KL}(q(x_0, \dots, x_T)||p_\theta(x_0, \dots, x_T))$

  - Note:
    - This is similar to variational auto-encoders, i.e. this amounts at maximizing a lower bound of the log-likelihood (ELBO)
    - But here this operates on the decoder (reverse diffusion process) and not on the encoder like for VAEs

Denoising Diffusion Probabilistic Models
Training – variational lower bound

$$E_{q(x_0)}[-log p_\theta(x_0)] \leq L$$

with the lower bound (ELBO) L

$$L = E_{q(x_0)q(x_{1:T}|x_0)}[-\log p_\theta(x_0|x_1) + D_{KL}\big(q(x_T|x_0) \parallel p(x_T)\big) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))]$$

- Let us examine the three terms of the lower bound L
  - $D_{KL}\big(p(x_T|x_0) \parallel p(x_T)\big)$
    - does not depend on parameters θ and can be ignored during training
  - $p_\theta(x_0|x_1)$
    - is modeled (Ho et al. 2020) as a separate discrete decoder (not detailed here)
  - $D_{KL}\big(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)\big)$ - (proofs next slides)
    - $q(x_{t-1}|x_t, x_0)$ is a tractable gaussian distribution
    - $p_\theta(x_{t-1}|x_t)$ is also a gaussian distribution
    - $D_{KL}\big(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)\big)$ can then be computed in a closed form
    - It reduces to a simple form

Denoising Diffusion Probabilistic Models
Training

- Let us consider the KL term $D_{KL}\big(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)\big)$
  - It can be shown that $q(x_{t-1}|x_t, x_0) = \mathcal{N}\big(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I\big)$, with:
    - $\tilde{\mu}(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t$ and $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$
      - Recall that $x_t = \sqrt{\bar{\alpha}_t} x_O + \sqrt{(1 - \ \bar{\alpha}_t)}\epsilon$ for $\epsilon \sim \mathcal{N}(0, I)$
      - and $\alpha_t = 1 - \beta_t, \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$
  - Then $\tilde{\mu}(x_t, x_0)$ can be rewriten in a simplified form as:
    - $\tilde{\mu}(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon)$

Denoising Diffusion Probabilistic Models
Training

- $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$
- Both $q(x_{t-1}|x_t, x_0)$ and $p_\theta(x_{t-1}|x_t)$ being Gaussian, the KL divergence writes as

$$\boxed{\begin{aligned} &E_{q(x_0),q(x_t|x_0)}[D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)])\\ &= E_{q(x_0),q(x_t|x_0)}\left[\frac{1}{2\sigma^2}\|\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)\|^2\right] + cte \end{aligned}}$$

- We would like to train $\mu_\theta(x_t, t)$ to approximate $\tilde{\mu}(x_t, x_0)$
  - How to do that: next slide

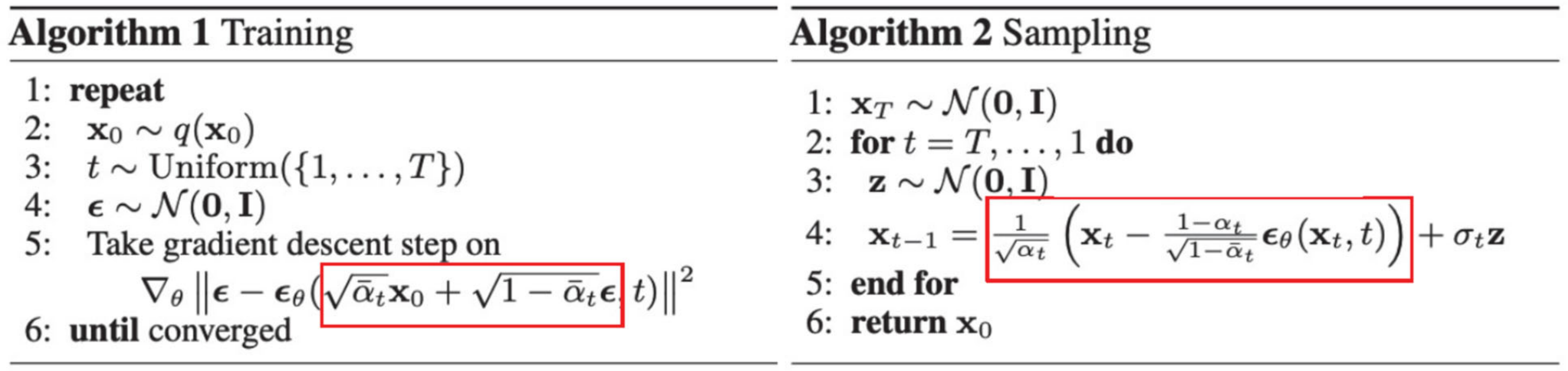
Denoising Diffusion Probabilistic Models
Training

- We would like to train $\mu_\theta(x_t, t)$ to approximate $\tilde{\mu}(x_t, x_0)$
  - i.e. $\mu_\theta(x_t, t)$ must approximate $\tilde{\mu}(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon)$
  - x_t is available as input at training time, (Ho et al. 2020) propose the following noise prediction parametrization
  - $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t))$
  - i.e. parametrize the gaussian noise term $\epsilon_\theta(x_t, t)$ to make it predict ϵ from the input x_t at time t
    - Note: parametrizing $\epsilon_\theta(x_t, t)$ is just another way to parametrize $\mu_\theta(x_t, t)$, but it has been found more efficient experimentally
- With this parametrization, the loss term
  - $L_{t-1} = E_{q(x_0), q(x_t|x_0)}[D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)])$ writes
  - $L_{t-1} = E_{x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0,1)}[\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\alpha_t)}\left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_O + \sqrt{(1 - \bar{\alpha}_t)}\epsilon \, , t\right)\right\|^2]$+Cte
- This is simplified in Ho et al. 2020 (heuristic), so that the global loss L writes as

$$\boxed{L = E_{x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0,1), t \sim \mathcal{U}(1,T)}[\left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_O + \sqrt{(1 - \bar{\alpha}_t)}\epsilon \, , t\right)\right\|^2]}$$

  - with $\mathcal{U}(1,T)$ a uniform distribution

Denoising Diffusion Probabilistic Models
Training and sampling algorithms



Algorithm 1 Training

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

Algorithm 2 Sampling

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

Fig. Ho et al 2020

Denoising Diffusion Probabilistic Models Implementation

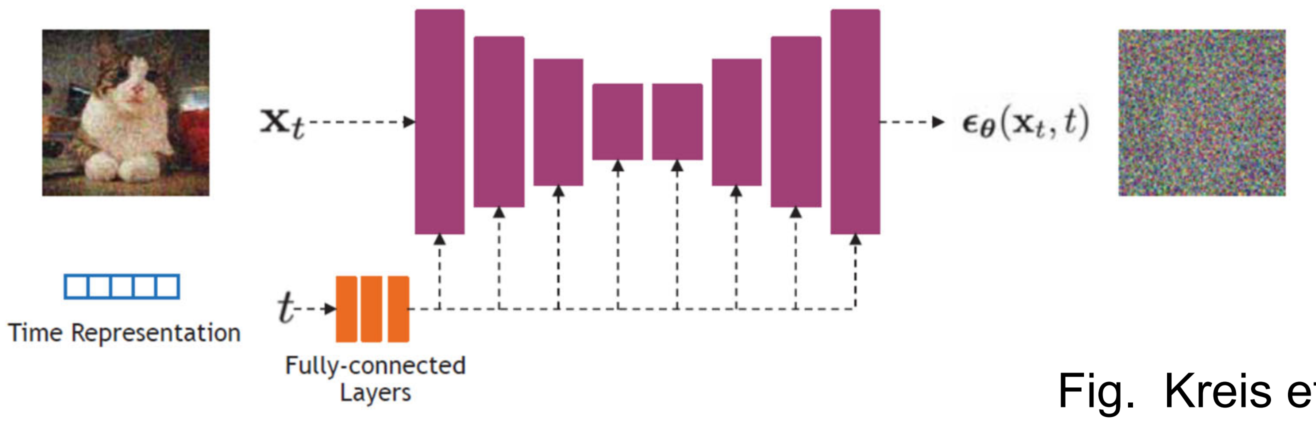


Fig. Kreis et al. 2022

- $\epsilon_\theta(x_t, t)$ is often implemented with a U-Net with ResNet blocks and self attention layers (recent implementations have been proposed with transformers)
- Time features are fed to residual blocks, time encoding follows the transformers sinusoidal position embedding
- The parameters are shared for all the time steps, only the time representation makes the difference between the time steps

Denoising Diffusion Probabilistic Models
Comments

- In Ho et al. 2020
  - $T = 1000, \beta_1 = 10^{-4}, \beta_T = 0.02$ with a linear schedule
  - The pixel values are normalized in $[-1,1]$
  - As usual, lots of influential architecture/ algorithmic parameters conditioning the good behavior of the model
  - The process of generation is **extremely** slow (the original model takes up to 20 h to generate 50k images of size 32x32)
- Several variants/ improvements proposed since the Ho et al. 2020 paper
  - Conditional models allow to generate e.g. images conditionned on text
  - Latent diffusion models (Rombach et al. 2022) perform diffusion in a latent space, accelarating the generation (used e.g. in stable diffusion)
    - The image is first encoded in a smaller diemensional latent space and decoded in order to produce the generated image in the original space
    - Diffusion and denoising happen in the latent space
    - The model allows for conditioning image generation (on text, classes, …)
  - Faster models, such as DDIM (Denoising Diffusion Implicit Models, Song et al. 2021)

Denoising Diffusion Probabilistic Models
ELBO Derivations

- We first show
  - $-E_{q(x_0)}[\log p_\theta(x_0)] \leq E_{q(x_{0:T})}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}] \triangleq L$
- and then
  - $L = E[-log p_\theta(x_0|x_1) + D_{KL}\left(q(x_T|x_0) \parallel p(x_T)\right) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))]$

Denoising Diffusion Probabilistic Models
ELBO Derivations

- $-E_{q(x_0)}[\log p_\theta(x_0)] \leq E_{q(x_0:T)}[\log \frac{q(x_{1:T} \,|x_0)}{p_\theta(x_{0:T})}] \triangleq L$
- Proof
  - $-\log p_\theta(x_0) \leq -\log p_\theta(x_0) + D_{KL}(q(x_{1:T}|x_0) \parallel p_\theta(x_{1:T}|x_0))$
  - $-\log p_\theta(x_0) \leq -\log p_\theta(x_0) + E_{x_{1:T} \sim q(x_{1:T}|x_0)}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})/p_\theta(x_0)}]$
  - $-\log p_\theta(x_0) \leq -\log p_\theta(x_0) + E_{x_{1:T} \sim q(x_{1:T}|x_0)}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})} + \log p_\theta(x_0)]$
  - $-\log p_\theta(x_0) \leq E_{x_{1:T} \sim q(x_{1:T}|x_0)}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}]$
  - $-E_{q(x_0)}[\log p_\theta(x_0)] \leq E_{x_{0:T} \sim q(x_{0:T})}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}]$

Denoising Diffusion Probabilistic Models
ELBO Derivations

- $L = E_{q(x_0:T)}[-\log p_\theta(x_0|x_1) + D_{KL}\big(q(x_T|x_0) \parallel p(x_T)\big) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))]$

- Proof
  - $L = E_{q(x_0:T)}[\log \frac{q(x_{1:T}\ |x_0)}{p_\theta(x_{0:T})}]$
  - $L = E_{q(x_0:T)}[-\log p(x_T) + \sum_{t=1}^{T} \log \frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)}]$
  - $L = E_{q(x_0:T)}\left[-\log p(x_T) + \sum_{t=2}^{T} \log \frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}\right]$
  - $L = E_{q(x_0:T)}[-\log p(x_T) + \sum_{t=2}^{T} \log(\frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} \cdot \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)}) + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}]$
  - $L = E_{q(x_0:T)}[-\log p(x_T) + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} + \sum_{t=2}^{T} \log \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}]$
  - $L = E_{q(x_0:T)}[-\log p(x_T) + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_T|x_0)}{q(x_1|x_0)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)}]$
  - $L = E_{q(x_0:T)}\left[\log \frac{q(x_T|x_0)}{p(x_T)} + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} - \log p_\theta(x_0|x_1)\right]$
  - $L = E_{q(x_0:T)}[-\log p_\theta(x_0|x_1) + D_{KL}\big(q(x_T|x_0) \parallel p(x_T)\big) + \sum_{t>1} D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))]$

Denoising Diffusion Probabilistic Models
ELBO Derivations

- $q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I\right)$ with
  - $\tilde{\mu}(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon)$
  - $q(x_{t-1}|x_t, x_0) = q(x_t|x_{t-1}, x_0)\,\frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}$
  - $q(x_{t-1}|x_t, x_0) \propto \exp -\frac{1}{2}(\frac{(x_t-\sqrt{\alpha_t}x_{t-1})^2}{\beta_t} + \frac{\left(x_{t-1}-\sqrt{\bar{\alpha}_{t-1}}x_0\right)^2}{1-\bar{\alpha}_{t-1}} - \frac{\left(x_t-\sqrt{\bar{\alpha}_t}x_0\right)^2}{1-\bar{\alpha}_t})$
  - … to be completed
-

Score based models

Score based models

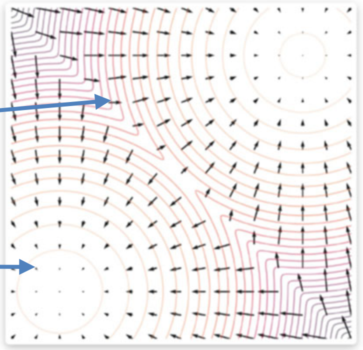
- The Score function of a data distribution $q(x), x \in R^n$ is:

$$\nabla_x \log q(x) \ \in R^n$$

  - Interpretation
    - Given a point x in data space, the score tells us which direction to move towards a region with higher likelihood
    - How to use this information for generating data from the distribution $q(.)$?
      - Sample x_0 from a prior (e.g. Gaussian) distribution $\pi(x)$ in R^n and iterate $x_{i+1} = x_i + \nabla_x \log q(x_i)$
      - Warning: indexes are in the reverse order compared to DDPM
      - This is similar to the reverse process in DDPMs

Low density region

High density region



Score function (the vector field) and density function (contours) of a mixture of two Gaussians.

Fig. Song 2022 illustrates the score function (arrows) and the density for a mixture of two gaussians

Score based models
Training

- **Score based model** (SBM) $s_\theta(.)$
  - $\nabla_x \log q(x)$ is usually intractable, one will learn a score based model, i.e. a parametric model $\mathrm{s}_\theta(\mathrm{x})$ to be implemented by a NN
    - $\mathrm{s}_\theta(\mathrm{x}) \approx \nabla_x \log q(x), s_\theta : R^n \to R^n$
    - $\mathrm{s}_\theta(\mathrm{x})$ will be learned from a sample of the target distribution $q(x)$
- **Score matching**
  - SBM can be trained by minimizing the following loss between the model $\mathrm{s}_\theta(.)$ and the data distribution $\nabla_x \log q(x)$
    - $E_{q(x)}[\|\nabla_x \log q(x) - \mathrm{s}_\theta(\mathrm{x})\|_2^2] = \int \|\nabla_x \log q(x) - \mathrm{s}_\theta(\mathrm{x})\|_2^2 q(x) dx$
- Summary
  - A distribution can be represented by its score function $\nabla_x \log q(x)$
  - The score function can be estimated by training a score based model $\mathrm{s}_\theta(\mathrm{x})$ using samples from the target distribution with score matching

Score based models
Generating samples from the target distribution

- Langevin dynamics
  - Once trained, $s_\theta(\mathrm{x})$ can be used by starting from a prior distribution $x_0 \sim \pi(x)$ (e.g. a Gaussian) and iterating a Markov chain for generating samples
    - $x_{i+1} = x_i + \epsilon s_\theta(x_i) + \sqrt{2\epsilon} z_i, i = 0, \dots, K$, with $z_i \sim \mathcal{N}(0, I)$, ϵ is a small constant
    - This is similar to the reverse process in DDPM
    - When $\epsilon \to 0$ and $K \to \infty$, x_K converges to a sample from $q(x)$ under some regularity conditions
      - □ In practice take ϵ small and K large (100 to 1000)

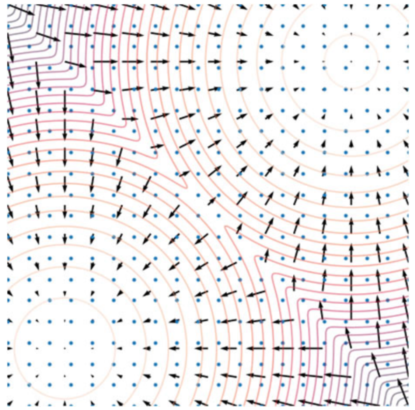


Fig. Song 2022
Langevin dynamics for sampling from a mixture of 2 gaussians, arrows indicate the score vector values, the animated Gif shows the convergence of the dynamics towards the target distribution

Score based models
Summary: training + generation

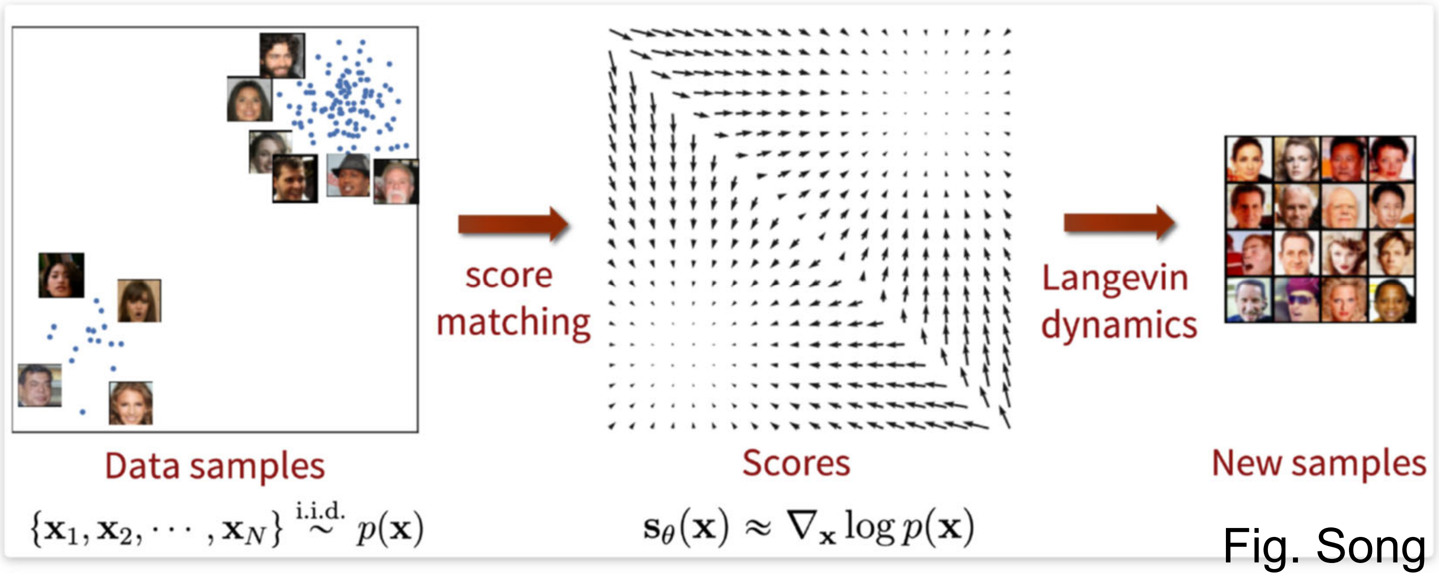

score matching
Langevin dynamics
Data samples
$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p(\mathbf{x})$
Scores
$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_\mathbf{x} \log p(\mathbf{x})$
New samples

Fig. Song 2022

Score based models
Training: Noise conditionned score network (NCSN)

- let us come back to the score matching training formulation
  - $argmin_\theta E_{q(x)}[\|\nabla_x \log q(x) - \mathrm{s}_\theta(\mathrm{x})\|_2^2]$
- This formulation leaves us with 2 problems (Song et al. 2020)
  - (1) $q(x)$ is unknown
  - (2) In low density regions, there are only a few data points available so that $s_\theta(x)$ will be inaccurate.
  - (Song et al. 2020) propose different solutions to this problem, let us describe one of them:
    - Noise conditionned score network (NCSN)

Score based models
Training: Noise conditionned score network (NCSN)

- Noise conditionned score network (NCSN)
  - Intuition
    - Instead of training on the data distribution directly, train on noisy data
    - Perturb data points with noise $\mathcal{N}(0, \sigma I)$, train score based models on the noisy points using score matching.
      - If the noise magnitude is large enough this should help populating the low density regions, i.e. helps solving pb (2) ($s_\theta(\mathrm{x})$ innaccurate in low density regions)
      - What should be the noise scale?
        - Large noise populate the space but alters the original distribution
        - Small noise does not cover low density regions

Score based models
Training: Noise conditionned score network (NCSN)

- Noise conditionned score network (NCSN)
  - This idea is then refined as follows
    - Use multiple and increasing scales of noise $\mathcal{N}(0, \sigma_i I), i = 1 \dots, T$ with $\sigma_1 < \sigma_2 < \cdots < \sigma_T$ in order to obtain T noise-perturbed distributions $q_{\sigma_i}(\tilde{x}) \triangleq \int q_{\sigma_i}(\tilde{x}|x) q(x) dx$
    - In practice this is achieved by drawing samples from $q_{\sigma_i}(\tilde{x})$ by sampling $x \sim q(x)$ and computing $\tilde{x} = x + \sigma_i z$ with $z \sim \mathcal{N}(O, I)$
    - Use a **unique (θ) score function** parameterized by $\sigma, \mathrm{s}_\theta(\mathrm{x}; \sigma)$ for all the noise scales and train it with the different noise scales using score matching so that $\mathrm{s}_\theta(\mathrm{x}; \sigma_\mathrm{i}) \approx \nabla_x \log q_{\sigma_i}(x)$
      - $\mathrm{s}_\theta(\mathrm{x}; \sigma)$ is called a **noise conditional score-based model**
    - Noise schedule: for example geometric schedule between two extreme values $\sigma_1$to σ_T

  - Note
    - This is similar to the forward process in DDPMs

Score based models
Noise conditionned score network (NCSN)
Training formulation detailed

- Noise conditionned score network (NCSN)
  - Let $\tilde{x}$ a perturbation of x generated according to the transition kernel $q_\sigma(\tilde{x}|x) = \mathcal{N}(\tilde{x}; \mathrm{x}, \sigma^2 I)$
    - i.e. $\tilde{x}$ is a noisy version of x
    - $\tilde{x}$ can be generated as $\tilde{x} = x + \sigma^2\epsilon, \epsilon \sim \mathcal{N}(0, I)$
    - Let us define $q_\sigma(\tilde{x}) \triangleq \int \; q_\sigma(\tilde{x}|x)q(x)dx$
  - The proposed loss function is
    - $\frac{1}{T}\sum_{i=1}^{T} \lambda(\sigma_i) E_{q_{\sigma_i}(x)}\left[\left\|\nabla_{\tilde{x}} \log q_{\sigma_i}(\tilde{x}) - s_\theta(\tilde{x}, \sigma_i)\right\|_2^2\right]$
      - This is a weighted sum of score matching losses, $\lambda(i) \in R, > 0$, often chosen as $\lambda(i) = \sigma_i^2$
  - This can be rewriten up to a constant as
    - $\frac{1}{T}\sum_{i=1}^{T} \lambda(\sigma_i) E_{x \sim q(x), \tilde{x} \sim q_{\sigma_i}(\tilde{x}|x)}\left[\left\|\frac{\tilde{x}-x}{\sigma_i^2} + s_\theta(\tilde{x}, \sigma_i)\right\|_2^2\right]$
      - $q_\sigma(\tilde{x}|x) = \mathcal{N}(\tilde{x}; \mathrm{x}, \sigma^2 I) \Rightarrow \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}) = -\frac{\tilde{x}-x}{\sigma^2}$
  - $\lambda(\sigma_i)$ is set for example to σ_i^2 - so that all the components inside the summation have the same order of magnitude and do not depend on σ
    - $\frac{1}{T}\sum_{i=1}^{T} E_{x \sim q(x), \tilde{x} \sim q_{\sigma_i}(\tilde{x}|x)}\left[\left\|\frac{\tilde{x}-x}{\sigma_i} + \sigma_i \, s_\theta(\tilde{x}, \sigma_i)\right\|_2^2\right]$
  - After training $\sigma_i, s_\theta(\tilde{x}, \sigma_i)$ will return an estimate of the score $\nabla_{\tilde{x}} \log q_{\sigma_i}(\tilde{x})$

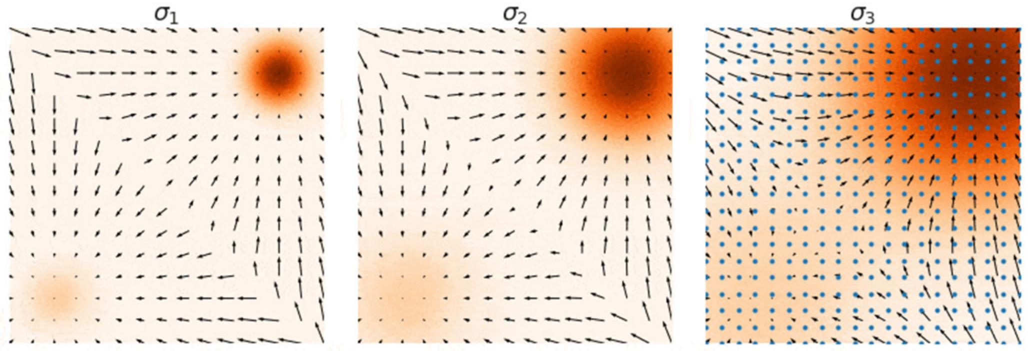
Score based models
Generation

- For the generation, it is proposed to use an annealed form of the Langevin dynamics

```
Initialize $x_0 \sim \mathcal{N}(0, I)$ (prior distribution)
For $t = T$ to $1$ (annealing iterations)
    set $\alpha_t$ the step size e.g. $\alpha_t = \epsilon \frac{\sigma_t^2}{\sigma_1^2}$ with $\epsilon$ a small positive constant
    For $i = 1$ to $N - 1$ (N steps of Langevin dynamics)
        Draw $z_i \sim \mathcal{N}(0, I)$
        $x_{i+1} = x_i + \alpha_t s_\theta(x_i, \sigma_t) + \sqrt{2\alpha_t} z_i$
    $x_0 = x_N$
Return $x_0$
```

Fig. Song – Blog 2021

Remark: at each annealing iteration, one starts from the final sample of the previous iteration



Score based models

- $s_\theta(x_i, t)$ is parametrized with U-Nets with residual connections as for DDPMs
- Equivalence with DDPM
  - The two training objectives (DDPM and SGM) are equivalent once we set
    - $\epsilon_\theta(x, t) = -\sigma_t(x, t)$

Score based models
Generation - example

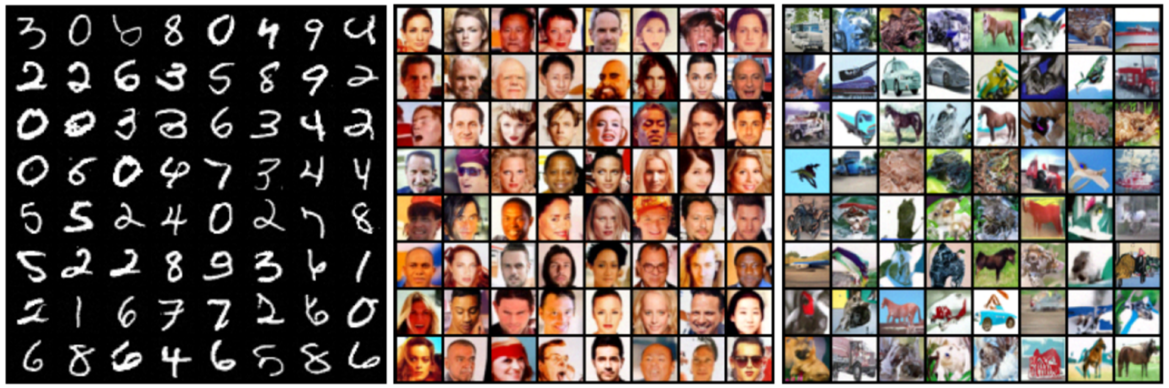

(a) MNIST (b) CelebA (c) CIFAR-10

Figure 5: Uncurated samples on MNIST, CelebA, and CIFAR-10 datasets.

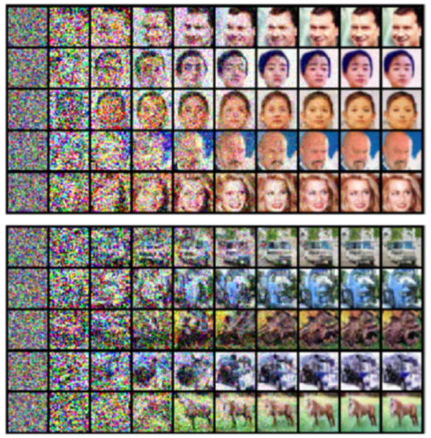


Figure 4: Intermediate samples of annealed Langevin dynamics.

Fig. Song et al 2020

Score stochastic differential equation

(Song et al. 2021)

Score stochastic differential equation

- Generalizes the discrete diffusion and score based formulations to time continuous dynamics
  - i.e. one considers the limit when the time step α_t in score based methods goes to 0
- Both DDPM and Score based approaches can be formulated as discretizations of SDE formulations

Score stochastic differential equation
Forward dynamics

- Stochastic differential equations (SDE)
  - $dx(t) = f(x,t)dt + g(t)d\omega$
    - $f(x,t)$ is a vector valued drift function, $f: R^n \rightarrow R^n$
    - $g(t)$ is a scalar valued diffusion function, $g: R \rightarrow R$
      - □ g is considered scalar and independent of x for s
      but could be a vector valued fonction and dependent
    - ω is a Wiener process (Brownian motion), $d\omega \sim \mathcal{N}($
    - Under some conditions, the SDE has a unique soluti
- Time discretization
  - $x_{t+\Delta t} = x_t + f(x_t,t)\Delta t + g(x_t,t)\Delta\omega$, with $\Delta\omega \sim \mathcal{N}(0,\Delta t)$
- Note
  - Langevin dynamics $x_{t+1} = x_t + \alpha_t s_\theta(x_t,t) + \sqrt{2\alpha_t} z_t$ appears as a special case of the discrete equation with:
    - $\Delta t = 1, f(x_t,t) = \alpha_t s_\theta(x_t,t), g(x_t,t) = \sqrt{2\alpha_t}, \Delta\omega = z_t$
  - As for the discrete case, the forward diffusion process does not depend on the data

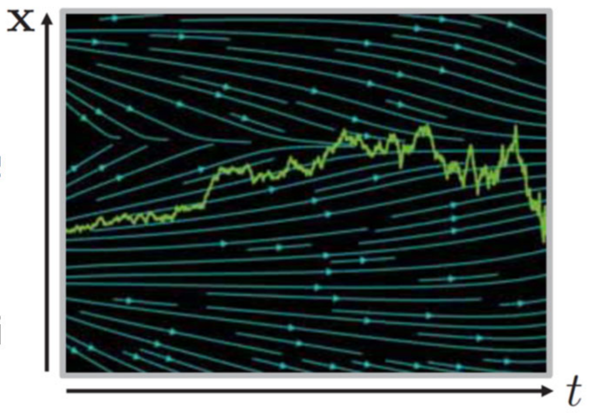


Fig. Kreis et al. 2022
Sample from a SDE trajectory

Score stochastic differential equation
Forward dynamics

- Diffusion processes can be modeled as solutions of SDEs
  - The solution of a SDE is a continous collection of random variables $\{x(t)\}_{t \in [0,T]}$
  - These variables trace stochastic trajectories when t grows from 0 to T
- Let us denote $q_t(x)$ the probability density of $x(t)$, and $q(x(t)|x(s))$ the transition kernel from $x(s)$ to $x(t)$ with $s < t$
- The objective is to construct a forward diffusion process $\{x(t)\}_{t \in [0,T]}$, indexed by the continuous variable t so that $x(0) \sim p_0$, the data distribution and $x(T) \sim q_T$ is a tractable distribution that can be easily sampled, i.e. a prior π, e.g. a gaussian with fixed mean and variance

Score stochastic differential equation
Forward process

- Illustration: stochastic trajectories for the forward diffusion process

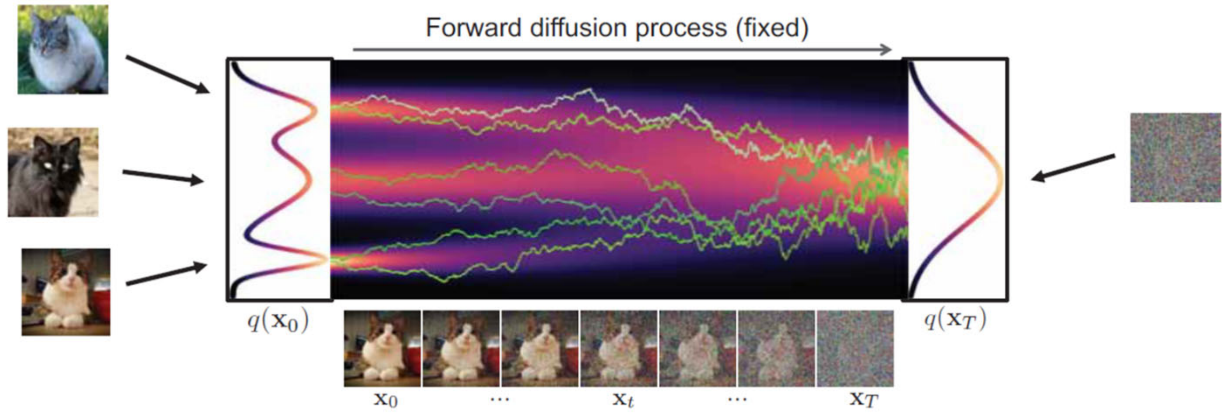


Fig. Kreis et al. 2022
Samples: SDE trajectories from different initial points

Score stochastic differential equation
Forward process

- Illustration: stochastic trajectories for the forward diffusion process

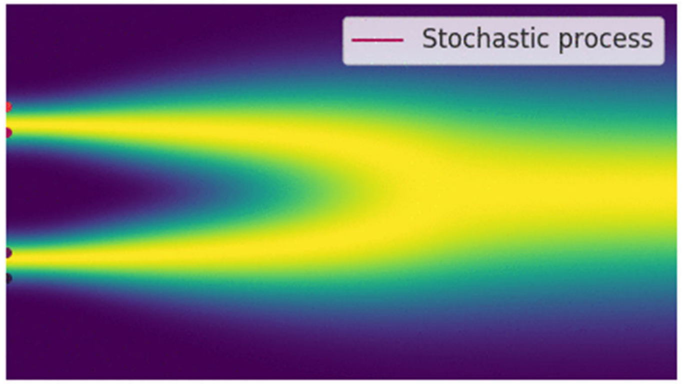


Fig. Song 2021 - https://yang-song.net/blog/2021/score/

Score stochastic differential equation

- DDPMs and SGMs are both special cases of the SDE discretization
- \>>>>>>>>>>>>>>>>>to be completed <<<<<<<<<<<<<<<<<<<<<

Score stochastic differential equation
Reversing the SDE

- For samples generation, one needs to reverse the SDE
  - Any diffusion process modeled as a SDE can be reversed by solving the reverse SDE backward, i.e. from $t = T$ to $t = 0$
    - i.e. one starts at $x(T) \sim q_T$ and reversing the process we obtain samples $x(0) \sim q_0$
  - The reverse SDE writes as
    - $dx = \big(f(x,t) - g(t)^2 \nabla_x log q_t(x)\big)dt + g(t)dw$, with dt an infinitesimal negative time step
      - □ $q_t(x)$ is the distribution of x at ti
      - □ Once $\nabla_x log q_t(x)$ is known for a it by sampling from $q_T(x)$ to gen

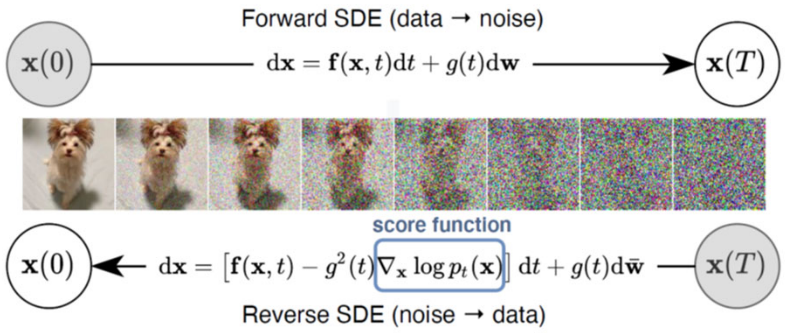
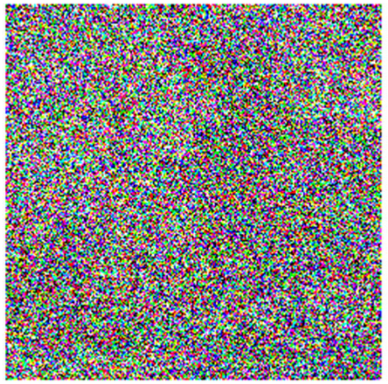

Forward SDE (data → noise)
$\mathbf{x}(0)$
$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w}$
$\mathbf{x}(T)$
score function
$\mathbf{x}(0)$
$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\boxed{\nabla_\mathbf{x} \log p_t(\mathbf{x})}]dt + g(t)d\bar{\mathbf{w}}$
$\mathbf{x}(T)$
Reverse SDE (noise → data)

Fig. Song et al. 2021

Score stochastic differential equation
Reversing the SDE

- Reverse process illustration
  - One starts from noisy samples to generate target data samples



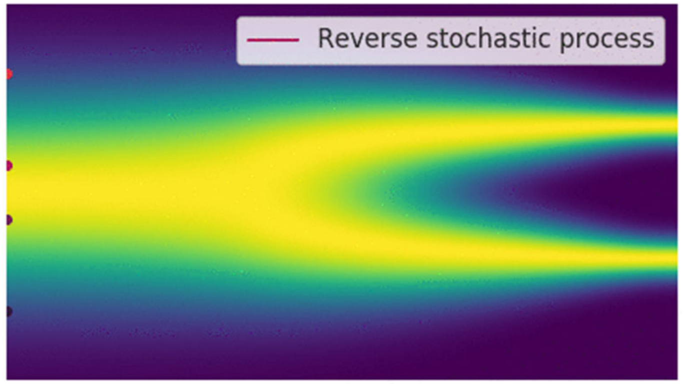


Fig. Song 2021 - https://yang-song.net/blog/2021/score/

Score stochastic differential equation

- With this formulation, we are then left with two problems
  - The training problem: how to estimate $\nabla_x log q_t(x)$ the score function of $q_t(x)$?
  - How to solve the reverse SDE?

Score stochastic differential equation
The training problem

- Solving the reverse SDE requires to know the terminal distribution $p_T(x)$ and the score function $\nabla_x log q_t(x)$
  - For the former one uses a prior distribution $\pi(x)$, typically a gaussian
  - For the latter, one trains a time-dependent score-based model $s_\theta(x,t)$ such that $s_\theta(x,t) \approx \nabla_x log q_t(x)$
    - Note: this is analogous to the discrete case $s_\theta(x,i) \approx \nabla_x log q_{\sigma_i}(x)$
- The training objective is a continuous extension of the one used with SGMs:
  - $E_{t\sim\mathcal{U}(0,T)}E_{q_t(x)}\left[\lambda(t)\|\nabla_x log q_t(x) - s_\theta(x,t)\|_2^2\right]$
    - $\mathcal{U}(0,T)$ is a uniform distribution over $[0,T]$ and $\lambda: R \rightarrow R$ is a positive weighting function
      - As for the discrete case, $\lambda(t)$ will be set so as to balance the magintude of the different score matching losses across time
- Generation
  - Once trained, one can simulate from $dx = (f(x,t) - g(t)^2 s_\theta(x,t))dt + g(t)dw$
- Practical training
  - Use a score matching method e.g. denoising score matching

Score stochastic differential equation
The training problem

- Denoising score matching
  - As in the discrete case, diffuse individual data points using diffusion kernels $q_t(x(t)|x(0))$
    - $Min_\theta E_{t\sim\mathcal{U}(0,T)} E_{x(0)\sim q_0(x)} E_{x(t)\sim q(x(t)|x(0)))} \left[\lambda(t) \left\| \nabla_{x_t} log q_t(x(t)|x(0)) - s_\theta(x(t),t) \right\|_2^2 \right]$
    - diffusion kernels $q(x(t)|x(0))$ are chosen Gaussian for linear SDEs (this means f is affine):

$$q(x(t)|x(0)) = \mathcal{N}\left(x(t); \gamma_t x(0), \sigma_t^2 I\right)$$

- Objective: as in the discrete case, the loss function can be derived as
  - $Min_\theta E_{t\sim\mathcal{U}(0,T)} E_{x\sim q(x)} E_{\epsilon\sim\mathcal{N}(O,I)} \left[\frac{\lambda(t)}{\sigma_t^2} \|\epsilon - \epsilon_\theta(x_t,t)\|_2^2 \right]$
- Practice
  - Different loss weightings are proposed, e.g. $\lambda(t) = \sigma_t^2$ for the simplest case
  - $s_\theta(x(t),t)$ or $\epsilon_\theta(x_t,t)$ implemented with U-Nets
  - For the time integration, one could use Fourier features on t or replace t by σ_t

Score stochastic differential equation
Solving the SDE

- Once $s_\theta(x,t)$ is learned, it can be plugged in the reverse SDE
  - $dx = (f(x,t) - g(t)^2 s_\theta(x,t))dt + g(t)dw$
  - Starting with $x(T) \sim \pi$, one can solve this reverse SDE to obtain a sample $x(0)$ from the target distribution $q(x)$ – or at least a sample from the approximate distribution $q_\theta(x) \approx q(x)$
- How to solve the reverse SDE
  - Learning free methods
    - SDE solvers – a variety of SDE solvers is available from the numerical analysis literature
      - □ Discretize the SDE in time and use a SDE solver
    - ODE solvers – this is detailed in the next slides – Faster that SDE solvers
  - Learning methods
    - Take benefit from the special for of the SDE in order to optimize the reverse solver

Score stochastic differential equation
ODE solvers

- (Song et al 2021) show that it is possible to associate an ODE to any SDE without changing the marginal distribution $\{q_t(x)\}_{t\in[0,T]}$. i.e. both the ODE and the SDE share the same set of marginal distributions $\{q_t(x)\}_{t\in[0,T]}$
  - The ODE associated to the reverse SDE is:
  - $\frac{dx}{dt} = f(x,t) - \frac{1}{2}g^2(t)\nabla_{\mathrm{x}}\log \mathrm{q_t(x)}$
  - This is called the probability flow ODE associated to the SDE
- It is then possible to sample from the same distribution as the reverse SDE by solving the ODE using classical ODE solvers (e.g. Runge Kutta)
- Note
  - When $\nabla_{\mathrm{x}}\log \mathrm{q_t(x)}$ is replaced by $s_\theta(x,t)$ the ODE becomes a special case of Neural ODE (see later in the course) – more precisely it is a continuous normalizing flow

Score stochastic differential equation
ODE solvers

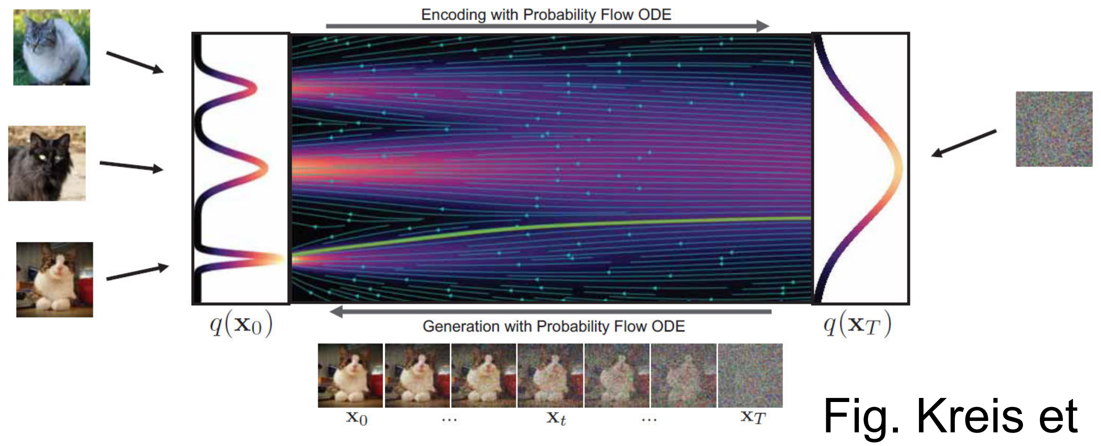
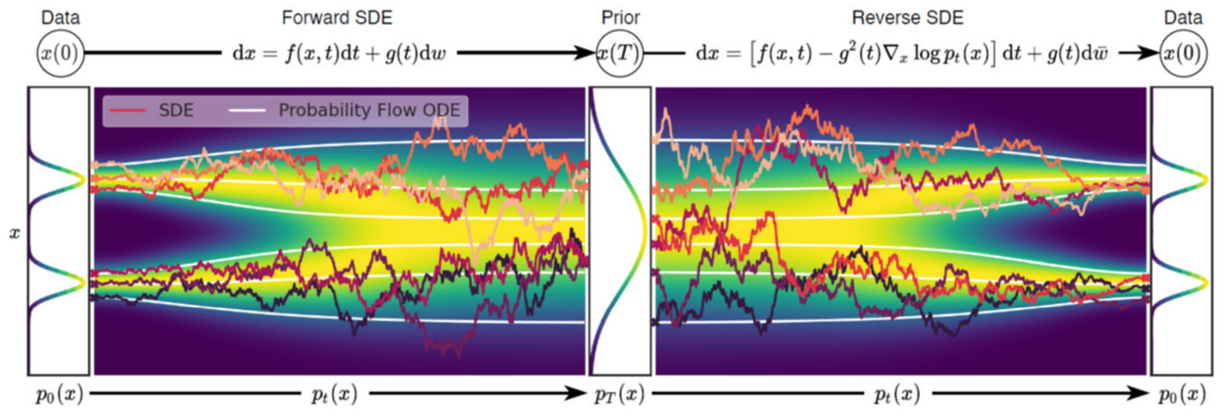


Fig. Kreis et al. 2021

- Current practice
  - Solve the forward process using the sde formulation (easy, no training)
  - Solve the reverse process using the ODE formulation
  - Note: the ODE could be used for the forward and reverse diffusion since (simply change the integration direction i.e. consider $t > 0$ for one direction and and $t < 0$ for the other direction), however the forward process is simpler with the fixed SDE formulation.

Score stochastic differential equation
ODE / SDE solvers



Data
Forward SDE
Prior
Reverse SDE
Data
$x(0)$
$\mathrm{d}x = f(x,t)\mathrm{d}t + g(t)\mathrm{d}w$
$x(T)$
$\mathrm{d}x = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]\,\mathrm{d}t + g(t)\mathrm{d}\bar{w}$
$x(0)$
SDE
Probability Flow ODE
x
$p_0(x)$
$p_t(x)$
$p_T(x)$
$p_t(x)$
$p_0(x)$

Figure 2: **Overview of score-based generative modeling through SDEs.** We can map data to a noise distribution (the prior) with an SDE (Section 3.1), and reverse this SDE for generative modeling (Section 3.2). We can also reverse the associated probability flow ODE (Section 4.3), which yields a deterministic process that samples from the same distribution as the SDE. Both the reverse-time SDE and probability flow ODE can be obtained by estimating the score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ (Section 3.3).

Fig. Song et al. 2021

- ODE trajectories are smoother that SDE trajectories, however they allow to sample the same marginals $\{p_t(x)\}_{t\in[0,T]}$

Score based models
Conditional setting

- Several applications imply conditional generation
  - Text to image: DALL-E, IMAGEN
  - Class conditional generation
  - Super resolution, colorization, panorama etc (Saharia et al. 2020)

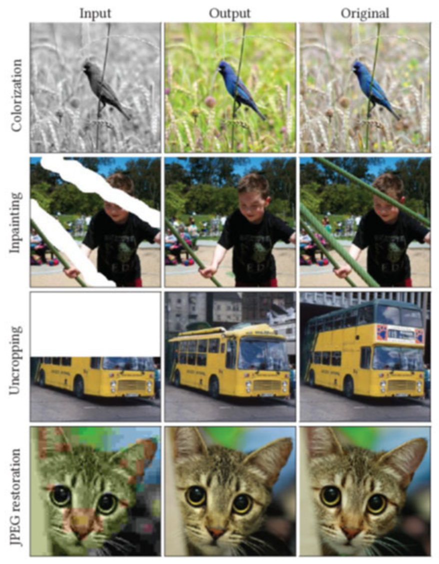


Figure 1: Image-to-image diffusion models are able to generate high-fidelity output across tasks without task-specific customization or auxiliary loss.

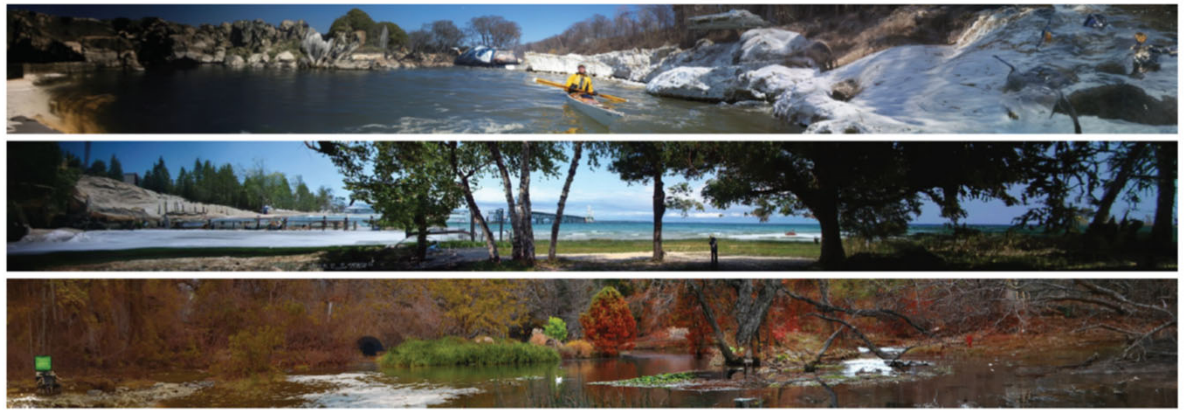


Figure 2: Given the central 256×256 pixels, we extrapolate to the left and right in steps of 128 pixels (2×8 applications of 50% Palette uncropping), to generate the final 256×2304 panorama. Figure D.3 in the Appendix shows more samples.

Fig from Saharia et al. 2020

Score based models
Conditional setting

- Conditional setting
  - Include the condition as input to the reverse process
  - The condition is input to the U-Net or whateverNet used for denoising
  - Class conditioning
    - Encode a scalar or class indicator as a vector embedding
  - Text conditioning
    - Vector embedding or sequence of vector embeddings, cross attetion, …
  - Image conditioning
    - Channel wise concatenation of the conditional image
- How to perform class conditioning
  - Several possibilities have been proposed
    - We detail here classifier guidance and classifier free guidance

Score based models
Conditional setting

- Classifier guidance
  - Instead of $q_t(x)$, one will attempt to compute $q_t(x|y)$ with y a conditioning variable
    - For simplification let us consider that y is a class indicator
    - $\nabla \log q_t(x|y) = \nabla \log \frac{q_t(x) q(y|x_t)}{q(y)}$
    - $\nabla \log q_t(x|y) = \nabla \log q_t(x) + \nabla \log q(y|x_t) - \log q(y)$
    - $\nabla \log q_t(x|y) = \nabla \log q_t(x) + \nabla \log q(y|x_t)$

    - To be completed

Diffusion models
Conclusion

- Pro
  - performance competitive with the best generative models
- Cons
  - extremely slow – due to the large number of sampling steps
- Several improvements – more to come
  - Sampling process
  - Training dynamics
  - Noise level parametrization

Diffusion models

- References
  - Blogs
    - Ayan Das 2021, https://ayandas.me/blog-tut/2021/12/04/diffusion-prob-models.html, https://ayandas.me/blog-tut/2021/07/14/generative-model-score-function.html
    - Lilian Weng, What are diffusion models: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
    - Song, Y. (2021). Generative modeling by estimating gradients of the data distribution. https://yang-song.net/blog/2021/score/
  - Slides and video
    - K. Kreis, R. Gao, A. Vahdat, CVPR tutorial, https://cvpr2022-tutorial-diffusion-models.github.io/
  - Papers
    - Karras, T., Aittala, M., Aila, T., & Laine, S. (2022). *Elucidating the Design Space of Diffusion-Based Generative Models. NeurIPS*. http://arxiv.org/abs/2206.00364
    - Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. *Neurips*, *2020-Decem*(NeurIPS 2020), 1–25.
    - Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-Resolution Image Synthesis with Latent Diffusion Models. *CVPR*, 10674–10685. https://doi.org/10.1109/cvpr52688.2022.01042
    - Saharia, C., Chan, W., Chang, H., Lee, C., Ho, J., Salimans, T., Fleet, D., & Norouzi, M. (2022). Palette: Image-to-Image Diffusion Models. In *Proceedings of ACM SIGGRAPH* (Vol. 1, Issue 1). Association for Computing Machinery. https://doi.org/10.1145/3528233.3530757
    - Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., & Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *ICML*, *3*, 2246–2255.
    - Song, Y., & Ermon, S. (2020). Generative modeling by estimating gradients of the data distribution. *Neurips*.
    - Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2021). Score-Based Generative Modeling through Stochastic Differential Equations. *ICLR*, 1–36. http://arxiv.org/abs/2011.13456
  - Tutorial / survey papers
    - Luo, C. (2022). Understanding Diffusion Models: A Unified Perspective. *Http://Arxiv.Org/Abs/2208.11970*, 1–23
    - Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., Shao, Y., Zhang, W., Cui, B., & Yang, M.-H. (2022). *Diffusion Models: A Comprehensive Survey of Methods and Applications*. *1*(1). http://arxiv.org/abs/2209.00796