Advanced Deep Learning:
- Generative Models
- AI4Science

Sorbonne Université – Masters DAC et MS2A. PatrickGallinari,
patrick.gallinari@sorbonne-universite.fr, https://pages.isir.upmc.fr/gallinari/

Year 2025-2026

Advanced Deep learning

- Generative models
  - Variational Auto-Encoders
  - Generative Adversarial Networks
  - Flow Matching and Diffusion models

- AI4Science - Physics Based Deep Learning
  - General introduction to AI4Science
  - Neural Nets and Ordinary Differential Equation
  - Neural Networks for modeling spatio-temporal dynamics
  - Applications: weather prediction

Generative models

Variational Auto-Encoders

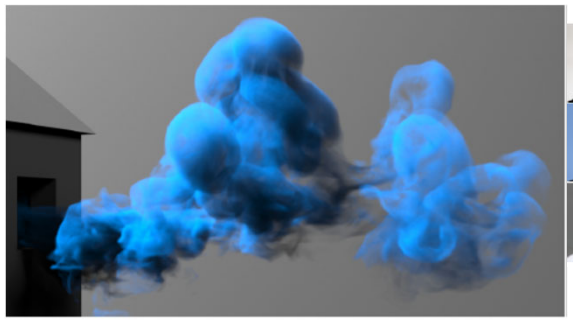
Generative Adversarial Networks

Diffusion models

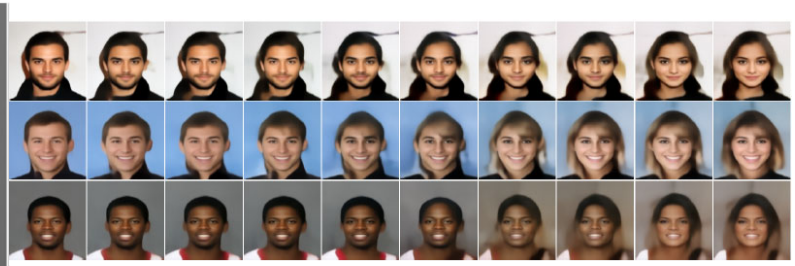
Generative models

- Objective
  - Learn a probability distribution model from data samples
    - Given $x^1, \ldots, x^N \in R^n$ 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



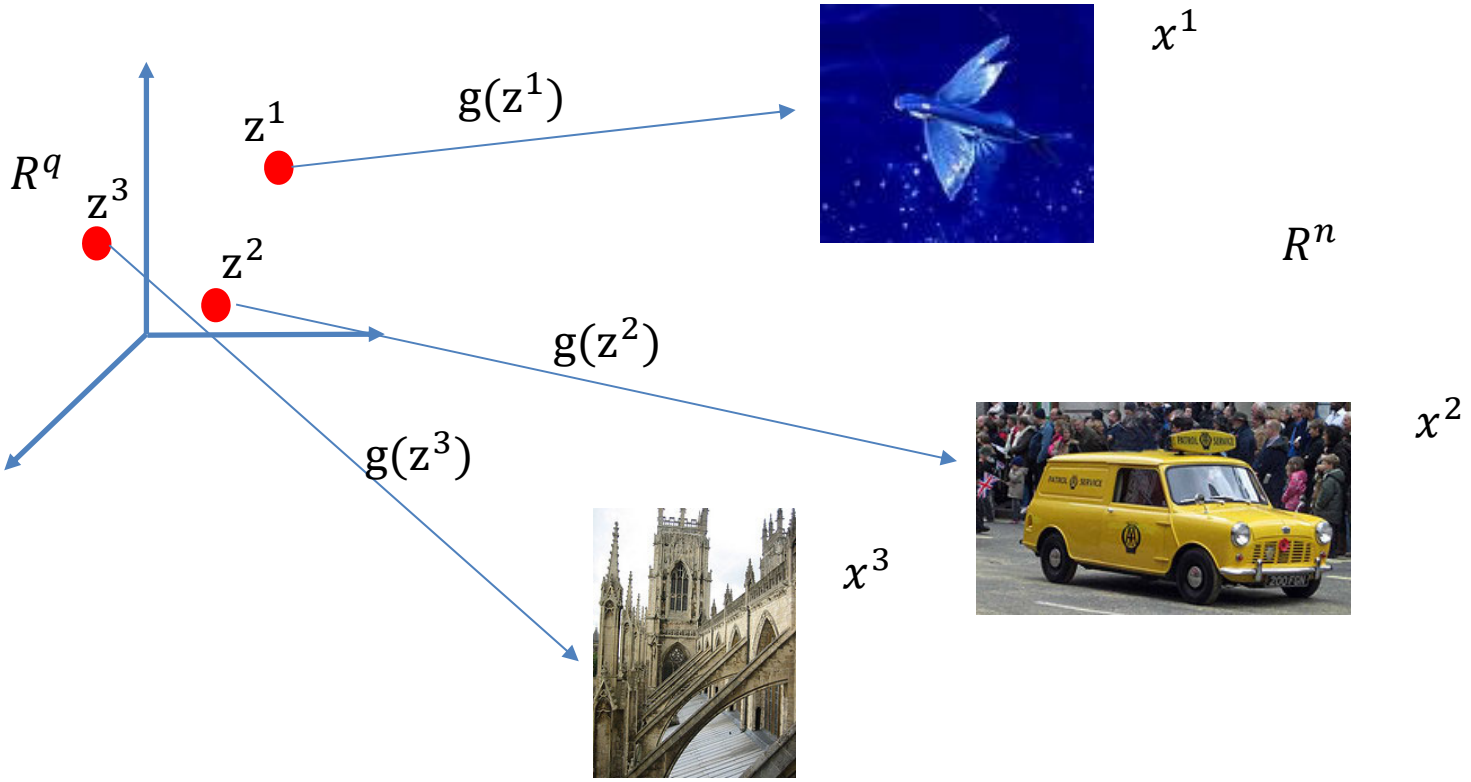
De Bezenac et al. 2021
Generating female images from male ones

Generative models

- Objective
  - General setup of deep generative models
    - Learn a generator network $g_\theta: R^q \rightarrow R^n$ 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

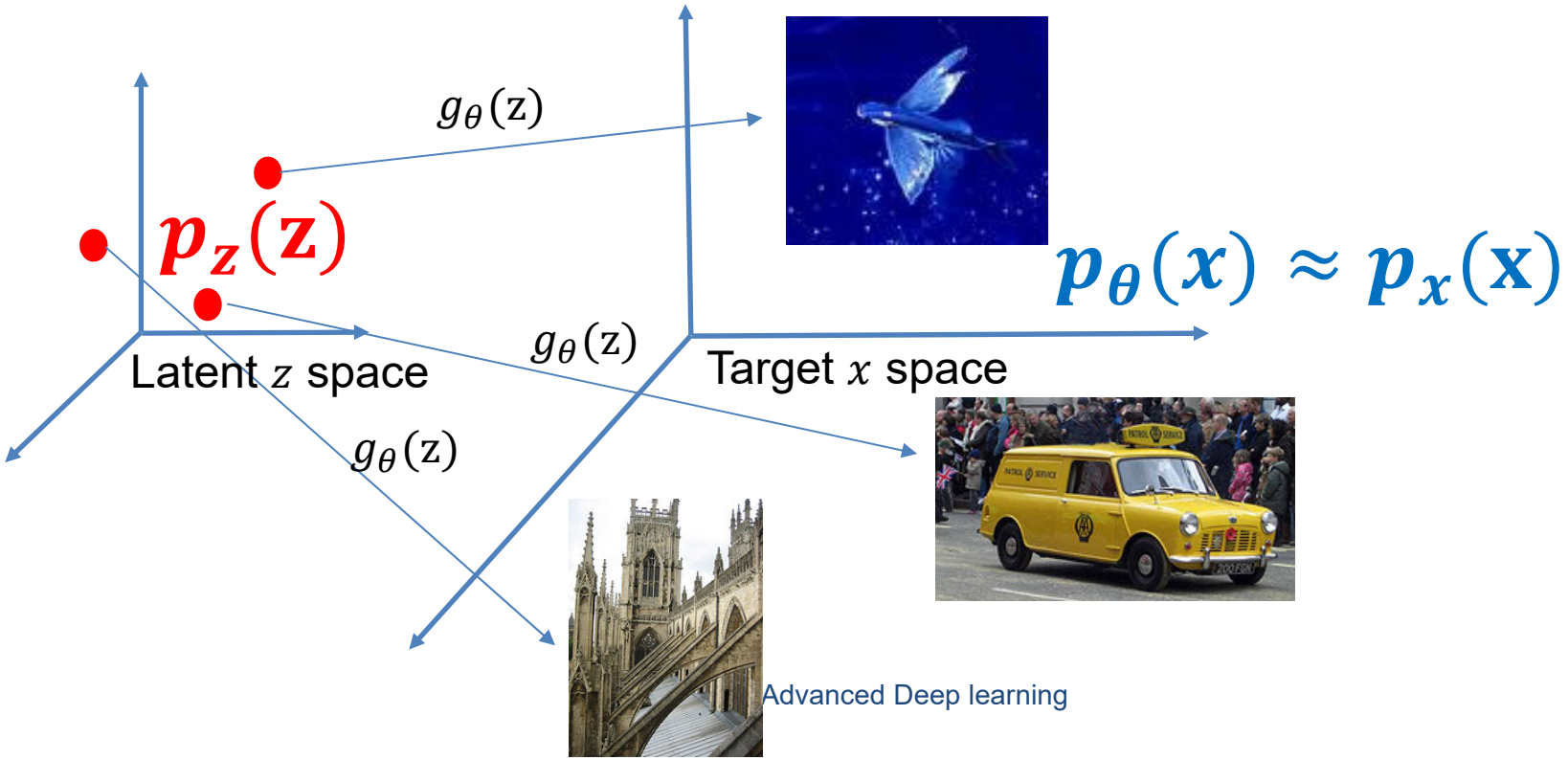
Generative models intuition

- 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:



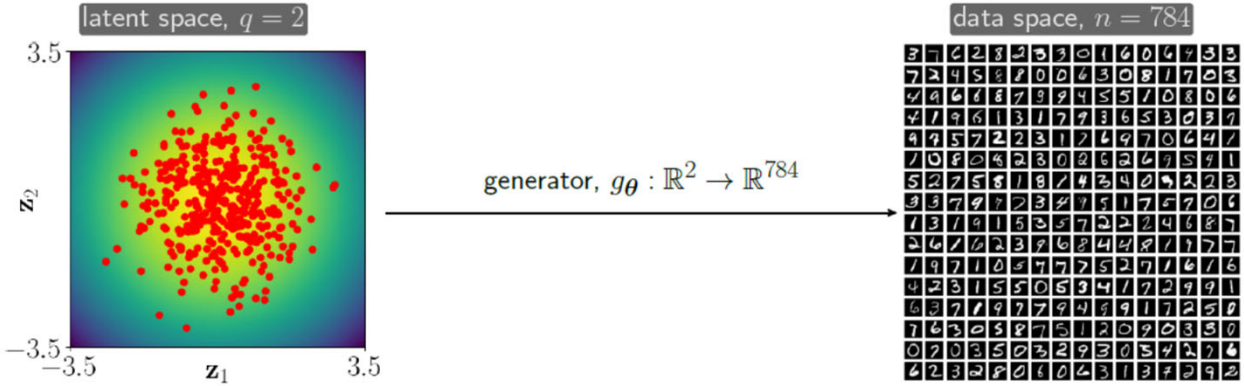
Generative models intuition

- 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, $g_\theta(z)$

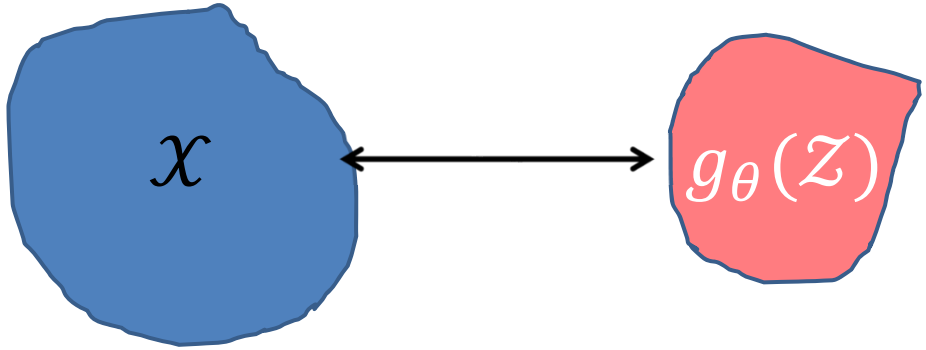


Generative models intuition

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



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



Generative models key ideas

- Objectives
  - Generative modeling aims at learning how to sample from a target distribution $p_X(x)$
  - Learn to sample using a training dataset, drawn from $p_X(x)$: $x^1, \ldots, \mathrm{x}^{\mathrm{N}}$
- Challenges
  - Enable fast sampling
  - Generate high quality samples
  - Cover $p_X(x)$
- How
  - Choose from a simple distribution, easy to sample from
  - Learn to push this distribution to the target distribution
  - The push operation is performed by a learned model (here a neural network)
- Note
  - Generation can be performed conditionally on a context
    - e.g. generating image/ video from a text promt

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
    - $g_\theta: R^q \rightarrow R^n, q \ll n$
    - Trained to maximize a lower bound of the samples' likelihood
    - Assumption: a density function explains the data
  - Generative Adversarial Networks
    - $g_\theta: R^q \rightarrow R^n, q \ll n$
    - 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
  - Flow matching and Diffusion models
    - $g_\theta: R^q \rightarrow R^n, q \ll n$ is an iterative process
    - Assumption: a density function explains the data

Variational Auto-Encoders

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

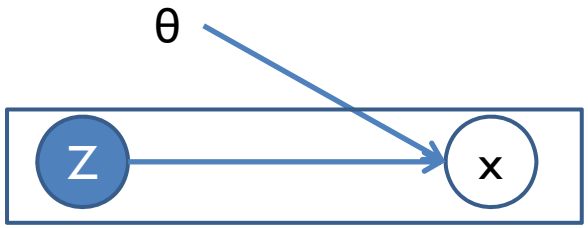
Plus some blogs – see the references

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 $p_\theta(x, z)$



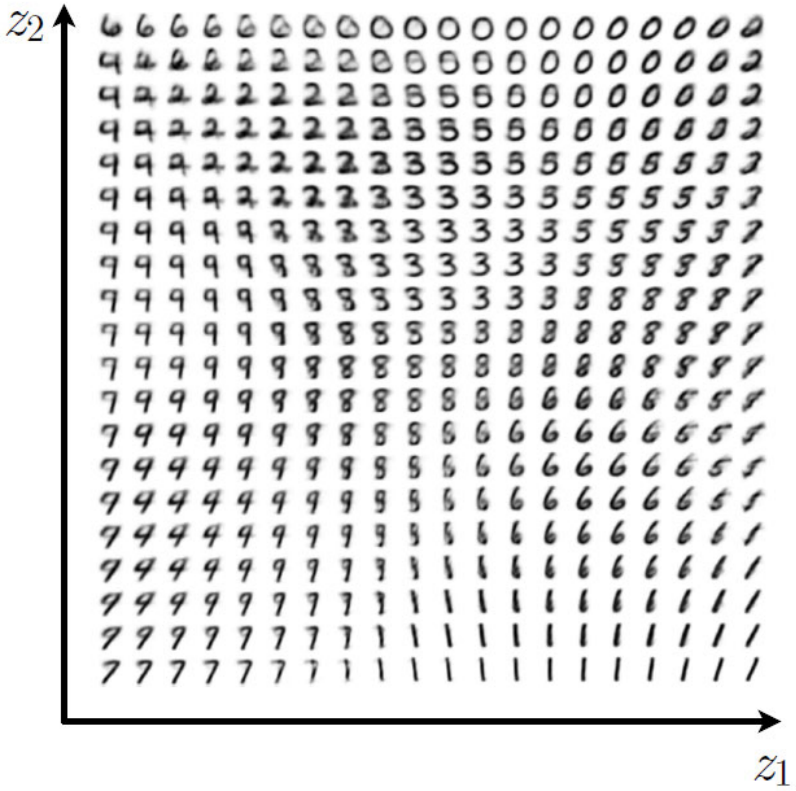
- The observations x are generated by the following process
  - Sample from $z \sim p_\theta(z)$ - $p_\theta(z)$ is the prior
  - generate $p_\theta(x|z)$ - $p_\theta(x|z)$ is the likelihood

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

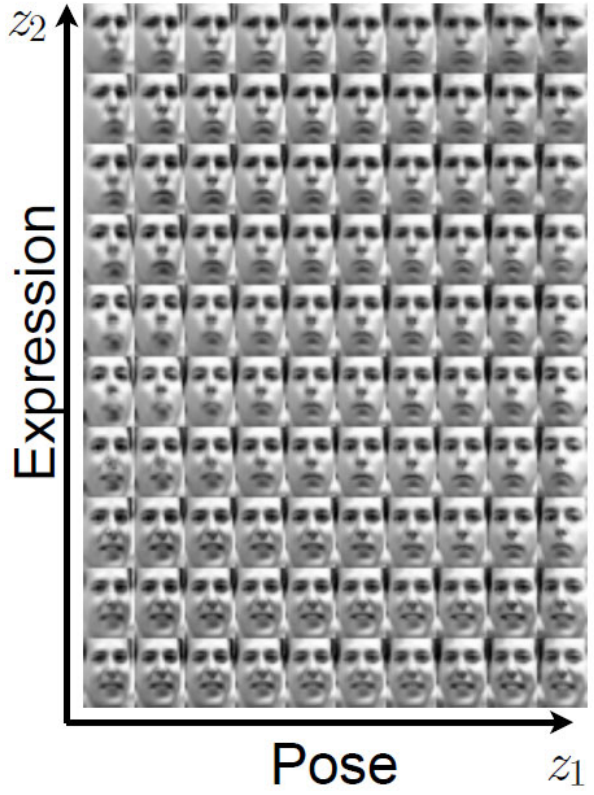


Fig. (Kingma 2015)

Generative models intuition

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

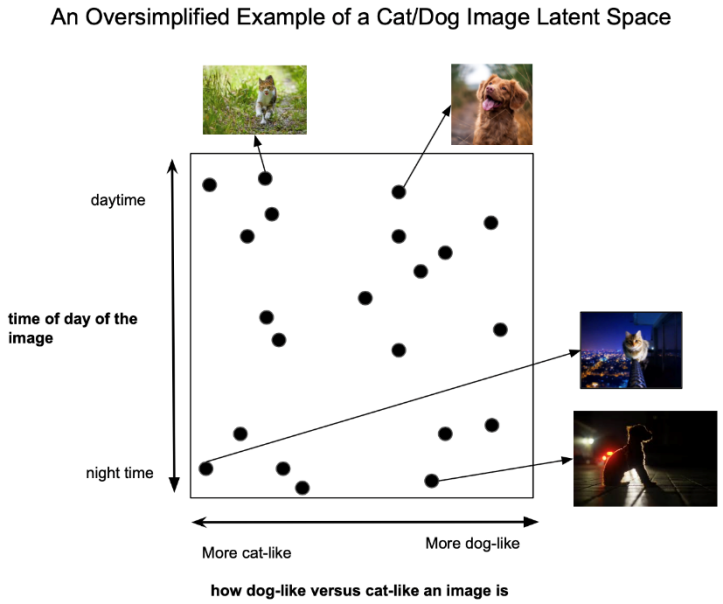


Fig.: https://ml.berkeley.edu/blog/posts/vq-vae/

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

Importance sampling and the variational approximation

- $p_\theta(x) = \int p_\theta(x \mid z)p(z)dz$
- The integral is intractable because
  - $p_\theta(x \mid z)$ is implemented by a NN, not analytically tractable
  - The latent space is high dimensional
  - Direct Monte Carlo sampling from $p(z)$ has high variance
    - most samples from $p(z)$ do not lead to samples from $p_\theta(x)$
    - almost all likelihood terms $p_\theta(x \mid z)$ are nearly zero
    - the estimator collapses, the log is dominated by $-\infty$ contributions
    - gradients are giant or undefined, training fails catastrophically.

Importance sampling and the variational approximation

- We introduce another distribution $q(z)$ with the same support as $p(z)$
  - $p_\theta(x) = \int p_\theta(x \mid z) \frac{p(z)}{q(z)} q(z) dz$
  - The variance of $p_\theta(x)$ is minimized for $q^*(z) = \frac{p_\theta(x|z)p(z)}{p_\theta(x)}$
  - This is exactly $p_\theta(z|x)$
    - However it is itself intractable ($p_\theta(x)$ in the denominator)
- VAE introduces a distribution $q_\Phi(z|x)$ that approximates $p_\theta(z|x)$
- And optimizes an evidence lower bound of the likelihood $p_\theta(x)$ that allows us to get rid from $p_\theta(z|x)$

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 $V_L(\theta, \phi; x)$
  - Thus we get rid of $p_\theta(z|x)$
- $V_L(\theta, \phi; x)$ 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

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

Proof

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

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

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:

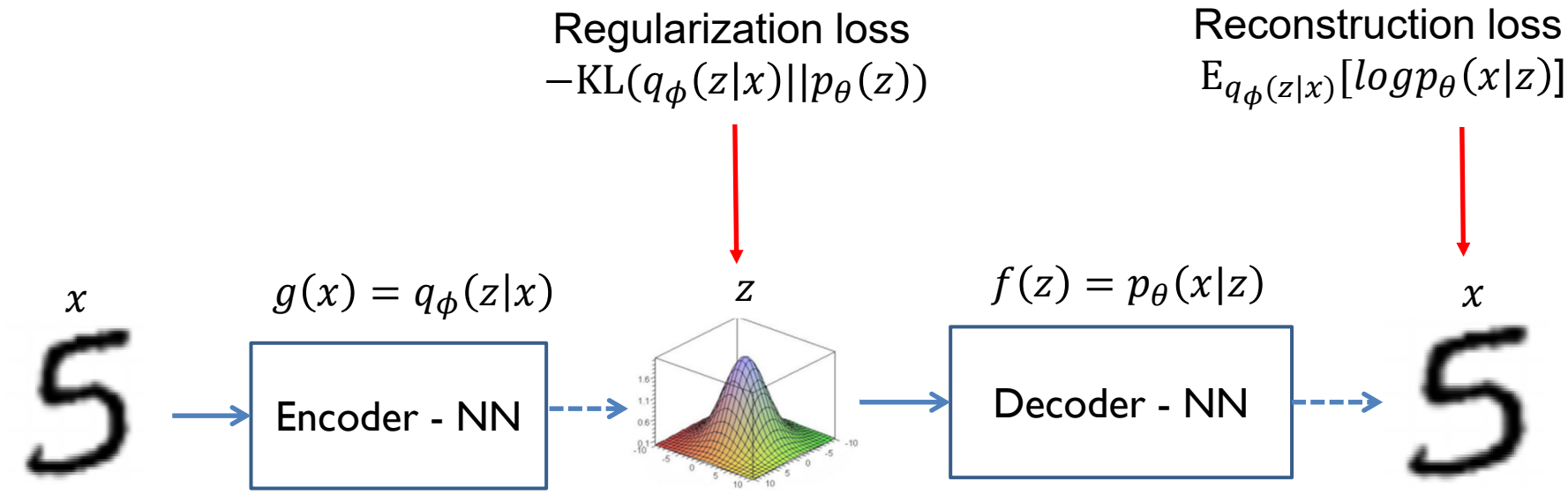
- $V_L(\theta, \phi; x) = \mathrm{E}_{q_\phi(z|x)}[log p_\theta(x, z) - \log q_\phi(z|x)]$
- $V_L(\theta, \phi; x) = \mathrm{E}_{q_\phi(z|x)}[log p_\theta(x|z) + \log p_\theta(z) - \log q_\phi(z|x)]$
- $V_L(\theta, \phi; x) = -\mathrm{D}_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z)) + \mathrm{E}_{q_\phi(z|x)}[log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

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 $p(z)$: $-\mathrm{D}_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z)) \approx 0$

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
    - ⇢ Sampling
    - → 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)

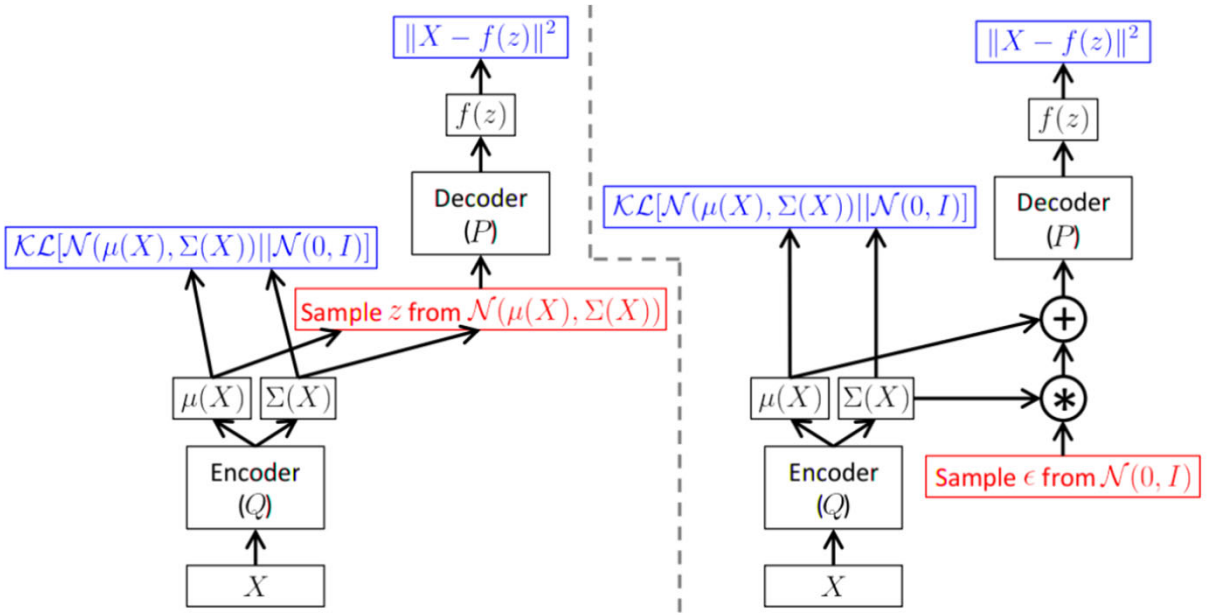


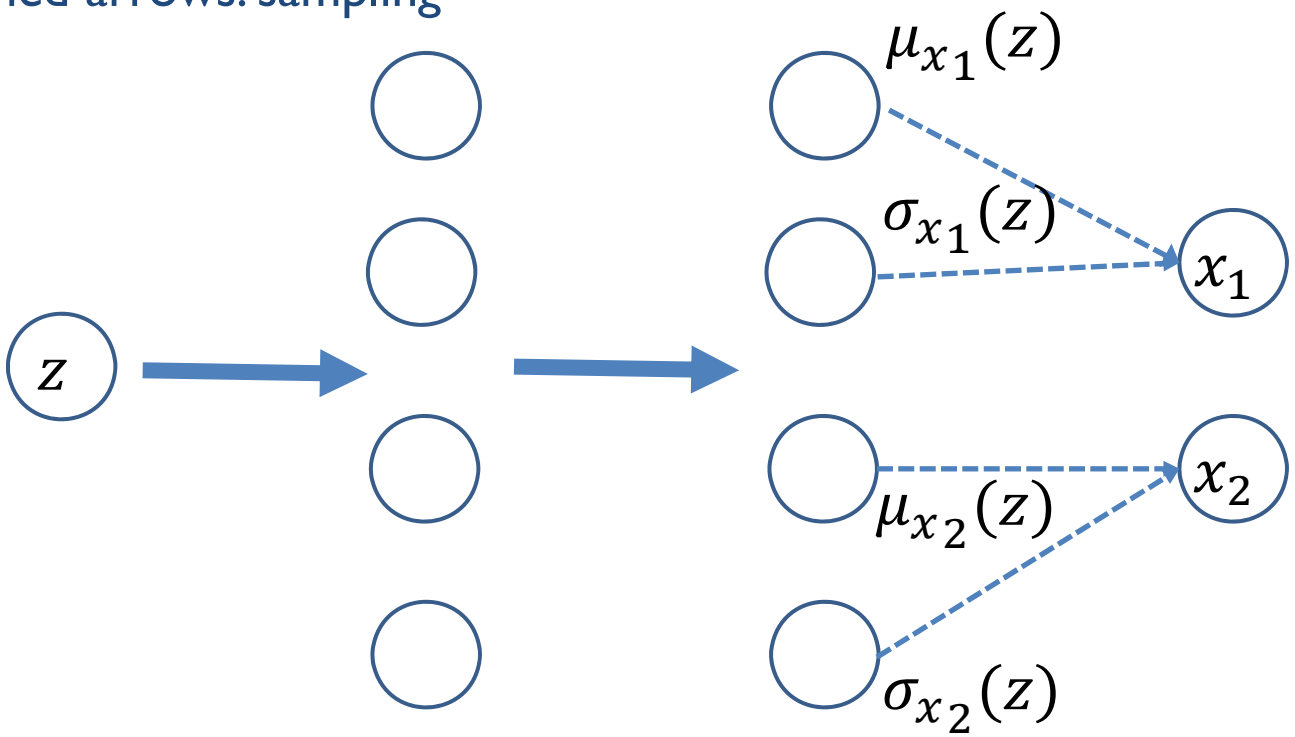
Figure 4: A training-time variational autoencoder implemented as a feed-forward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior 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, $z \in R^J$

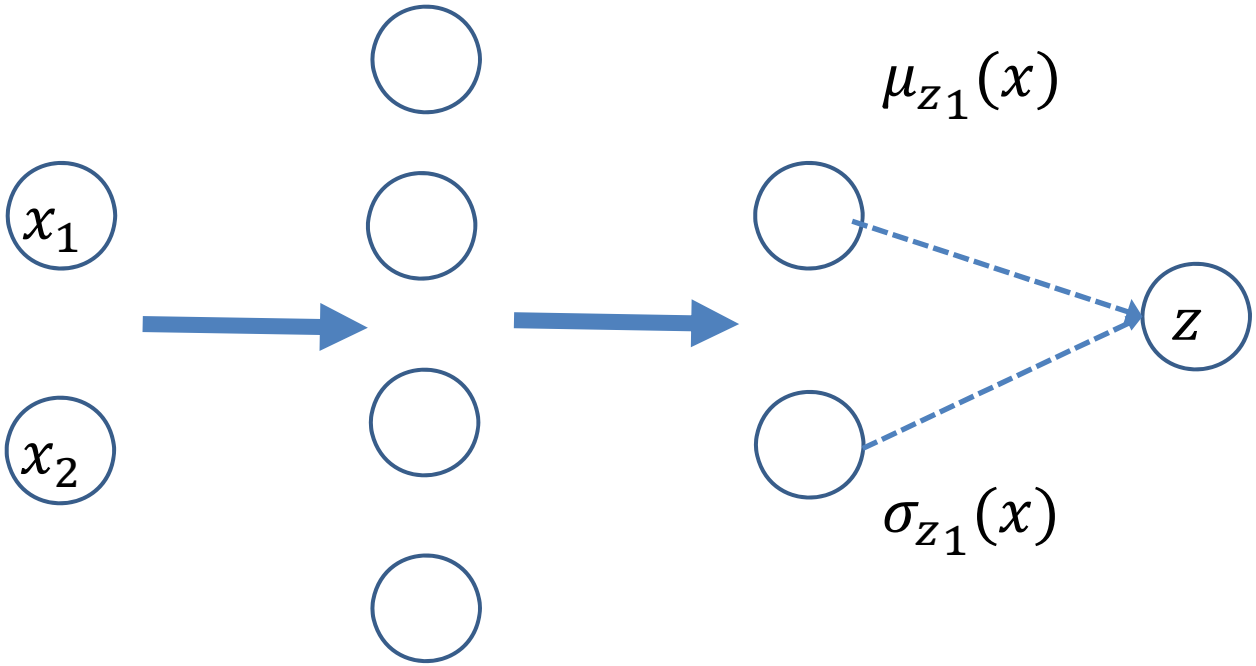
VAE
Exemple: Gaussian priors and posteriors - illustration

- 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



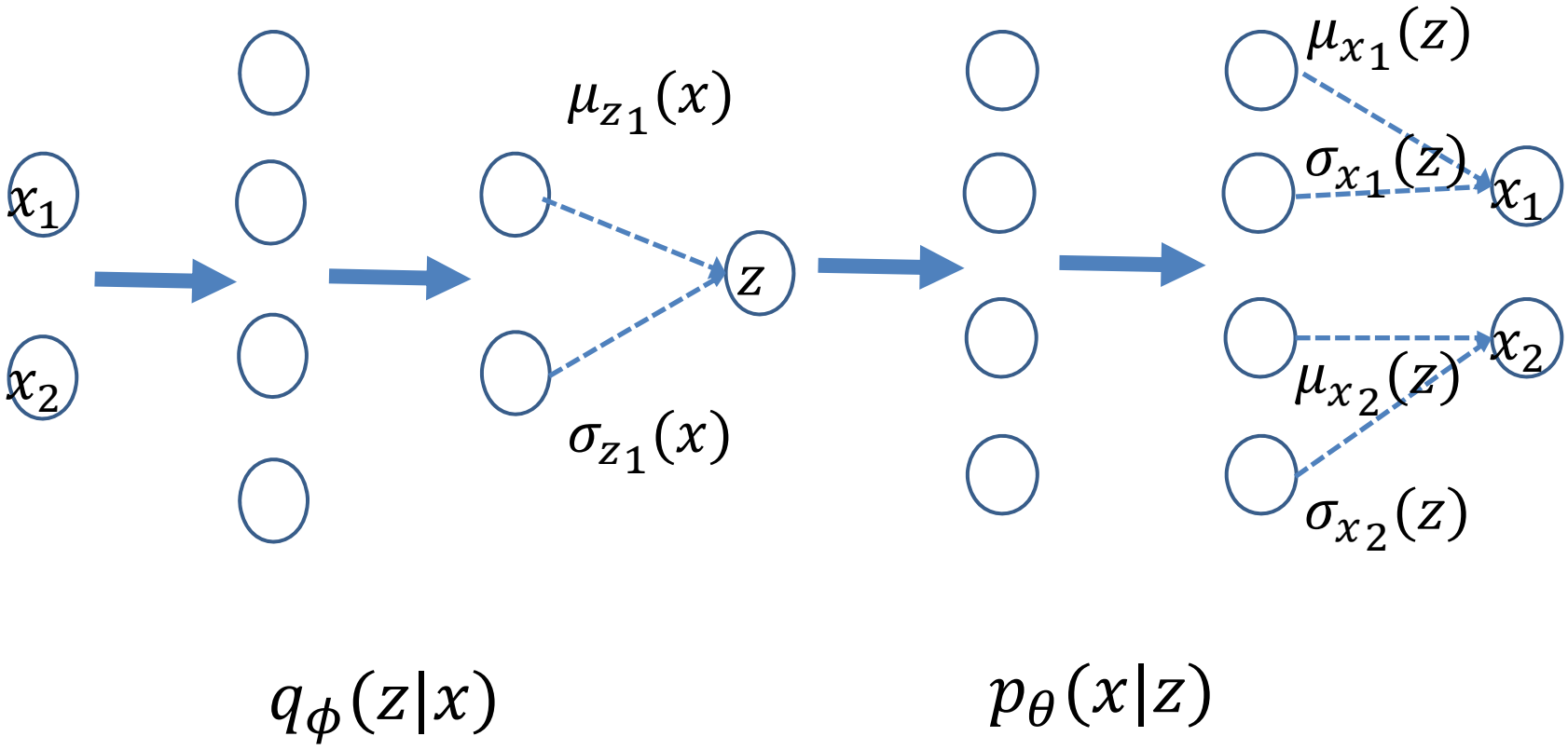
VAE
Gaussian priors and posteriors - illustration

- 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



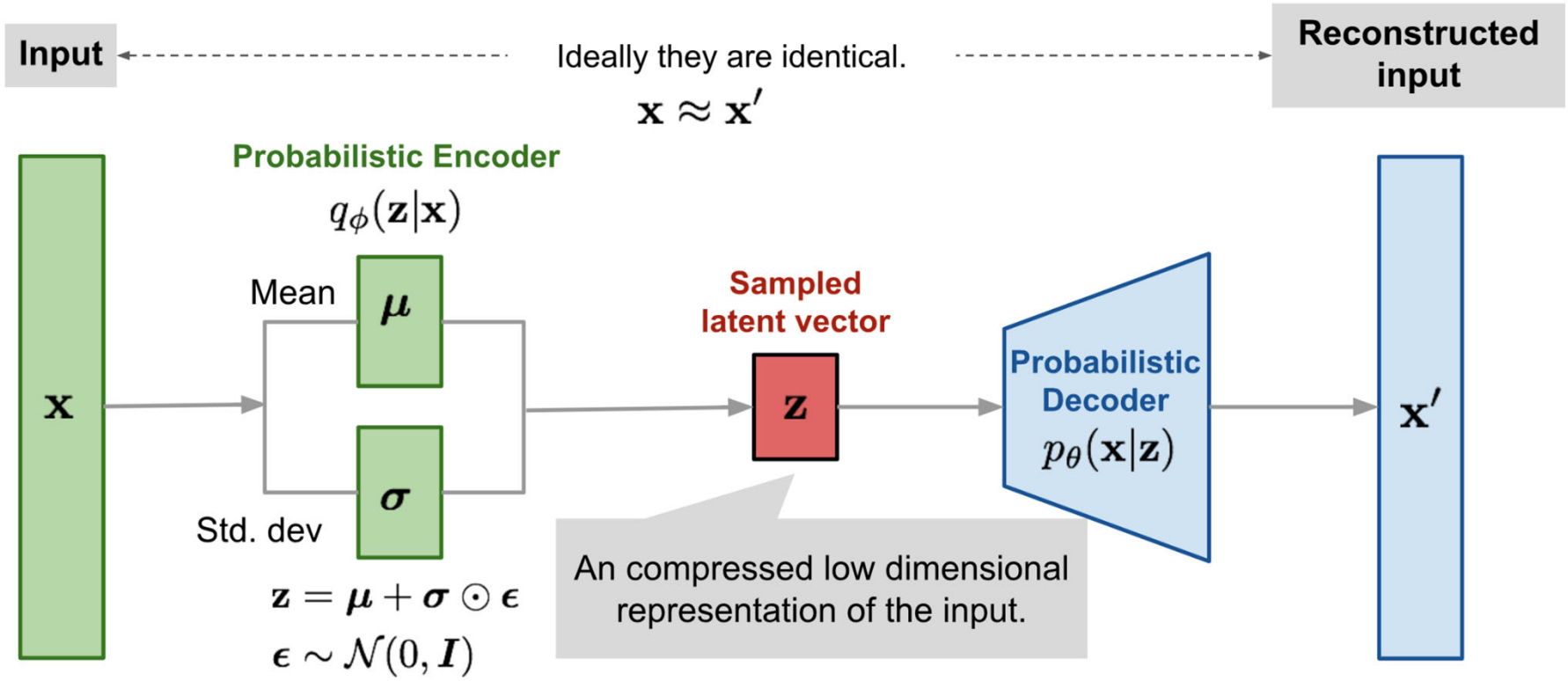
VAE
Gaussian priors and posteriors

- Putting it all together



VAE
Gaussian priors and posteriors

- Additional illustration



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 $z \in R^J$: $-D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log\left((\sigma_j)^2\right) - (\mu_j)^2 - (\sigma_j)^2)$
- proof
  - $D_{KL}(q_\phi(z)||p(z)) = \int \quad q_\phi(z) \log \frac{q_\phi(z)}{p(z)} dz$
- 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 $\int q_\phi(z) \log p(z) dz = \int \mathcal{N}(z; \mu, \sigma) \log \mathcal{N}(z; 0,1) dz$
- $$= E_{q_\Phi}[\log \mathcal{N}(z; 0,1)]$$
  $$= E_{q_\Phi}\left[\log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right)\right)\right]$$
  $$= E_{q_\Phi}\left[-\frac{1}{2}\log 2\pi - \frac{z^2}{2}\right]$$
  $$= -\frac{1}{2}\log 2\pi - \frac{1}{2} E_{q_\Phi}\ [z^2]$$
- 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 $\int \boldsymbol{q_\phi(z)\, log\, p(z) dz = -\frac{1}{2} log\, 2\pi - \frac{1}{2}(\mu^2 + \sigma^2)}$

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
  - $-D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{J}(1 + \log\left(\left(\sigma_j\right)^2\right) - \left(\mu_j\right)^2 - \left(\sigma_j\right)^2)$
- Reproduction term
  - $\log\big(p(x|z)\big) = \sum_{j=1}^{D}\frac{1}{2}\log(\sigma_j^2(z)) + \frac{(x_j-\mu_j(z))^2}{2\sigma_j^2(z)}$
- 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?

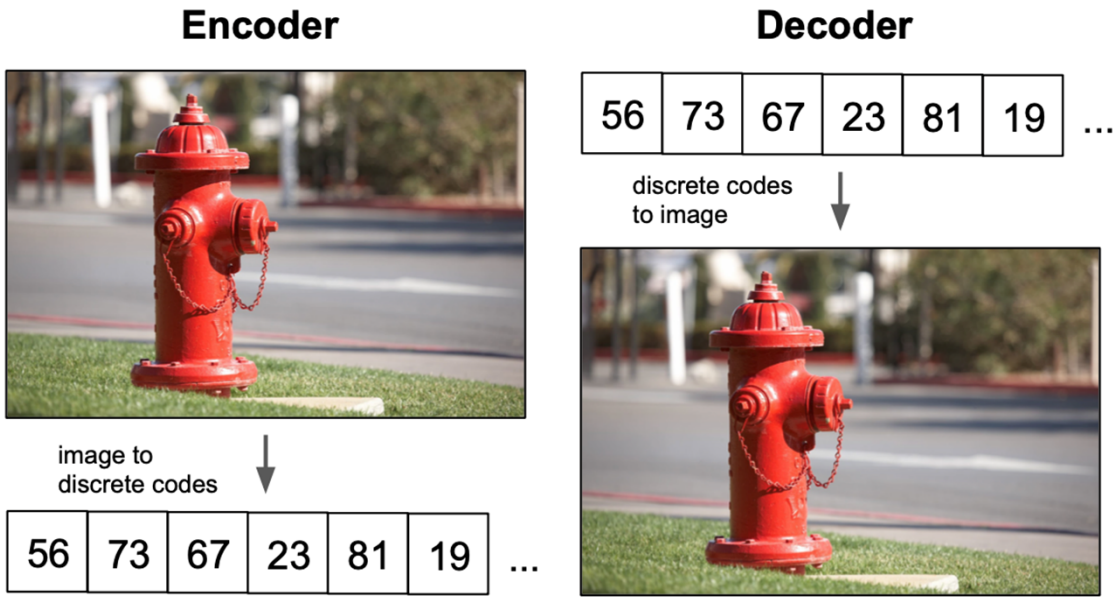
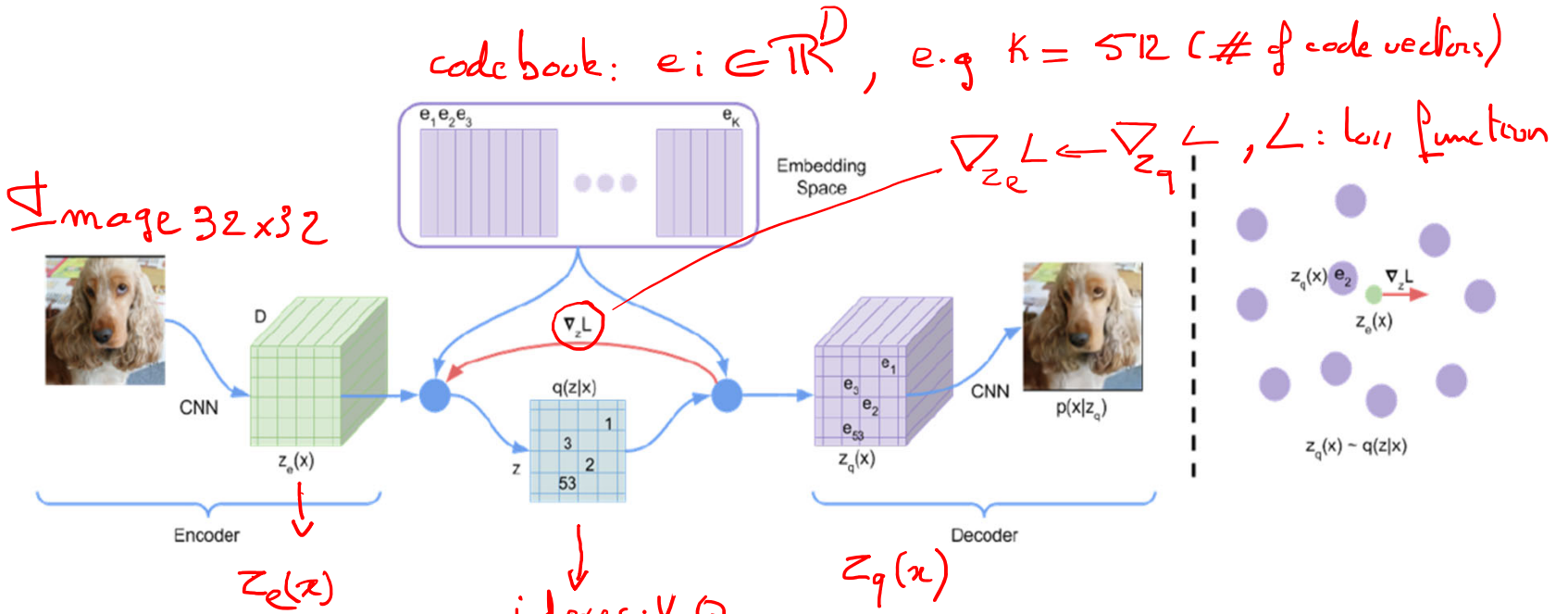


Fig: https://ml.berkeley.edu/blog/posts/vq-vae/

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



Learning discrete distributions: VQ-VAE

- Loss function
  - $L = \| x - Dec\left(z_q(x)\right) \|^2 + \| sg\left(z_e(x)\right) - z_q(x) \|^2 + \beta \| z_e(x) - sg\left(z_q(x)\right) \|^2$
  - 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**
    - $\| sg\left(z_e(x)\right) - z_q(x) \|^2$: 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)*, *MI*, 1–14. http://arxiv.org/abs/1312.6114

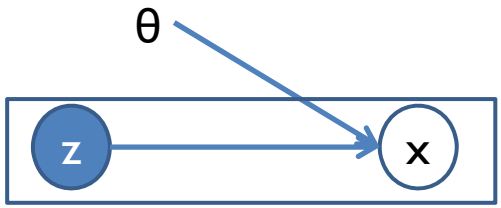
Generative Adversial Networks - GANs

Ian J. Goodfellow, et al. 2014

There has been a strong hype for GANs for several years - O(10000) GAN papers on Arxiv

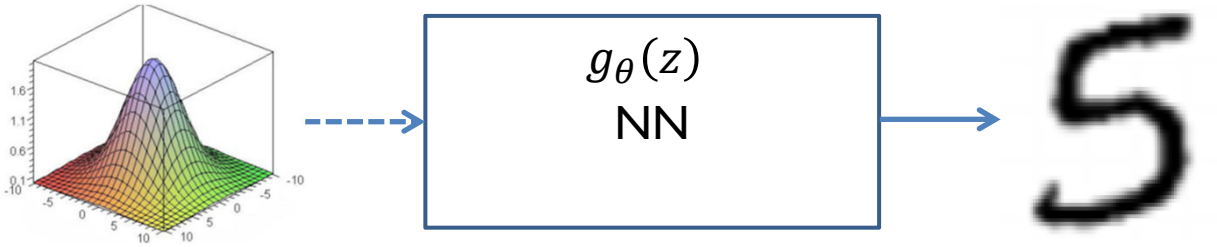
GANs

- Generative latent variable model



  - Given Samples $x^1, \ldots, x^N \in R^n$, 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 $g_\theta : R^q \rightarrow R^n$ such that:

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



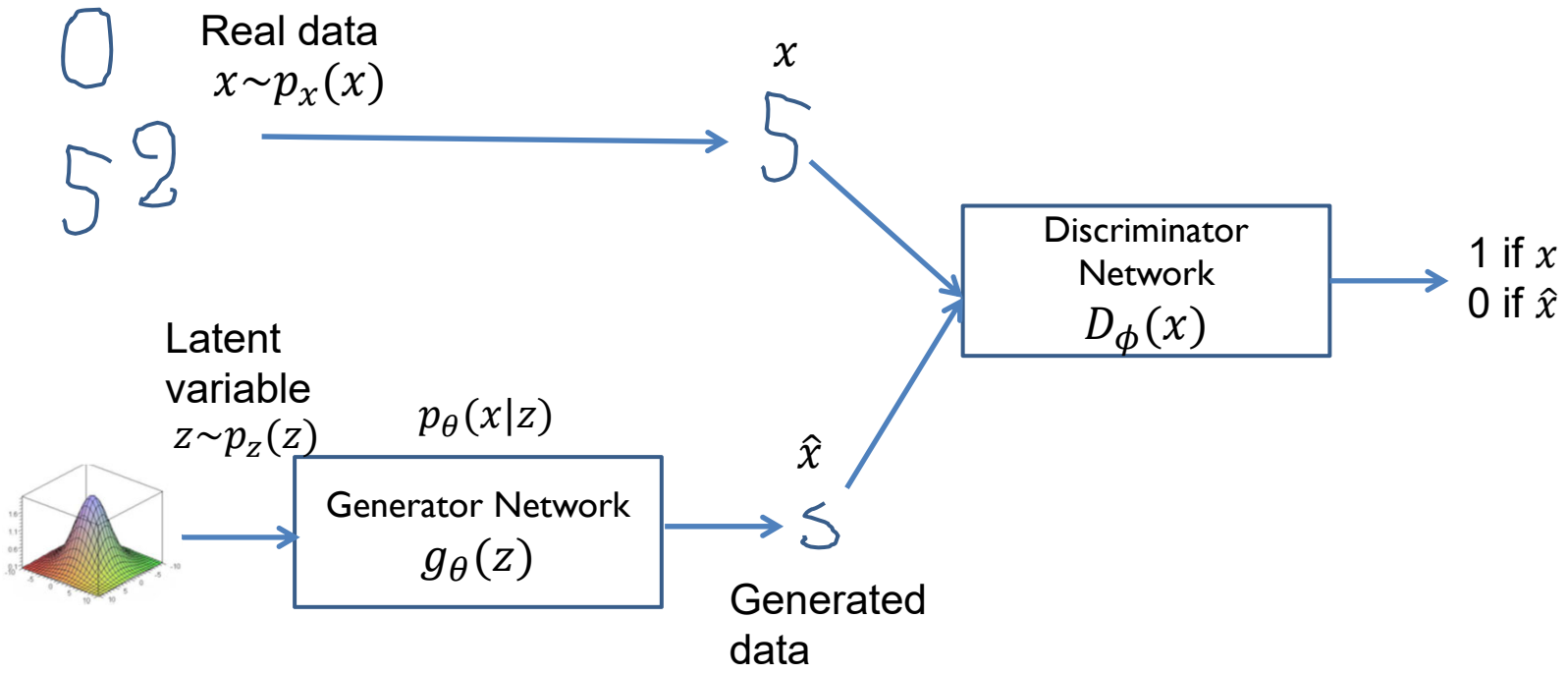
- Different solutions for measuring the similarity between $p_\theta(x)$ and $p_x(x)$
  - In this course: binary classification
- Note:
  - Once trained, sample from z directly generates the samples $g_\theta(z)$
  - Different from VAEs and Flows where the NN $g_\theta(.)$ generate distribution parameters

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

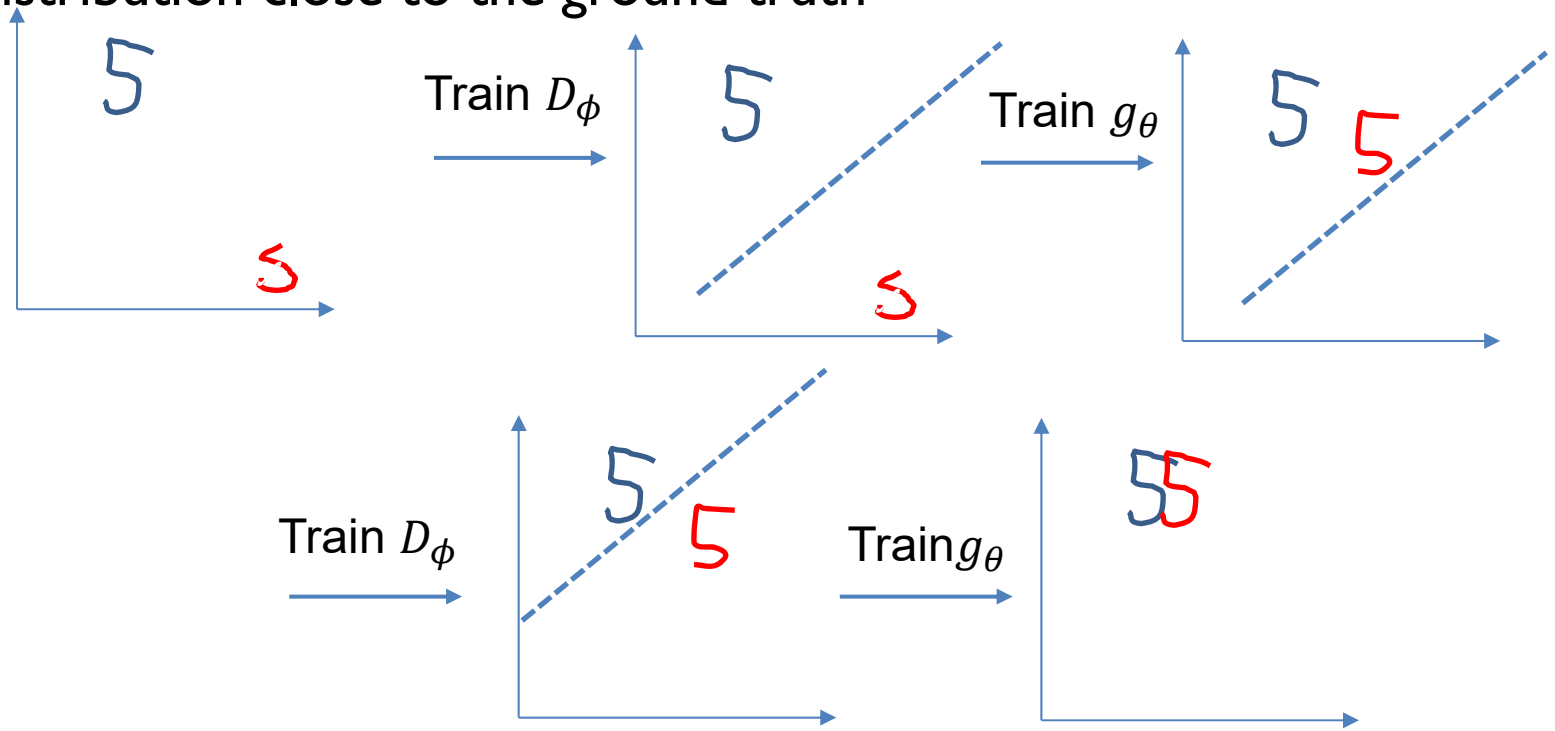
- Discriminator is presented alternatively with true (x) and fake ($\hat{x} = g_\theta(z)$)data



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
  - $\min_{\theta} \max_{\phi} L(D_\phi, g_\theta) = E_{x \sim p_x(x)}[log D_\phi(x)] + E_{z \sim p_z(z)}[\log\left(1 - D_\phi\left(g_\theta(z)\right)\right)]$
    - $g_\theta : R^q \rightarrow R^n$ 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
  - $L(D_\phi, g_\theta) = E_{x\sim p_x(x)}[logD_\phi(x)] + E_{x\sim p_\theta(x)}[\log\left(1 - D_\phi(x)\right)]$
    - 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
  - $\mathrm{D}^*(x) = \frac{p_X(x)}{p_X(x)+p_\theta(x)}$
    - 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
    - $Max_D\, L(D,G) = E_{x\sim p_X(x)}[logD(x)] + E_{\mathrm{x}\sim p_\theta(x)}[\log(1-D(x))]$ is then obtained for:
      - $\mathrm{D}^*(x) = \frac{p_X(x)}{p_X(x)+p_\theta(x)}$

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

- Let $C(g) = \max_D L(g, D) = L(g, D^*)$
- It si easily verified that:
  - $C(g) = -\log 4 + KL\left(p_X(x); \frac{p_X(x)+p_\theta(x)}{2}\right) + KL\left(p_\theta(x); \frac{p_X(x)+p_\theta(x)}{2}\right)$
  - Since $KL(p; q) \geq 0$ and $KL(p; q) = 0$ iff $p = q$
    - $C(g)$ is minimum for $p_\theta = p_X$ 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_X
- 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
  - $\frac{p_X(x)}{p_\theta(x)} = \frac{p(x|y=1)}{p(x|y=0)} = k\frac{p(y=1|x)}{p(y=0|x)} = k\frac{D^*(x)}{1-D^*(x)}$ 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)

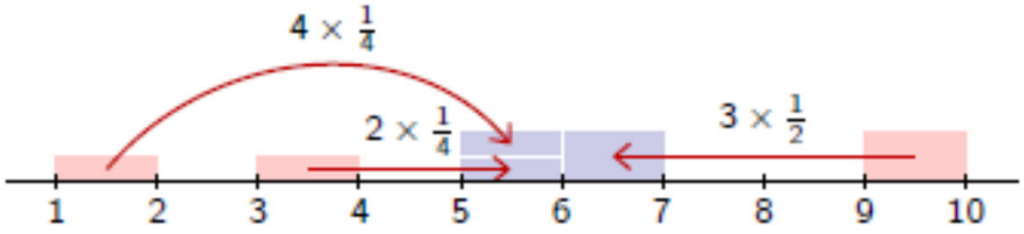
- If a generated sample is well classified but far from the real data distribution, there is no reason for the generator to be updated
- LS-GAN replaces the cross entropy loss with a LS loss which penalizes generated examples by moving them close to the real data distribution.
- The objective becomes
  - $L(D) = E_{x\sim p_X(x)}[(D(x) - b)^2] + E_{z\sim p_z(z)}[(D(g(z)) - a)^2]$
  - $L(g) = E_{z\sim p_z(z)}\left[\left(D\left(g(z)\right) - c\right)^2\right]$
  - 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
  - $W(p_X, p_\theta) = \inf_{\gamma \in \Pi(p_X, p_\theta)} E_{(x,x')\sim\gamma}[\| x - x' \|]$
    - where $\Pi(p_X, p_\theta)$ is the set of distributions over X^2 , with $X \subset R^n$ the space of data, whose marginals are respectively $p_X(x)$ and $p_\theta(x)$, $\| x - x' \|$ is the Euclidean norm.
  - Intuitively,
    - $W(,)$ is the minimum amount of work required to transform $p_X(x)$ to $p_\theta(x)$ – see next slide
    - it makes sense to learn a generator g minimizing this metric
      - $g^* = argmin_G W(p_X, p_\theta)$

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
- $p_X(x) = \int_y \gamma(x, y)dy$ 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 amount of mass multiplied by the distance it moves, then the transport cost is: $\gamma(x, y). \| x - y \|$ and the minimum transport cost is $\inf_{\gamma \in \Pi(p_X, p_\theta)} E_{(x,x') \sim \gamma}[\| x - x' \|]$

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

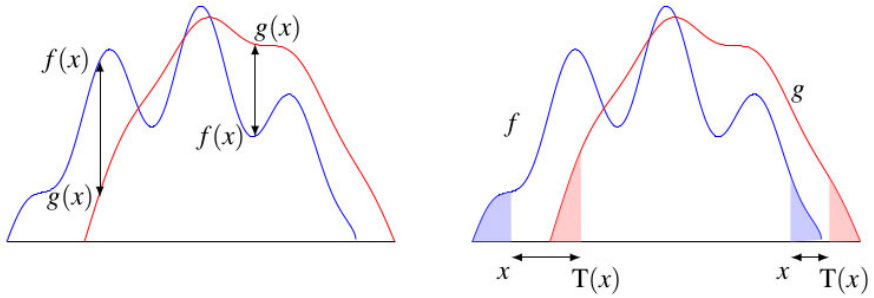


Fig. Santambrogio, 2015

- 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:
  - $W(p_X, p_\theta) = \sup_{\|f\|_L \leq 1} E_{x \sim p_X}|f(x)| - E_{x \sim p_\theta}|f(x)|$
  - 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. $\parallel f \parallel_L \leq 1$ 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: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, \ldots, n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of priors.
    $g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

From Arjovski 2017

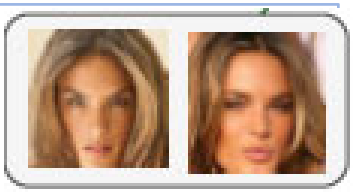
GANs examples
Deep Convolutional GANs (Radford 2015) - Image generation

- 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

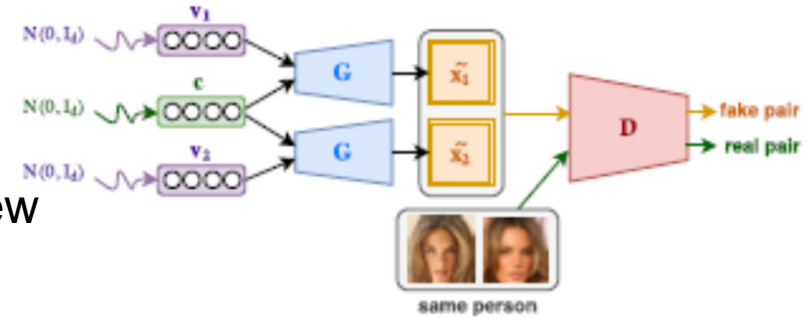
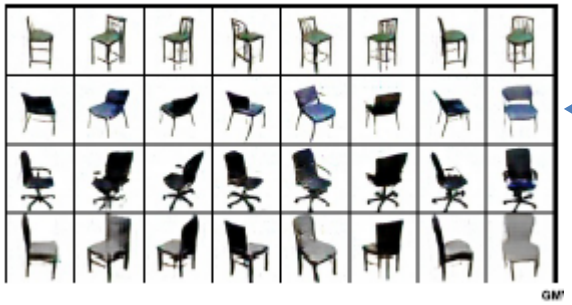


Fig. Chen 2018

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

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

Conditional GANs (Mirza 2014)

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

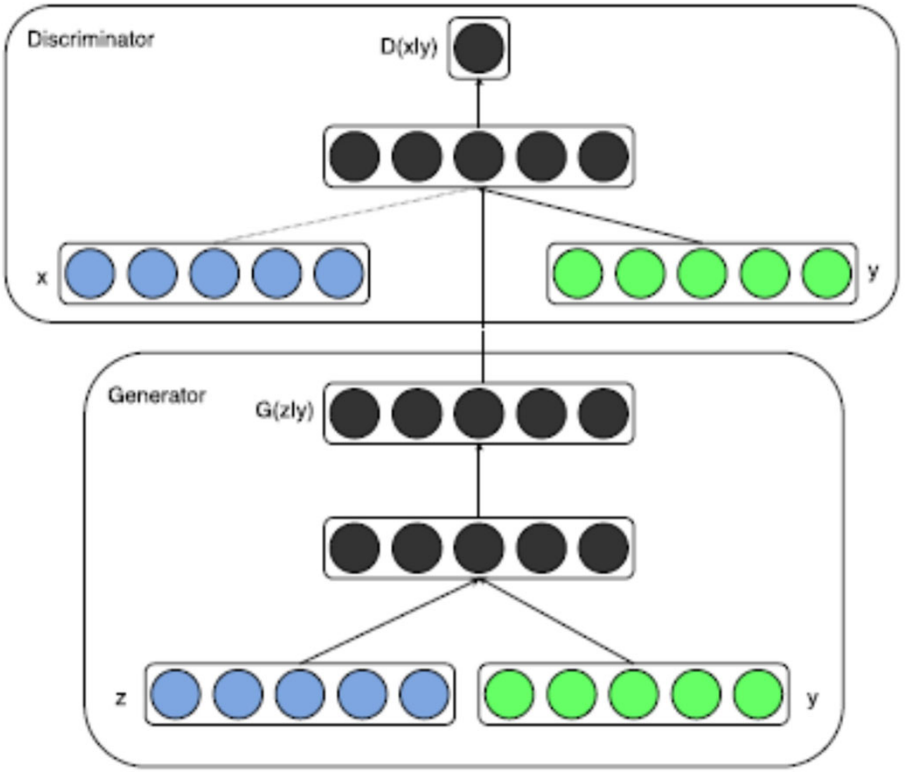


Fig. (Mirza 2014)

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

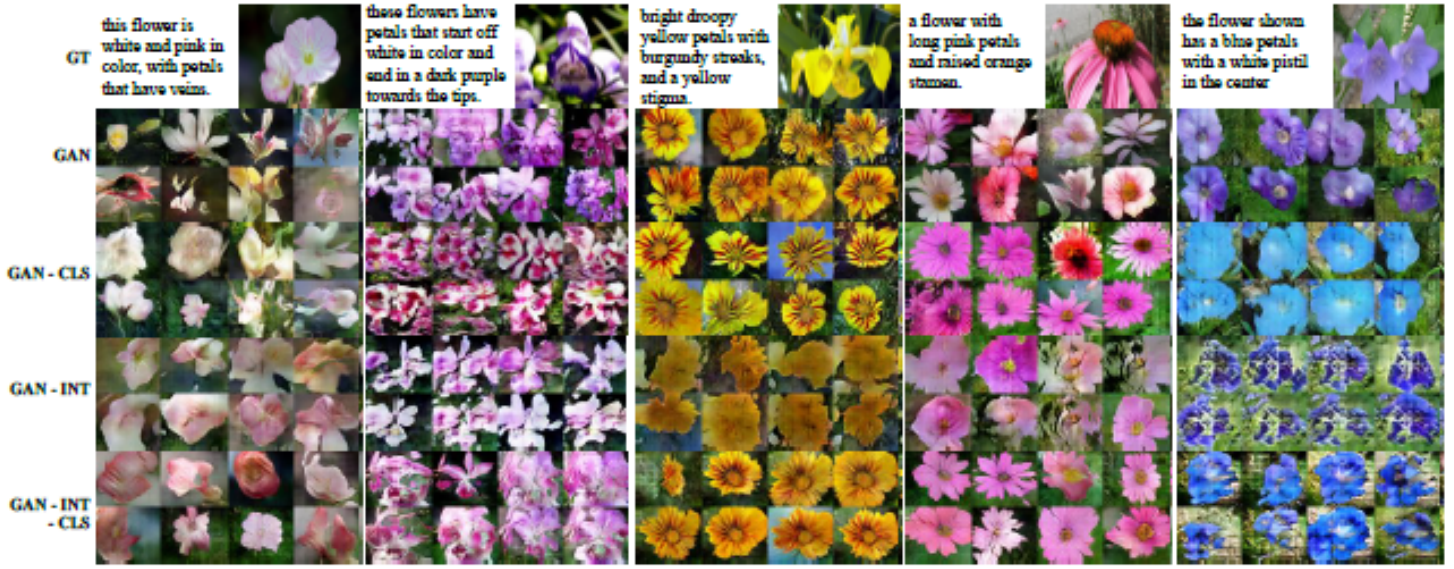
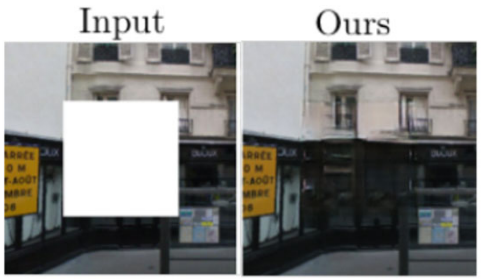


Figure 4. Zero-shot generated flower images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. All variants generated plausible images. Although some shapes of test categories were not seen during training (e.g. columns 3 and 4), the color information is preserved.

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
- $\min_g \max_D L(D,g) = E_{\substack{x\sim p_X(x) \\ y\sim p(y)}}[logD(x,y)] + E_{\substack{z\sim p(z) \\ y\sim p(y)}}[\log\big(1 - D(g(z,y),y)\big)]$
  - 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 $g(z,y)$ « 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)

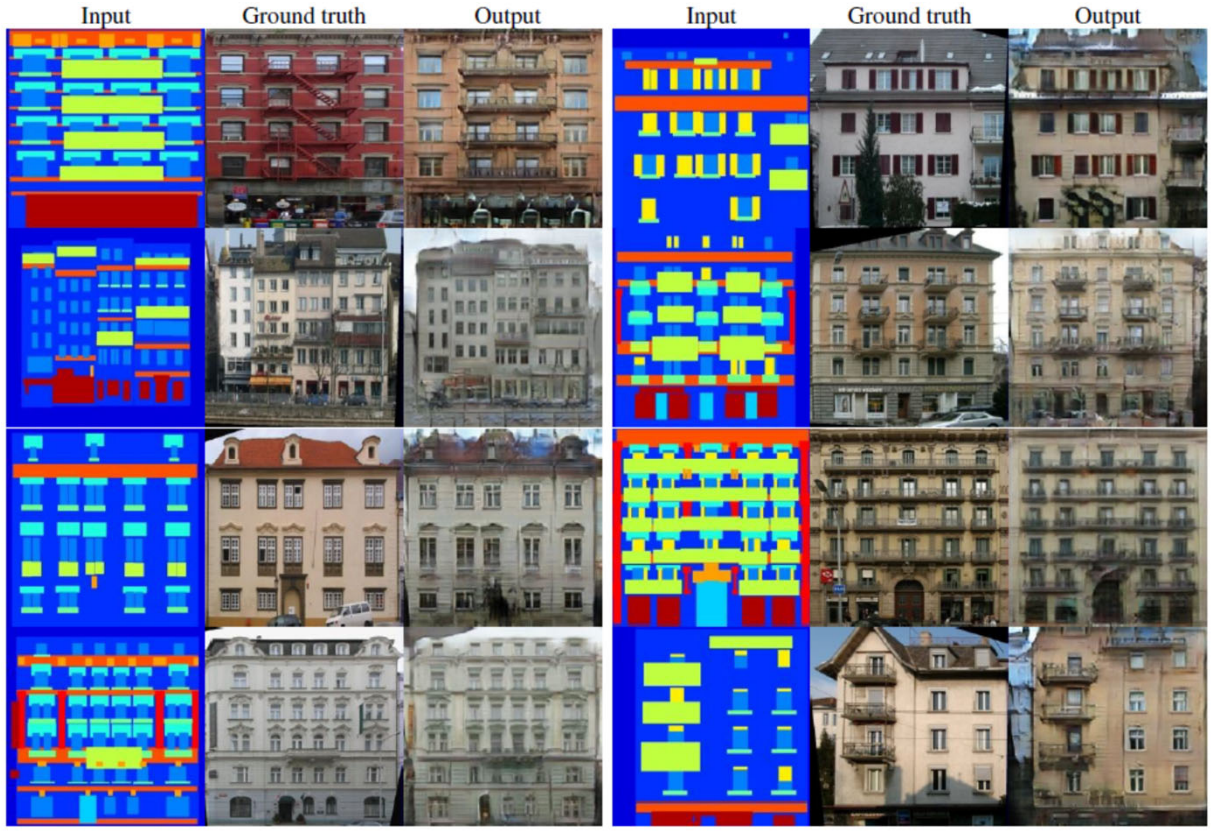


Figure 13: Example results of our method on facades labels→photo, compared to ground truth.

Fig. (Isola 2016)

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

- Failure examples

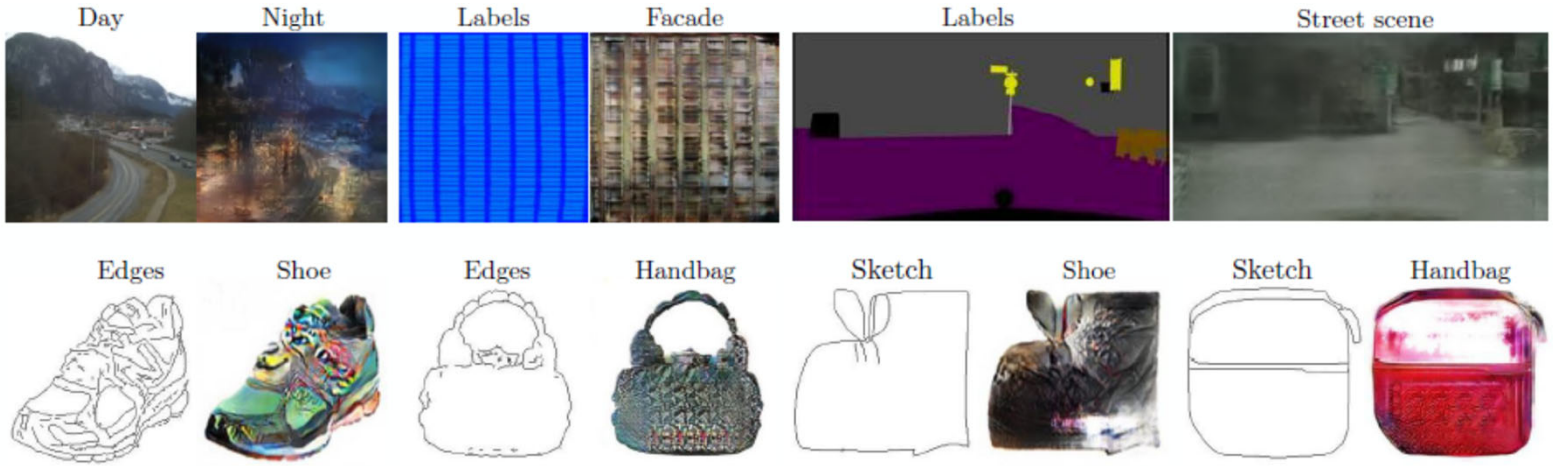


Figure 20: Example failure cases. Each pair of images shows input on the left and output on the right. These examples are selected as some of the worst results on our tasks. Common failures include artifacts in regions where the input image is sparse, and difficulty in handling unusual inputs. Please 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)

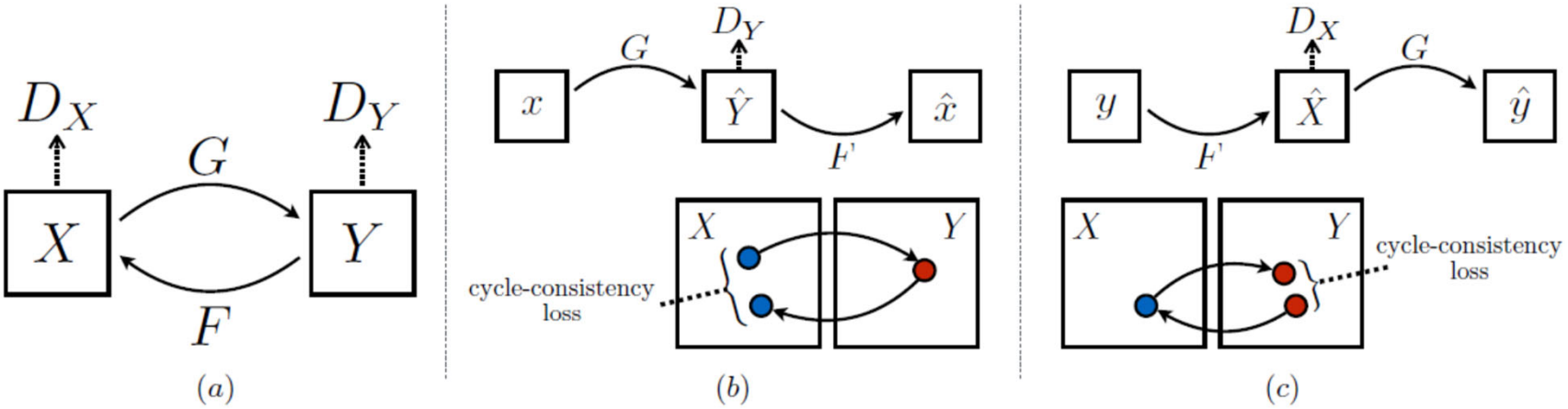


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X, F, and X. To further regularize the mappings, 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

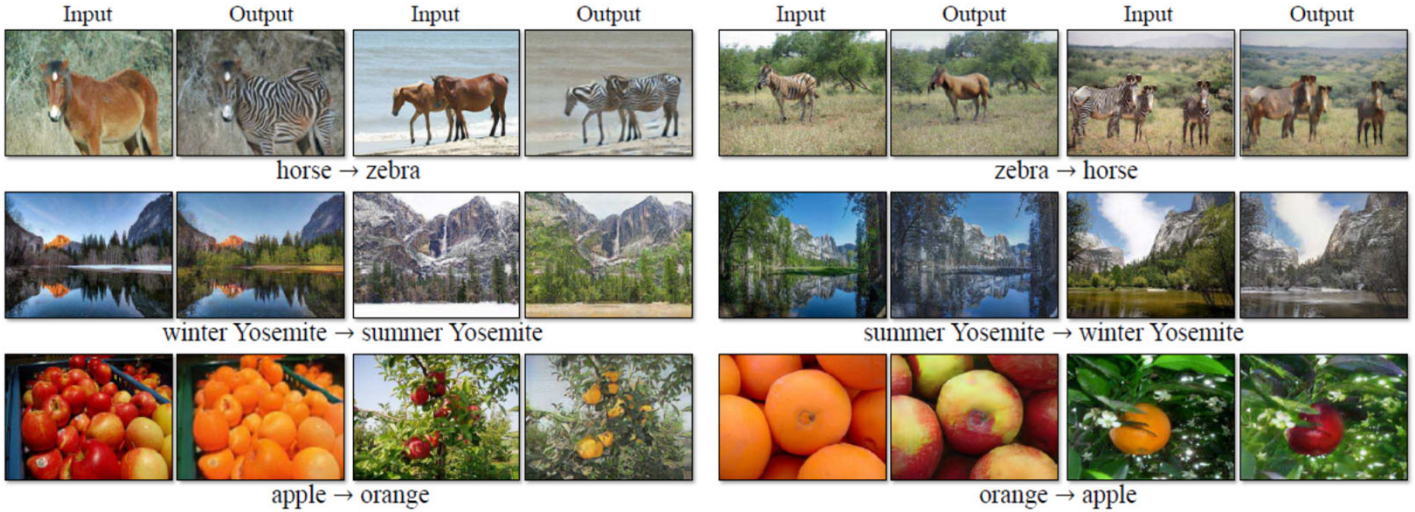
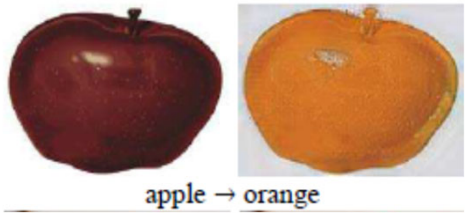


Figure 7: Results on several translation problems. These images are relatively successful results – please see our website for more comprehensive results.

Input Monet Van Gogh Cezanne Ukiyo-e

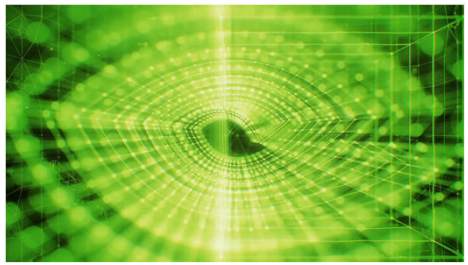
- Failures

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

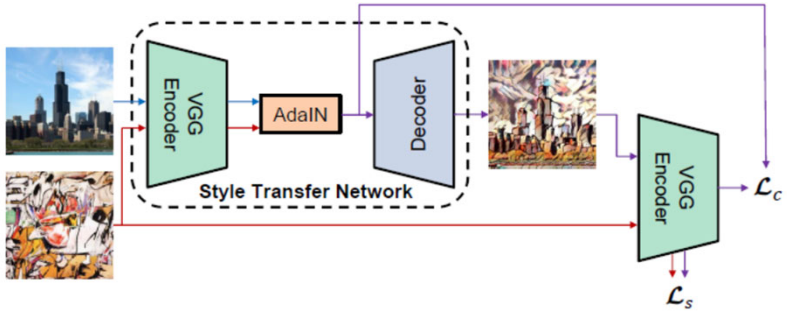


Figure 2. An overview of our style transfer algorithm. We use the first few layers of a fixed VGG-19 network to encode the content and style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is learned to invert the AdaIN output to the image spaces. We use the same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

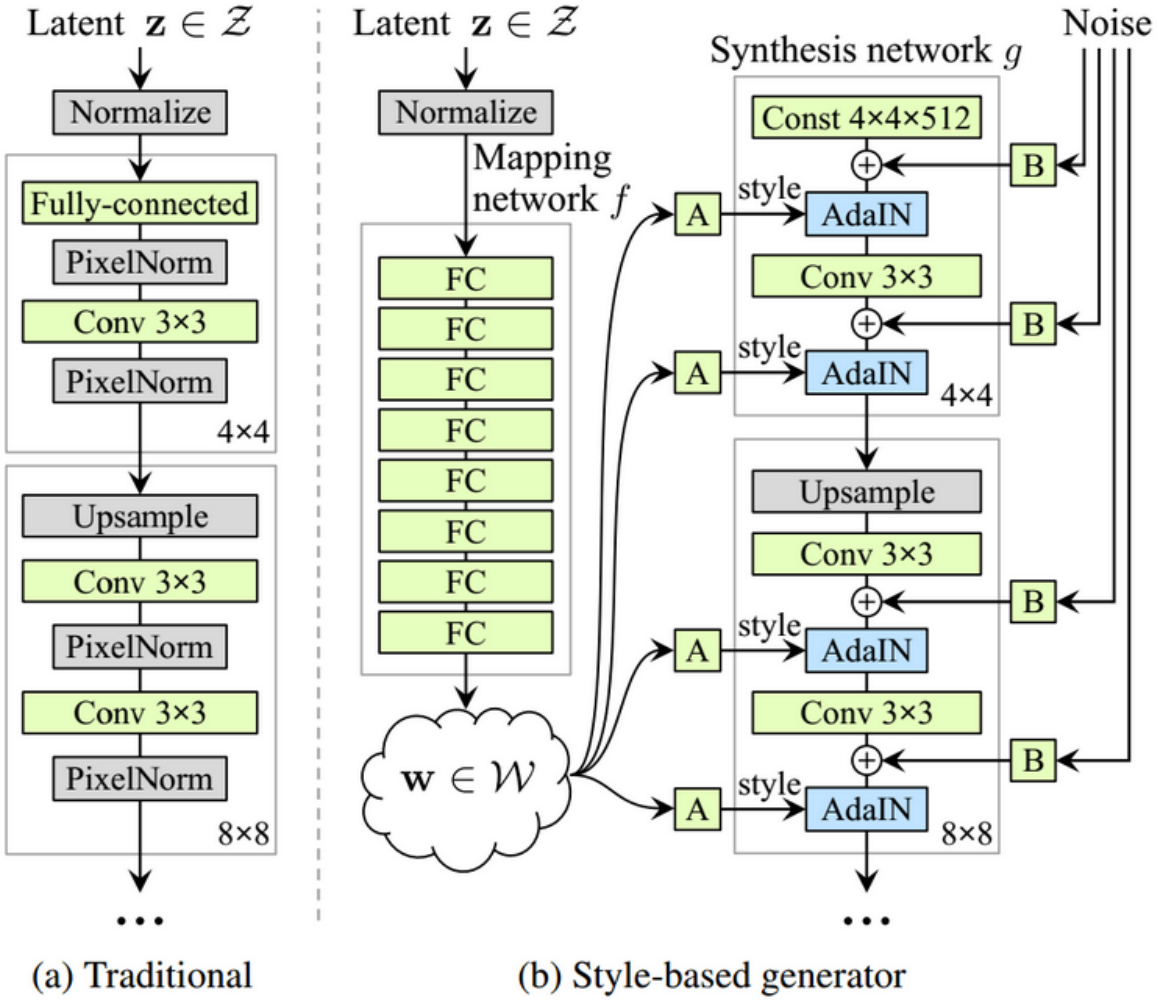
(Huang 2017)

Style Gan
Preliminary: Adaptive Instance Normalization (AdaIN)

- (Huang 2017) examples

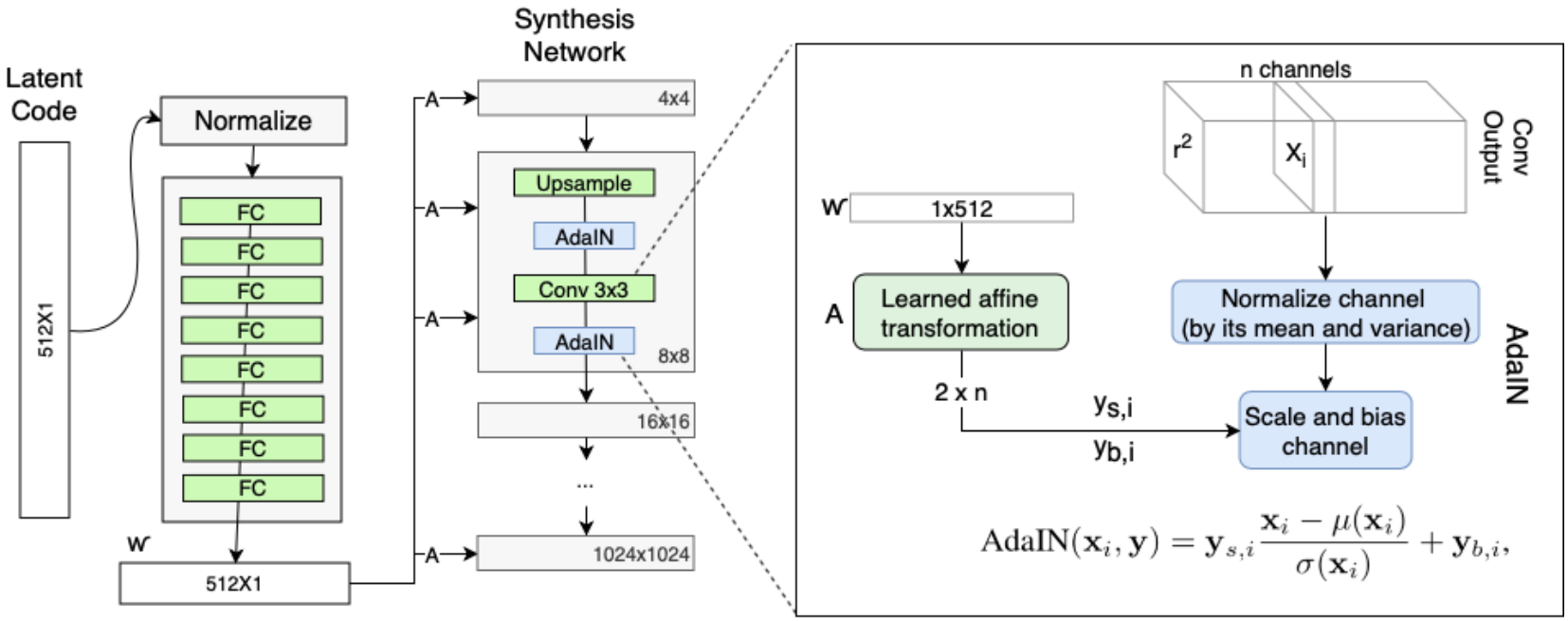
Architecture of Style Gan

Karras et al. 2019



- 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

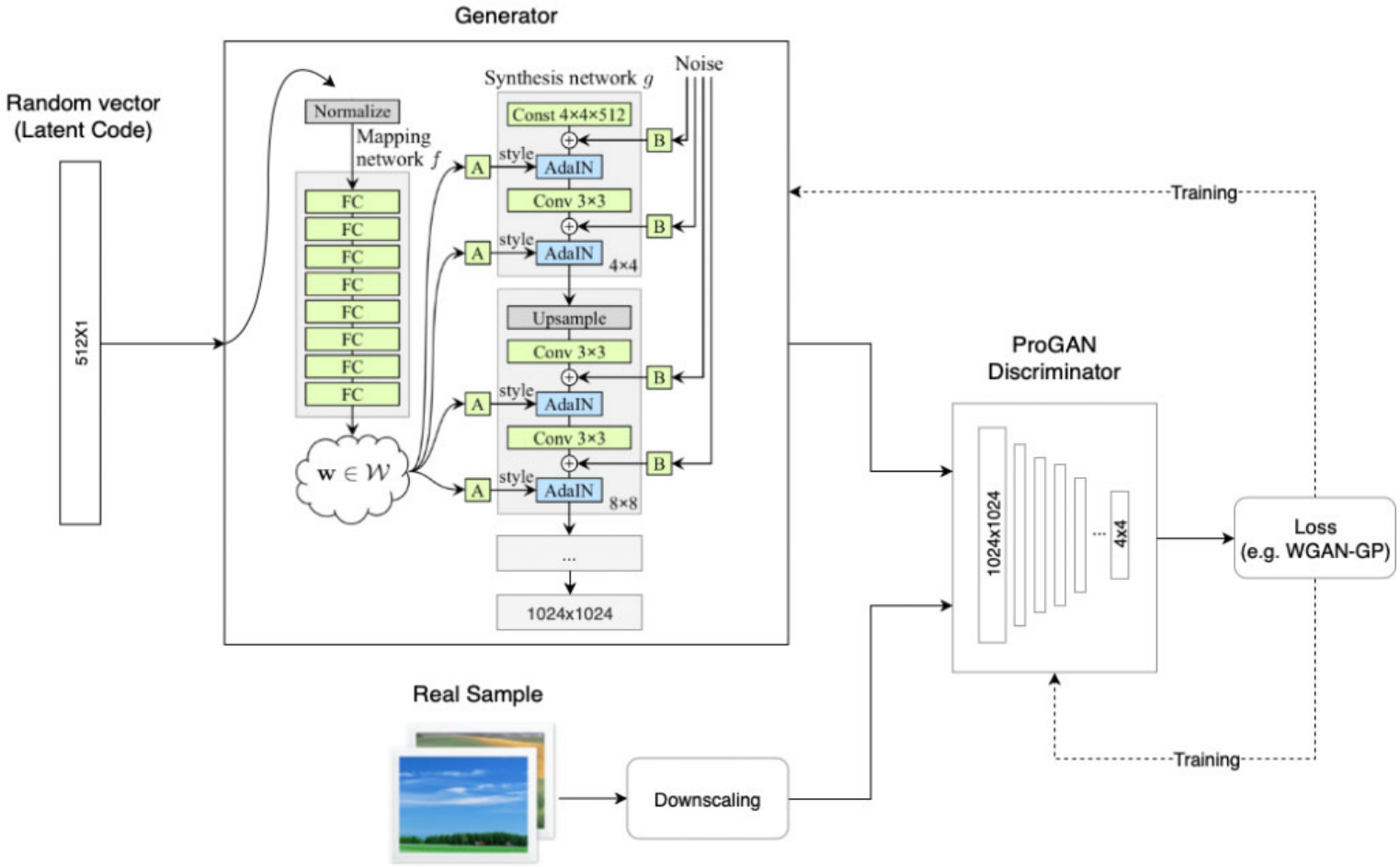
Architecture of Style Gan



- 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.

Flow Matching

Used in different forms in many recent image/ video generators, e.g. Google Banana, OpenAI Sora, Google Veo, …

Flow matching - key notions

- Ordinary differential equation ODE
  - An ODE describes the evolution of a system that depends of a variable, usually called time
  - $\frac{d}{dt}x(t) = v_t(x(t))$
  - $x(0) = x_0$ - initial condition
    - the evolution function $v_t(x(t))$ is called the vector field of the ODE
- Trajectory
  - The solution of an ODE is a trajectory, i.e. a function that maps time to a location in $\mathbb{R}^d$
    - $x(t)$: $[0,1] \rightarrow \mathbb{R}^d$
    - Said otherwise, a trajectory is the integral curve of the vector field starting from an initial condition

Fig. Lipman et al. 2024

Flow matching - key notions

- Velocity field
  - A vector field v_t assigns a velocity vector to every point $x \in \mathbb{R}^d$
  - $v_t: [0,1] \times \mathbb{R}^d \rightarrow \mathbb{R}^d$
  - It describes the instantaneous velocity (direction and speed) of a « particle » at each time t and position x
  - You can visualize it as a field of arrows, where each arrow shows how a particle located at x would move at that instant.

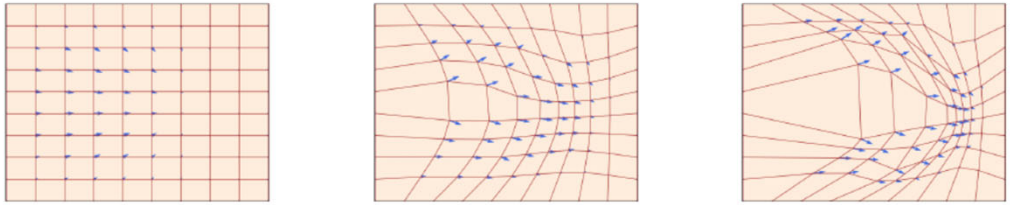


Figure 6 A flow $\psi_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$ (square grid) is defined by a velocity field $u_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$ (visualized with blue arrows) that prescribes its instantaneous movements at all locations. We show three different times t.

Fig. Lipman et al. 2024

Flow matching - key notions

- Flow of the ODE
  - The **flow Φ_t** is the *map* that takes an initial condition x_0 and a time t, and gives the state of the system at that time: $\Phi_t(x_0) = x(t; x_0)$
  - $\Phi_t: [0,1] \times \mathbb{R}^d \rightarrow \mathbb{R}^d, (x_0, t) \rightarrow \Phi_t(x_0)$
  - $\frac{d}{dt}\Phi_t(x_0) = v_t(\Phi_t(x_0))$
  - $\Phi_t(x_0) = x_0$ - initial condition
  - The flow is determined by the vector field

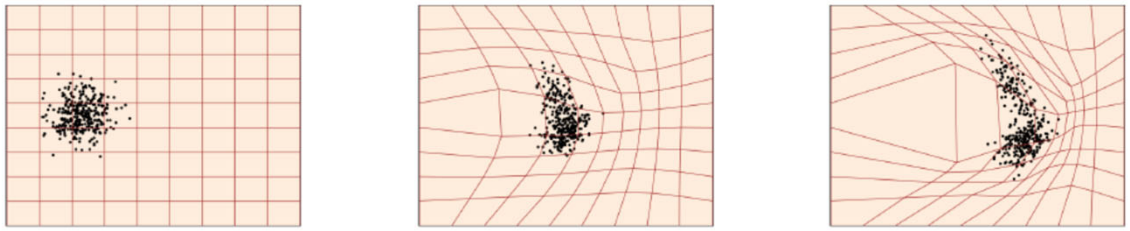


Figure 5 A flow model $X_t = \psi_t(X_0)$ is defined by a diffeomorphism $\psi_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$ (visualized with a brown square grid) pushing samples from a source RV X_0 (left, black points) toward some target distribution q (right). We show three different times t.

Fig. Lipman et al. 2024

Flow existence and uniqueness

Theorem 1

If $v\colon [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ is continuously differentiable with a bounded derivative, the ODE, $\frac{d}{dt}\Phi_t(x_0) = v_t\big(\Phi_t(x_0)\big), \Phi_t(x_0) = x_0,$ has a unique solution given by flow Φ_t. Φ_t is a diffeomorphism for all t, i.e. continuously differentiable with a continuously differentiable inverse.

Flow matching - key notions

Vector fields define ODEs - ODE solutions are flows

Concept	definition	Relation
Vector field $v_t(x)$	Function assigning a velocity vector to each point	Defines the dynamics
Trajectory $x(t; x_0)$	Solution curve starting from x_0	Integral curve of v_t
Flow $\Phi_t(x_0)$	Map giving position after time t	Generated by integrating v_t

Flow matching - key notions

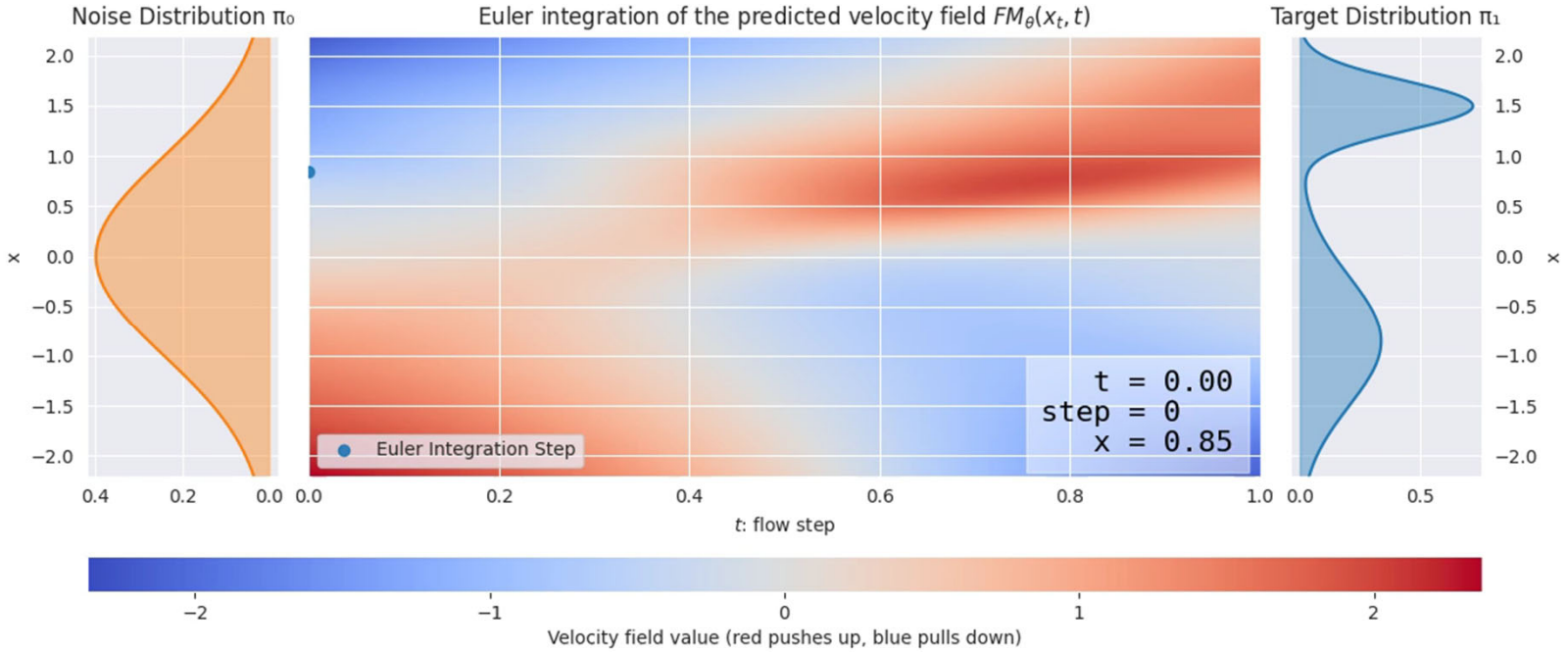
- To *simulate* the flow, we need to **numerically integrate** the ODE. Time is discretized between 0 and 1: $0, h, 2h, \ldots, 1$
- The simplest numerical integration method is **Euler's scheme:**

Start at x_0
Iterate $x_{k+h} = x_k + hv_t(x_k), k = 0, h, \ldots, 1 - h$

- This approximates the true trajectory $x_{k+h} \approx x(t_{k+h})$

Flow matching - key notions

- Euler integration



- Velocity field is shown as a heatmap
- The figure shows the integration of a sample from the noise distribution to the target distribution (an approximation of the true trajectory), using the velocity field

Credits: https://peterroelants.github.io/posts/flow_matching_intro/

Flow matching

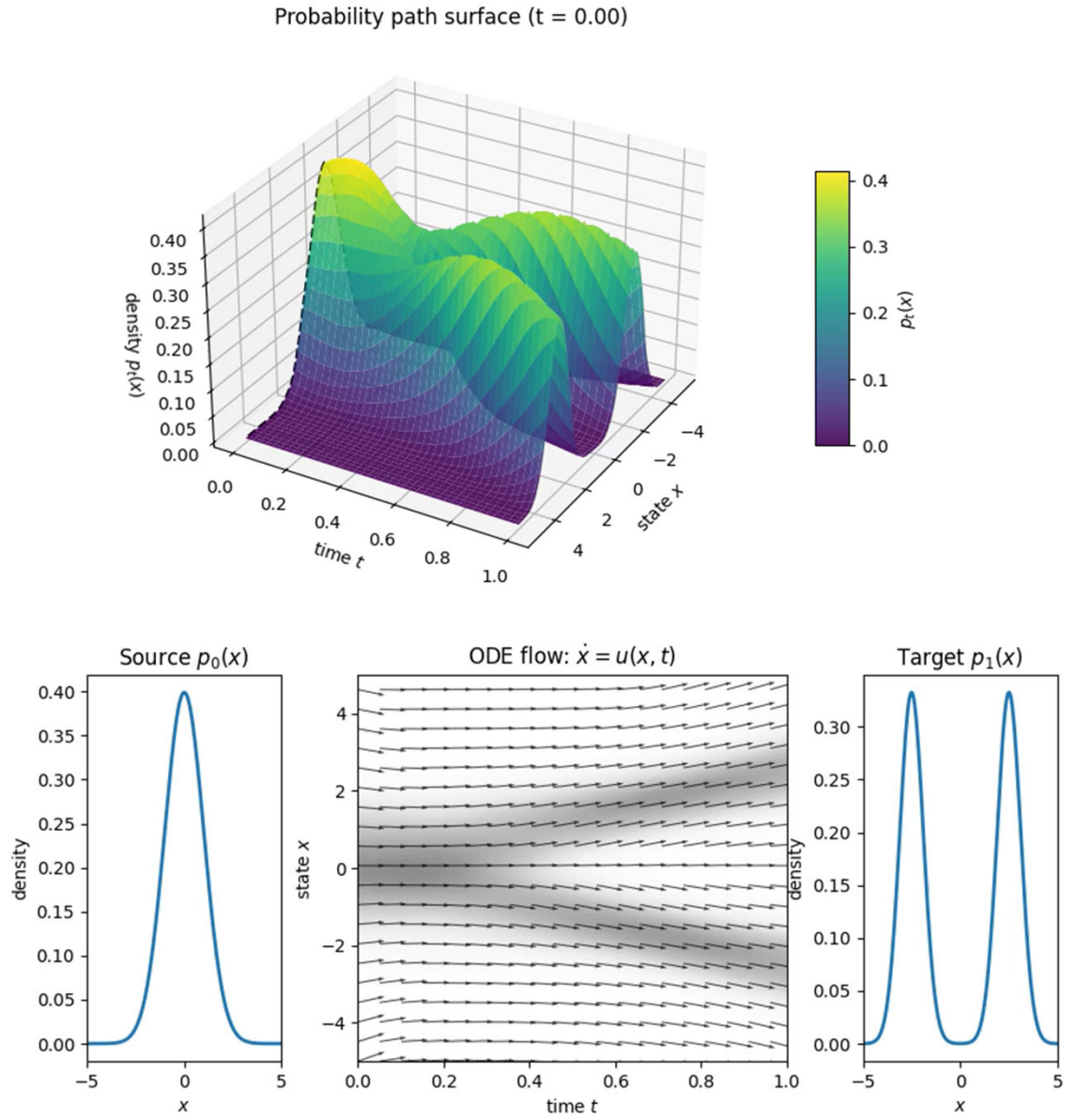
- Objective
  - Learn a mapping from distribution $p_0(x)$ to a target distribution $p_1(x)$
- Probability path
  - Flow matching learns a probability path $p_t, t \in [0,1]$.
    - This is a time dependent probability that interpolates between two distributions p_0 and p_1
      - $p_t: [0,1] \times \mathbb{R}^d \rightarrow \mathbb{R}^+$
      - $p_t(t,.)$ is a probability function of variable x
  - The probability path is generated by a velocity field v_t which defines the instantaneous velocities of samples (direction and speed)
    - $v_t(t,x)$ provides the instantaneous velocity at time t and sample x
  - The velocity field generates the probability path p_t, if its associated flow Φ_t satisfies
    - $x_t := \Phi_t(x_0) \sim p_t$ for $x_0 \sim p_0$

Flow matching

- The velocity field is the only tool required to sample from p_t using the ODE $\frac{d}{dt}x(t) = v_t(x(t))$
  - Hence the objective of flow matching is to **learn a parameterized vector field $v_t^\theta(x_t)$**
  - Once trained, the flow is defined by the ODE

$$x_0 \sim p_0$$
$$\frac{dx_t}{dt} = v_t^\theta(x_t)$$
with $x_t = \Phi_t(x_0)$

Illustration: probability path and velocity field



Flow matching inference

- Sampling (inference)
- Once the parameterized vector field is learned, the target distribution can be sampled by solving the ODE

$x_0 \sim p_0$

Integrate the ODE $\frac{dx_t}{dt} = v_t^\theta(x_t)$ from $t = 0$ to $t = 1$

e.g. Euler integration $x_{t+h} = x_t + hv_t^\theta(x_t)$

Output x_1

Conditionals and marginals probability paths

- Flow matching will learn the (marginal) probability path as the agregation of conditional probability path
- Let $z \in \mathbb{R}^d$, a conditional probability path is a set of distributions $p_t(x|z)$ such that:
  - $p_0(.|z) = p_0$ and $p_1(.|z) = \delta_z$ for all $z \in \mathbb{R}^d$
  - i.e. it maps distribution p_0 to a single point z
- A conditional probability path defines a marginal path $p_t(x)$ obtained by sampling target points $x_1 \sim p_1$ and then sampling from $p_t\,(x|x_1)$

The marginal probability path p_t results from the agregation of the conditional probability paths $p_{t|1}$ for multiple samplings x_0, x_1

$$p_t(x) = \int p_t\,(x|x_1)p_1(x_1)dx_1$$

Conditional and marginal velocity fields

- Accordingly, one will define the conditional and marginal velocity fields.

  Let $z \in \mathbb{R}^d$, let $v_t(.|z)$ denote a conditional vector field, whose corresponding ODE defines a conditional probability path $p_{t|z}(.|z)$:

$$x_0 \sim p_0, \frac{dx_t}{dt} = v_t(x_t|z) \implies x_t \sim p_t(x|z), t \in [0,1]$$

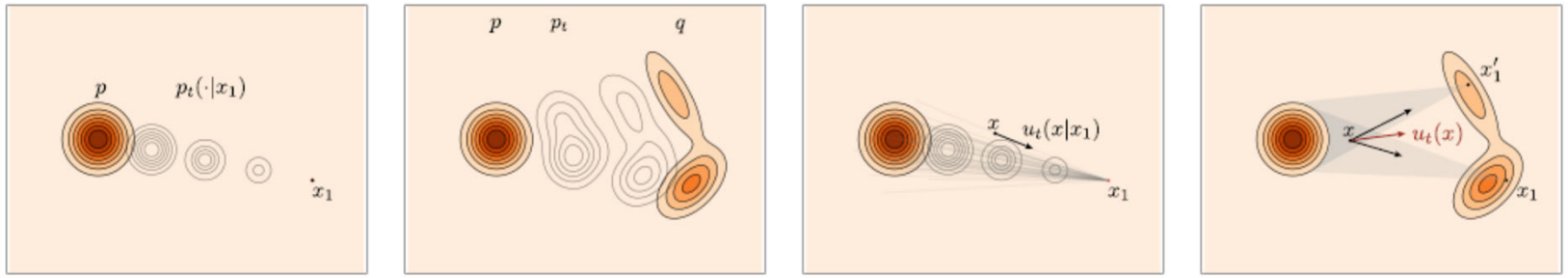
  Then the marginal vector field $v_t(x)$ is defined as:

$$v_t(x) = \int v_t(x|z) \frac{p_t(x|z)p_1(z)}{p_t(x)} dz$$

- It will be shown later
  - That this marginal vector field follows the corresponding marginal probability path
  - Training a flow model proceeds by learning conditional probability paths

Conditionals and marginals probability paths

- Illustrating the conditional/ marginal probability paths and velocity fields



(a) Conditional probability path $p_t(x|x_1)$. **(b)** (Marginal) Probability path $p_t(x)$. **(c)** Conditional velocity field $u_t(x|x_1)$. **(d)** (Marginal) Velocity field $u_t(x)$.

Figure 3 *Path design in Flow Matching.* Given a fixed target sample $X = x_1$, its conditional velocity field $u_t(x|x_1)$ generates the conditional probability path $p_t(x|x_1)$. The (marginal) velocity field $u_t(x)$ results from the aggregation of all conditional velocity fields—and similarly for the probability path $p_t(x)$.

Fig. Lipman et al. 2024

Example: Gaussian conditional probability path

- Let α_t, β_t be two continuous differentiable monotonic functions such that $\alpha_0 = \beta_1 = 0$ and $\alpha_1 = \beta_0 = 1$

The Gaussian conditional path

$$p_t(.|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I)$$

Fulfills

$$p_0(.|z) = \mathcal{N}(\alpha_0 z, \beta_0^2 I) = \mathcal{N}(0, I) \text{ and } p_1(.|z) = \mathcal{N}(\alpha_1 z, \beta_1^2 I) = \delta_z$$

Sampling from the marginal path consists in

$$x_1 \sim p_1, \epsilon \sim \mathcal{N}(0, I) \Longrightarrow x = \alpha_t x_1 + \beta_t \epsilon \sim p_t$$

Conditional and marginal probability paths for the Gaussian probability path

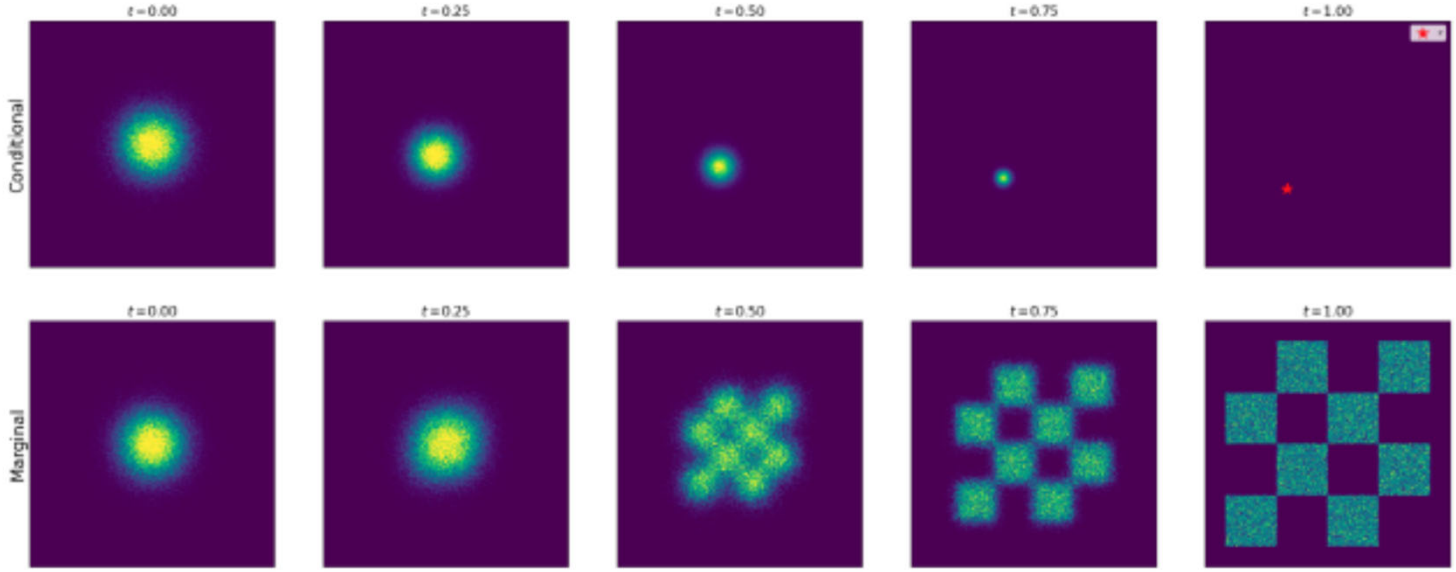


Figure 5: Illustration of a conditional (top) and marginal (bottom) probability path. Here, we plot a Gaussian probability path with $\alpha_t = t, \beta_t = 1 - t$. The conditional probability path interpolates a Gaussian $p_{\text{init}} = \mathcal{N}(0, I_d)$ and $p_{\text{data}} = \delta_z$ for single data point z. The marginal probability path interpolates a Gaussian and a data distribution p_{data} (Here, p_{data} is a toy distribution in dimension $d = 2$ represented by a chess board pattern.)

Fig. Holderrieth& Erives

Recap: Flow, velocity field, probability path

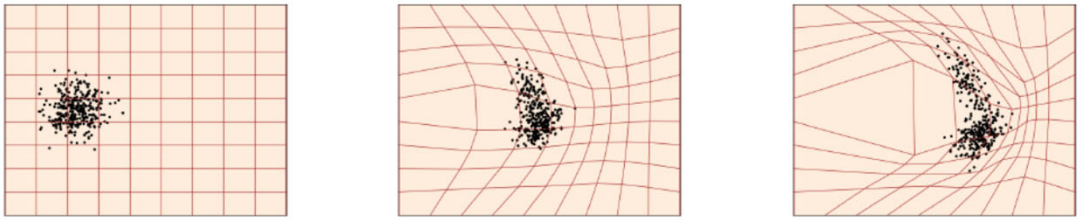


Figure 5 A flow model $X_t = \psi_t(X_0)$ is defined by a diffeomorphism $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with a brown square grid) pushing samples from a source RV X_0 (left, black points) toward some target distribution q (right). We show three different times t.

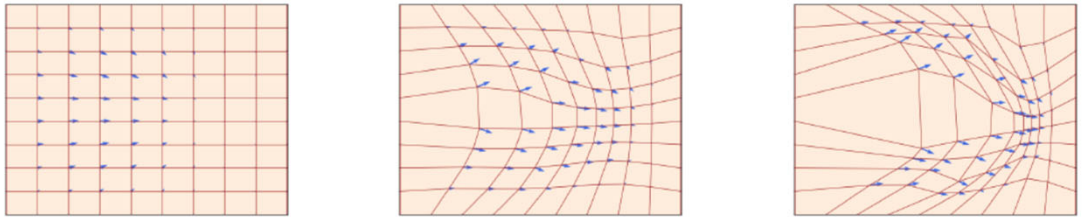


Figure 6 A flow $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (square grid) is defined by a velocity field $u_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with blue arrows) that prescribes its instantaneous movements at all locations. We show three different times t.

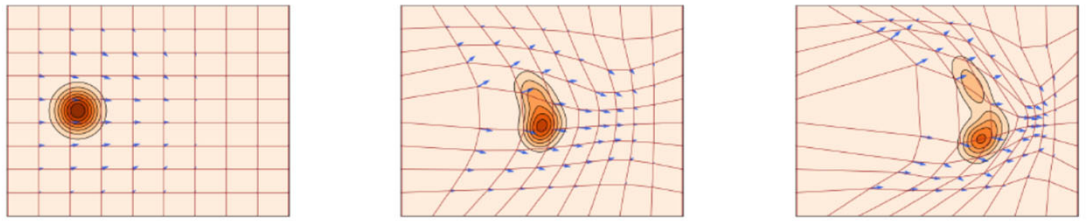


Fig. Lipman et al. 2024

Figure 7 A velocity field u_t (in blue) *generates* a probability path p_t (PDFs shown as contours) if the flow defined by u_t (square grid) reshapes p (left) to p_t at all times $t \in [0, 1)$.

Training

- Flow matching loss
  - v_t^θ should approximate the a target field v_t
- The **flow matching loss** is defined as:
  - $\mathcal{L}_{FM} = E_{t\sim U[0,1],\, x_t\sim p_t}\left[\left\|v_t^\theta(x_t) - v_t(x_t)\right\|^2\right]$
    - With v_t a target field and v_t^θ its parametric approximation
  - Target v_t is unknown (it is untractable), flow matching proposes to optimize instead a conditional loss
    - $\mathcal{L}_{CFM} = E_{t\sim U[0,1],\, x_1\sim p_1(x),\; x_t\sim p_t(x_t|x_1)}\left[\left\|v_t^\theta(x_t) - v_t(x_t|x_1)\right\|^2\right]$
    - The conditional field $v_t(x_t|x_1)$ will have a **tractable analytical expression**, enabling the training - We have the following property;

$$\nabla_\theta \mathcal{L}_{FM} = \nabla_\theta \mathcal{L}_{CFM}$$

minimizing $\mathcal{L}_{CFM}$ is equivalent to miminizing $\mathcal{L}_{FM}$

Training

- CFM introduces an anchor point $x_1 \sim p_1$
  - Instead of modeling directly the marginal p_t on the whole domain of p_1, training is performed around this anchor point
  - Then by sampling several target points x_1 one progressively aggregates the local informations in the target space
  - This simplifies the problem of modeling the marginal p_t because one can often find a conditional vector field satisfying the following equation analytically by hand.

$$x_0 \sim p_0, \frac{dx_t}{dt} = v_t(x_t|z) \implies x_t \sim p_t(x|z), t \in [0,1]$$

Training:building the conditional probability path and the conditional vector field

- Considering $\mathcal{L}_{CFM}$, one needs tractable expressions for $p_t(x_t|x_1)$ and $v_t(x_t|x_1)$
  - There is an infinite number of (conditional) probability paths mapping a distribution p_0 to a target distribution p_1 (and p_0 to x_1)
- A popular choice for the conditional path is the linear path:

$$p_{t|1}(x|x_1) = \mathcal{N}(x|tx_1, (1-t)^2 I)$$

- Using this probability path, let us define the following random variable

$$x_t = \Phi_t(x_0) = tx_1 + (1-t)x_0 \text{ with } x_0 \sim p_0 = \mathcal{N}(O, I)$$
$$x_t \sim \mathcal{N}(x|tx_1, (1-t)^2 I)$$

  - Given $x_0 \sim \mathcal{N}(O, I)$ and $x_1 \sim p_1$, the flow x_t follows a linear path between x_0 and x_1
- The flow defines the following vector field

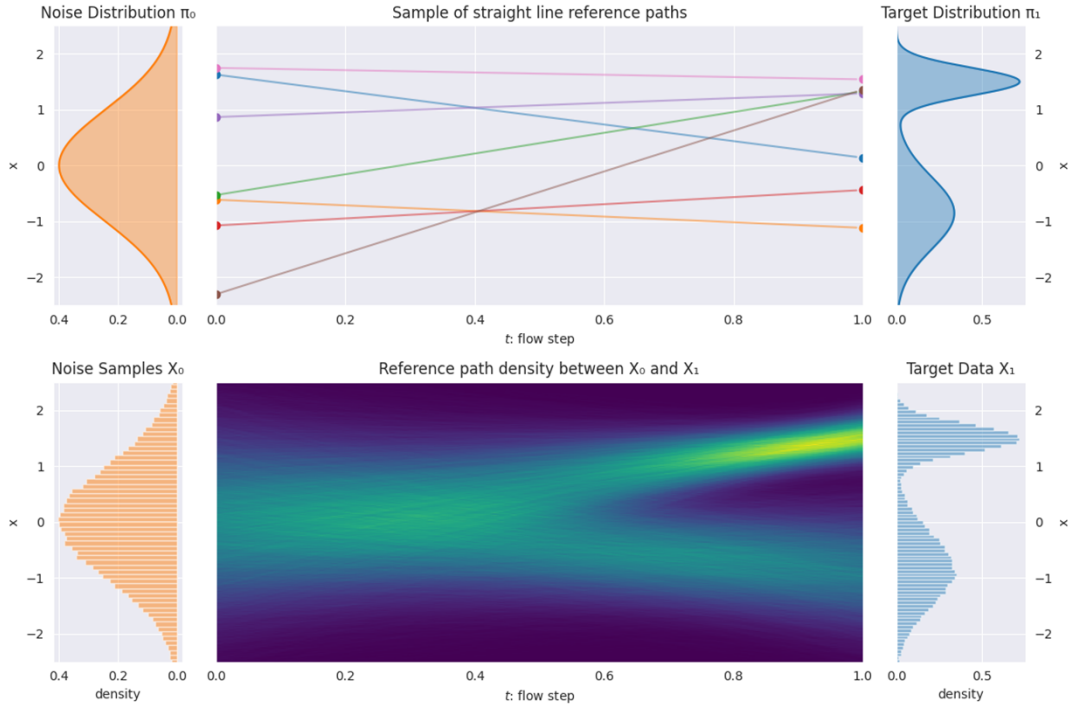
$$v_t(x_t|x_1) = \frac{d}{dt} x_t = -x_0 + x_1$$

this is obtained by simple differentiation of the field $\Phi_t(x_0)$ expression

- And the loss $\mathcal{L}_{CFM}$ becomes

$$\mathcal{L}_{CFM} = E_{t\sim U[0,1],\, x_1\sim p_1(x),\, x_t\sim p_t(x_t|x_1)} \left[\left\| v_t^\theta(x_t) - (x_1 - x_0) \right\|^2 \right]$$

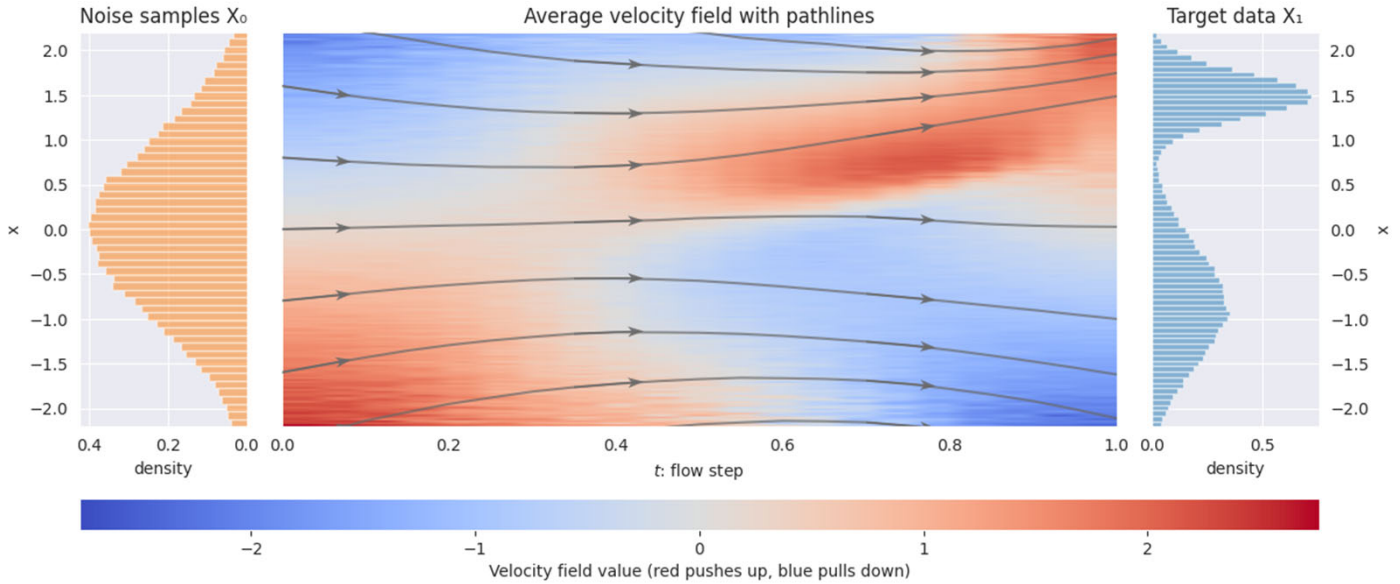
Training: Building the conditional probability path and the conditional vector field



- Top: sample straight line reference paths
- Bottom: reference paths distribution generated by sampling a large nb of straight line reference paths

Credits: https://peterroelants.github.io/posts/flow_matching_intro/

Training: Building the conditional probability path and the conditional vector field



Average flow field computed by sampling a large nb of reference paths and computing the average velocity for fixed bins over the flow field

Credits: https://peterroelants.github.io/posts/flow_matching_intro/

Training algorithm

- This leads to the following training algorithm

Sample $x_0 \sim p_0$, $x_1 \sim p_1$, $t \sim U[0,1]$
Compute $x_t = \Phi_t(x_0) = (1-t)x_0 + tx_1$
Compute the loss $\mathcal{L} = \left\| v_t^\theta(x_t) - (x_1 - x_0) \right\|^2$
Gradient descent $\theta = \theta - \epsilon \nabla_\theta \mathcal{L}$

Flow matching
A few explanations

The marginalization trick used in flow matching

- Explains why the conditional vector fields allow us to build the marginal field

Theorem 2

For $z \in \mathbb{R}^d$, let $v_t(.|z)$ denote a conditional vector field, whose corresponding ODE defines a conditional probability path $p_{t|z}(.|z)$:

$$x_0 \sim p_0, \frac{dx_t}{dt} = v_t(x_t|z) \implies x_t \sim p_t(x|z), t \in [0,1]$$

Then the marginal vector field $v_t(x)$ defined as:

$$v_t(x) = \int v_t(x|z) \frac{p_t(x|z)p_1(z)}{p_t(x)} dz$$

Follows the marginal probability path:

$$x_0 \sim p_0, \frac{dx_t}{dt} = v_t(x_t) \implies x_t \sim p_t \; t \in [0,1]$$

Continuity equation

Theorem 3

Let a flow model defined by its vector field v_t and initial condition $x_0 \sim p_0$, then $x_t \sim p_t, t \in [0,1]$, if and only if the continuity equation is verified

$$\frac{\partial p_t(x)}{\partial t} = -\nabla.(p_t v_t)(x), \forall x \in \mathbb{R}^d, t \in [0,1]$$

Where $\nabla.$ Is the divergence operator:

$$\nabla. v(\mathrm{x}) = \sum_{i=1}^{d} \frac{\partial v_t(x)}{\partial x_i}, \text{ and } x = (x_1, \ldots, x_d)^T$$

- The continuity equation expresses the conservation of quantities like physical mass, energy, probability mass.
  - Here, $\frac{\partial p_t(x)}{\partial t}$ describes how much p_t changes over time, and the negative divergence $-\nabla.(p_t v_t)$ expresses the total inflow of probability mass.
  - Probabiliy mass being conserved, the two quantities must be equal

Using the continuity equation to demonstrate the marginalization trick

$$
\begin{aligned}
\frac{\partial}{\partial t} p_t(x) &\overset{(i)}{=} \frac{\partial}{\partial t} \int p_t(x \mid z)\, p_1(z)\, dz \\
&= \int \frac{\partial}{\partial t} p_t(x \mid z)\, p_1(z)\, dz \\
&\overset{(ii)}{=} \int -\nabla \cdot (p_t(x \mid z)\, v_t(x \mid z))\, p_1(z)\, dz \\
&\overset{(iii)}{=} -\nabla \cdot \left(\int p_t(x \mid z)\, v_t(x \mid z)\, p_1(z)\, dz \right) \\
&\overset{(iv)}{=} -\nabla \cdot \left(p_t(x) \int v_t(x \mid z) \frac{p_t(x \mid z)\, p_1(z)}{p_t(x)}\, dz \right) \\
&\overset{(v)}{=} -\nabla \cdot (p_t(x)\, v_t(x))\,.
\end{aligned}
$$

Note: (ii) uses the continuity equation for the conditional probability path $p_t(x|z)$

By theorem 3 this demonstrates theorem 2

Training

Theorem 4

The marginal flow loss $\mathcal{L}_{FM}$ equals the conditional flow matching loss $\mathcal{L}_{CFM}$ up to a constant, hence their gradients coincide

$$\nabla_\theta \mathcal{L}_{FM} = \nabla_\theta \mathcal{L}_{CFM}$$

and minimizing $\mathcal{L}_{CFM}$ is equivalent to miminizing $\mathcal{L}_{FM}$

Guided flow matching

Data coupling

- For training flow matching, we need to sample from $p_{0,1}(x_0, x_1)$
- In the Gaussian probability path example, for the conditional probability path/ velocity field we have been sampling independently the source and the target.
  - We then assumed $p_{0,1}(x_0, x_1) = p_0(x_0)p_1(x_1)$
- Other coupling may be used leveraging a dependency between x_0 and x_1
  - e.g. $p_{0,1}(x_0, x_1) = p_{0|1}(x_0|x_1)p_1(x_1)$
  - This means that we first sample $x_1$and then conditioned on x_1, we sample x_0
- Example: recovering a full image from a partial/ noisy image
  - This is a ill-defined problem: given a partial/ noisy image, there are several possible reconstructions
  - How to build a training set?
    - Sample target images x_1
    - Apply a transformation (inpainting, noise, super-resolution, etc) to get the conditioned x_0
  - Train as in the flow matching algorithm describe above

Guidance

- Up to now, we have been considering generating target data x_1 from noise x_0 or incomplete versions of x_1
- Consider the more general setting of generating x_1 (e.g. an image) conditioned on context y (e.g. a class indicator or a vector embeding of a text query)
- The objective then becomes generating/ sampling from $p_1(x|y)$ and not from $p_1(x)$
  - This is called guided generation
  - This amounts at learning a guided vector field $v_t(x|y)$
  - Note: the term guided is used to distinguish the conditioning on the context from the conditional vector field $v_t(x|x_1)$ used in flow matching

Guidance
Guided conditional flow matching objective

- Let y be a conditioning vector – e.g. text prompt embedding –
  - Suppose that we sample from $p_1(x|y)$ instead of $p_1(x)$ in the unconditional flow matching model.
  - Then the corresponding CFM objective is simply

$$E_{x_1 \sim p_1(x|y),\, x \sim p_t(.|x_1)} \left[\left\| v_t^\theta(x|y) - v_t(x|x_1) \right\|^2 \right]$$

  - Note: $v_t(x|x_1)$ does not depend on y. The dependence comes through the sampling process: $x_1 \sim p_1(x|y)$, then $x \sim p_t(.\,|x_1)$
- Taking the expectation over all the choices of y and on the distribution of t we get the guided conditional flow matching objective

$$\mathcal{L}_{CFM-guided}(\theta) = E_{t \sim U[0,1],\, (x_1,y) \sim p_1(x,y),\, x \sim p_t(x|x_1)} \left[\left\| v_t^\theta(x|y) - v_t(x|x_1) \right\|^2 \right]$$

  - Note: the difference with the unconditional loss is that sampling is performed on the joint distribution $p_1(x, y)$ instead of $p_1(x)$ in the unconditional case

Classifier-free guidance

- This principled procedure is not the one used in practice
- A popular procedure for training a guided generator is classifier-free guidance
- It replaces $v_t^\theta(x|y)$ with $\tilde{v}_t^\theta(x|y)$ defined as

$$\tilde{v}_t^\theta(x|y) = (1-w)v_t^\theta(x) + wv_t^\theta(x|y)$$

  - This is a linear combination of the unguided field $v_t^\theta(x)$ and the guide term $v_t^\theta(x|y)$
  - Both velocity fields are implemented through the same network θ
    - The absence of label in $v_t^\theta(x)$ is implemented as new label place-holder vector $\emptyset$
  - $w \geq 0$ is the guidance scale
  - Note: the term « classifier-free » comes from the score diffusion model for which the first version of guidance was derived using a classifier $p(y|x)$
    - Here one only makes use of densities $p(x|y)$

Classifier-free guidance

- The corresponding loss function becomes

$$\mathcal{L}_{CFM-CFG}(\theta) = E_{t\sim U[0,1],\,(x_1,y)\sim p_1(x,y;\eta),\,x\sim p_t(x|x_1)}\left[\left\|v_t^{\theta}(x|y) - v_t(x|x_1)\right\|^2\right]$$

  - $p_1(x, y; \eta)$ means y is replaced by $\emptyset$ with a given probability η

Classifier-free guidance
Training algorithm for Gaussian probability path

- We consider the Gaussian probability path defined as

$$p_t(x|x_1) = p_{t|1}(x|x_1) = \mathcal{N}(x|\alpha_t x_1, \beta_t^2 I)$$

- Classifier-free guidance algorithm
  - Iterate for each mini batch
    - Sample $(x_1, y) \sim p_1(x_1, y)$
    - Sample $t \sim \text{Unif}[0,1)$
    - Sample $\epsilon \sim \mathcal{N}(0, I_d)$
    - With probability η set set $y \leftarrow \emptyset$
    - Compute loss
      - $\mathcal{L}(\theta) = \left\| v_t^\theta(x|y) - \partial_t \alpha_t \epsilon + \partial_t \beta_t x_1 \right\|^2$
    - Update gradients

  - Note, in the derivation for the non guided case, we used $\alpha_t = t, \beta_t = 1 - t$, so that the loss becomes
    - $\mathcal{L}(\theta) = \left\| v_t^\theta(x|y) - \epsilon - x_1 \right\|^2$

Classifier-free guidance - examples

Figure 1: Classifier-free guidance on the malamute class for a 64x64 ImageNet diffusion model. Left to right: increasing amounts of classifier-free guidance, starting from non-guided samples on the left.

Fig. Ho & Salimans, 2022, https://arxiv.org/abs/2207.12598

Fig. Holderriet & Erives 2025, https://arxiv.org/abs/2506.02070

References on flow matching

- Tutorial and great site at MIT, the course is largely inspired from this presentation
  - https://diffusion.csail.mit.edu/
  - Course notes https://arxiv.org/abs/2506.02070
- A simple visual introduction to flow matching
  - https://peterroelants.github.io/posts/flow_matching_intro/
- An easy to follow introduction
  - https://nilsschaetti.ch/2025/02/02/flow-matching-comprendre-les-derniers-modeles-de-generation-dimages/#:~:text=Qu'est%2Dce%20que%20le,texte%20ou%20m%C3%AAme%20des%20vid%C3%A9os.
- Reference papers
  - Lipman, Y., Havasi, M., Holderrieth, P., Shaul, N., Le, M., Karrer, B., Chen, R. T. Q., Lopez-Paz, D., Ben-Hamu, H., & Gat, I. (2024). Flow Matching Guide and Code. http://arxiv.org/abs/2412.06264 (lots of details but more tricky than the course note above)
  - Lipman, Y., Chen, R. T. Q., Ben-Hamu, H., Nickel, M., Le, M., & Ai, M. (2023). Flow Matching for Generative Modeling ICLR.

Diffusion models

Diffusion models

- Diffusion models emerged in 2019, gained momentum in 2021
- As in 2025, 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, protein structure
- Several approaches including
  - Discrete time models
    - Denoising Diffusion Probabilistic Models (DDPMs)
    - Score based Generative Models (SGM)
  - Time continuous models
    - Stochastic differential equations

Diffusion models

- We introduce diffusion models through stochastic differential equations – this provides a unifying – synthetic view of these models
- A stochastic differential equation extends the deterministic dynamics of ODE by adding a stochastic component
- It is usually denoted as

$$dX_t = v_t(X_t)dt + \sigma_t dW_t$$

$$\mathrm{X}_0 = x_0 \text{ (initial condition)}$$

Here

$X_t \in \mathbb{R}^d$ is a random process

v_t is the drift (the vector field)

$\sigma_t \in \mathbb{R}^+$ is a diffusion coefficient

$W_t \in \mathbb{R}^d$ is a Brownian motion also called Wiener process

The solution X of a SDE is called a stochastic trajectory

$$\mathrm{X}: [0,1] \to \mathbb{R}^d, t \to X_t$$

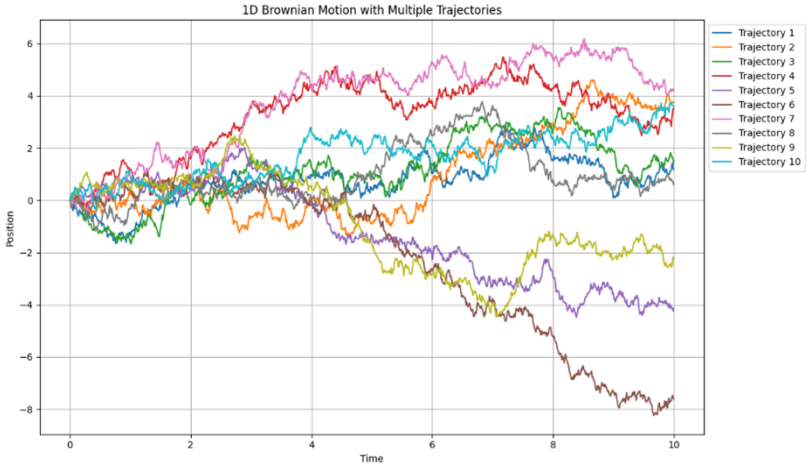
Diffusion models
SDE solution existence and uniqueness

Theorem 1

If $v: [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ is continuously differentiable with a bounded derivative, and $\sigma_t$is continuous, then the SDE, $dX_t = v_t(X_t)dt + \sigma dW_t$, $\mathrm{X}_0 = x_0$ (initial condition) has a solution given by a unique stochastic process

Diffusion models
Brownian motion

- A brownian motion $W = (W_t)$ for $t \in [0,1]$ is a stochastic process and satisfies
  - $W_0 = 0$
  - Independent increments: $W_{t_i} - W_{t_{i-1}}$ are independent variable of the past for any $0 \leq t_0 < t_1 < \cdots < t_n = 1$
  - Gaussian increments: $W_t - W_s \sim \mathcal{N}(0, (t-s)I_d), 0 \leq s < t$
- They can be simulated through
  - $W_{t+h} = W_t + \sqrt{h}\epsilon_t$ for $\epsilon_t \sim \mathcal{N}(0, I_d), \mathrm{h} > 0$ is the step size



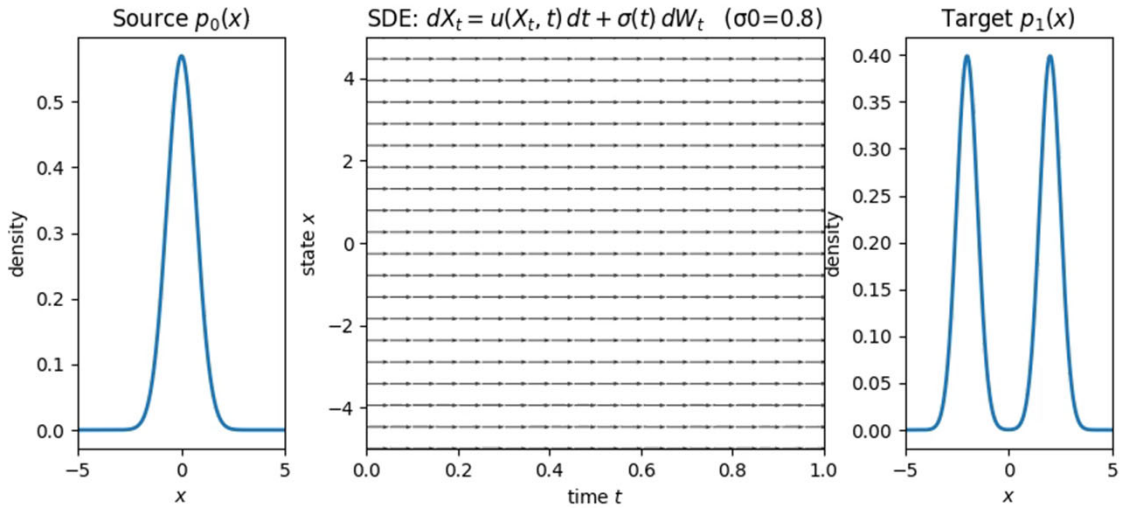
Diffusion models
Simulating a SDE

- The simplest integration scheme for a SDE is the Euler-Maruyama method
- This is the SDE analogue of forward Euler for SDEs

Initialise $X_0 = x_0$

$$X_{t+h} = X_t + hv_t(X_t) + \sqrt{h}\sigma_t\epsilon_t \; \epsilon_t \sim \mathcal{N}(0, I_d)$$

Where h is the step size



Diffusion models

- Suppose that we have trained a vector field v_t^θ, we can then build a generative model with an SDE as we did for an ODE

Sampling from a diffusion model

Set $t = 0$, set step size $h = \frac{1}{n}$
Sample $X_0 \sim p_0$ e.g. a Gaussian
For $i = 1, \ldots, n-1$ do
    draw a sample $\epsilon_t \sim \mathcal{N}(0, I_d)$
    $X_{t+h} = X_t + h v_t^\theta(X_t) + \sqrt{h}\sigma_t \epsilon_t$
    $t = t + h$
Return X_1

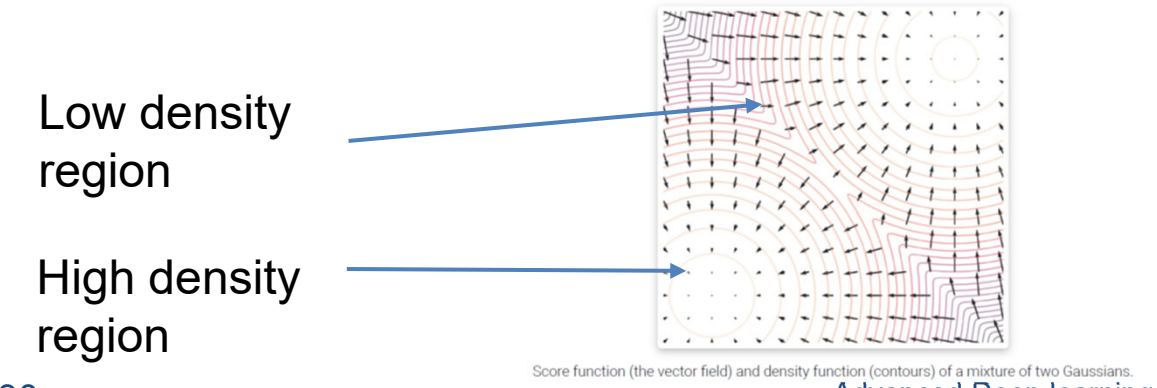
A diffusion model will be defined by
    A parametric vector field $v_t^\theta [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$
    A diffusion coefficient $\sigma_t : [0,1] \to [0, +\infty), t \to \sigma_t$

Diffusion models
Score function

- Together with a vector field, diffusion models learn, a score function
- The score function of a data distribution $p(x), x \in R^d$ is:

$$s(X) = \nabla_X \log \mathrm{p(X)} \in R^d$$

  - 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 $p(.)$?
      - Sample x_0 from a prior (e.g. Gaussian) distribution $p_0(x)$ in R^d and iterate $X_{i+1} = X_i + \nabla_X \log p(X_i)$



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

Diffusion models
Score function – example Langevin dynamics

- Langevin dynamics
  - The Langevin dynamics for sampling from a known distribution $p(X)$ is an iterative procedure:

$$X_{i+1} = X_i + \epsilon \nabla_X \log p(X_i) + \sqrt{2\alpha}\epsilon_i$$

    - $i = 0, \ldots, K$, with $\epsilon_i \sim \mathcal{N}(0, I)$, α is a small constant
    - When $\alpha \to 0$ 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)

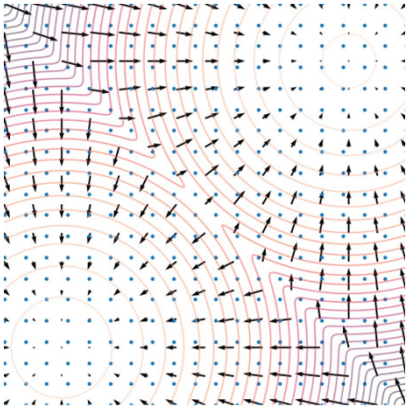


Fig. Song 2022
Langevin dynamics for sampling from a mixture of 2 gaussians, arrows indicate the score vector values, the animated Gif shows the convergence of the dynamics towards the target distribution

Diffusion models

- Lets define the (marginal) score function of $p_t$as $\nabla \log p_t(\mathrm{x})$, we can define a SDE that follows the probability path p_t

Theorem 2

Let $v_t(x)$a marginal vector field, then the following SDE follows the same probability path $p_t$as the ODE, with arbitrary diffusion coefficients σ_t :

$$X_0 \sim p_0, dX_t = [v_t(X_t) + \frac{\sigma_t^2}{2} \nabla \log p_t(X_t)]dt + \sigma_t dW_t$$

$$\Longrightarrow X_t \sim p_t, 0 \leq \mathrm{t} \leq 1$$

In particular $X_1 \sim p_1$ the target distribution

- Note
  - The same property holds for a conditional vector field $v_t(x|z)$ and the conditional probability path $p_t(x|z)$
  - This will allow us to define a training algorithm as we did for the flow model with the ODE, by training to approximate a conditional score $\nabla p_t(x|z)$

Diffusion models
Training algorithm

- Score matching loss
  - $\mathcal{L}_{SM}(\theta) = E_{t\sim Unif[0,1],\, z\sim p_1, \epsilon\sim\mathcal{N}(0,I_d)}\left[\left\|s_t^\theta(x) - \nabla \log p_t(x)\right\|^2\right]$
- Conditional score matching loss
  - $\mathcal{L}_{CSM}(\theta) = E_{t\sim Unif[0,1],\, z\sim p_1, \epsilon\sim\mathcal{N}(0,I_d)}\left[\left\|s_t^\theta(x) - \nabla \log p_t(x|z)\right\|^2\right]$

Theorem 3

The score matching loss equals the conditional score matching loss up to a constant:

$$\mathcal{L}_{SM}(\theta) = \mathcal{L}_{CSM}(\theta) + C$$

Their gradient coincide:

$$\nabla_\theta \mathcal{L}_{SM}(\theta) = \nabla_\theta \mathcal{L}_{CSM}(\theta)$$

Diffusion models
Training algorithm

- We consider the following SDE
- $X_0 \sim p_0, dX_t = [v_t(X_t) + \frac{\sigma_t^2}{2} \nabla \log p_t(X_t)]dt + \sigma_t dW_t$
- Training
  - Training will amount to train parametric approximations v_t^θ of the vector field v_t and s_t^θ of the score function $\nabla \log p_t$
  - Learning v_t^θ will proceed as for Flow matching
  - We indicate how to learn s_t^θ
- Inference
  - After training we can choose a diffusion coefficient σ_t and simulate the following SDE:

$$X_0 \sim p_0, dX_t = [v_t^\theta(X_t) + \frac{\sigma_t^2}{2} s_t^\theta(X_t)]dt + \sigma_t dW_t$$

Diffusion models
Score matching training

Score matching training

Given: a dataset of samples $z \sim p_1$, a score network s_t^θ

Repeat

Sample $z \sim p_1$

Sample $t \sim Uniform[0,1]$

Sample $x \sim p(.|z)$

Compute loss

$$\mathcal{L}(\theta) = \left\| s_t^\theta(x) - \nabla \log p_t(x|z) \right\|^2$$

Update the model parameters

$$\theta \leftarrow \theta - \nabla \mathcal{L}(\theta)$$

Diffusion models
Score matching training for Gaussian probablity path

- The Gaussian probability path allows us to derive an algebraic expression for $\nabla \log p_t(x|z)$
- Recall: Gaussian probability path:

  - Let α_t, β_t be two continuous differentiable monotonic functions such that $\alpha_0 = \beta_1 = 0$ and $\alpha_1 = \beta_0 = 1$

The Gaussian conditional path

$$p_t(.|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I)$$

Fulfills

$$p_0(.|z) = \mathcal{N}(\alpha_0 z, \beta_0^2 I) = \mathcal{N}(0, I) \text{ and } p_1(.|z) = \mathcal{N}(\alpha_1 z, \beta_1^2 I) = \delta_z$$

Sampling from the marginal path consists in

$$x_1 \sim p_1, \epsilon \sim \mathcal{N}(0, I) \Rightarrow x = \alpha_t x_1 + \beta_t \epsilon \sim p_t$$

Diffusion models
Score matching training for Gaussian probablity path

- For the Gaussian probability path
  - $\nabla \log p_t(x|z) = -\frac{x-\alpha_t z}{\beta_t^2}$
- Plugging in this expression gives the conditional score matching loss:

$$\mathcal{L}_{CSM}(\theta) = E_{t\sim Unif[0,1],\, z\sim p_1, \epsilon\sim\mathcal{N}(0,I_d)}\left[\left\| s_t^\theta(\alpha_t z + \beta_t \epsilon) + \frac{\epsilon}{\beta_t}\right\|^2\right]$$

Diffusion models
Score matching training for Gaussian probablity path

Score matching training

Given:

A dataset of samples $z \sim p_1$, a score network s_t^θ

Schedulers α_t, β_t with $\alpha_0 = \beta_1 = 0, \alpha_1 = \beta_0 = 1$

Repeat

Sample $z \sim p_1$

Sample $t \sim Uniform[0,1]$

Sample $\epsilon \sim \mathcal{N}(0, I_d)$

Set $x_t = \alpha_t z + \beta_t \epsilon$

Compute loss

$$\mathcal{L}(\theta) = \left\| s_t^\theta(x) + \frac{\epsilon}{\beta_t} \right\|^2$$

Update the model parameters

$$\theta \leftarrow \theta - \nabla \mathcal{L}(\theta)$$

Diffusion models
Denoising diffusion model

- Denoising diffusion models, one of the early, and popular diffusion model corresponds to a diffusion model with a Gaussian probability path $p_t(.|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I)$
- The initial formulation did not use SDEs but constructed Markov chains in discrete time
- For the Gaussian probability path, the vector field and the score function can converted one to the other post training
  - It is not necessary to train both, train for one of them only, and the other could be obtained post training
  - In particular we can choose if want to use flow matching or score matching to train it
-

Denoising Diffusion Probabilistic Models
Original presentation

Denoising Diffusion models

- Denoising 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

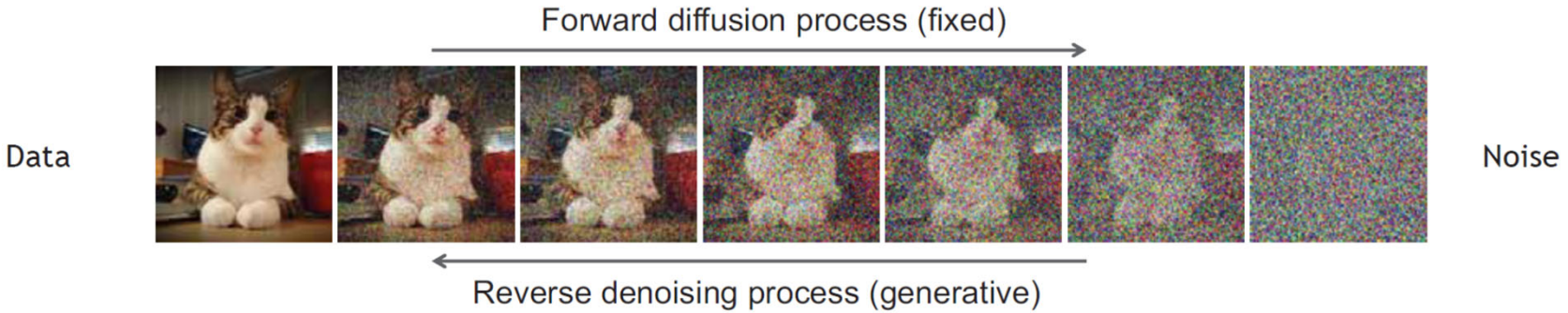


Fig. Kreis et al. 2022

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

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

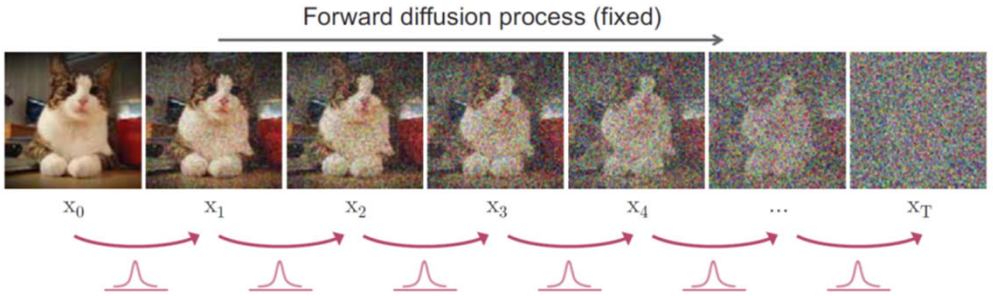


Fig. Kreis et al. 2022

- The forward process corresponds to a probability path $p_t(.|z)$ with z sampled from the target distribution

Denoising Diffusion Probabilistic Models
Forward (diffusion) process

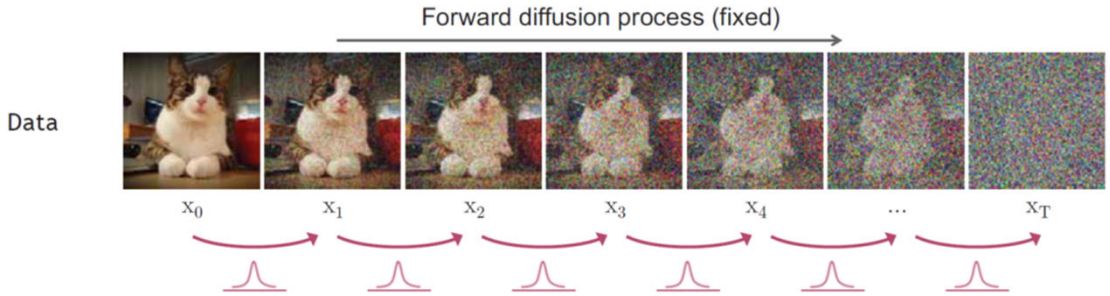


Fig. Kreis et al. 2022

- A typical design for the kernel is a gaussian perturbation $q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{\alpha_t}x_{t-1}; (1-\alpha_t)I\right) \forall t \in \{1, \dots, T\}$
- Using the reparametrization trick, one can write:

$$x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}z, \text{ with } z \sim N(O, I)$$

  - I is the identity matrix, with the same size as image x_0, $\alpha_t \in (0,1)$ is a variance parameter hand fixed or learned, we consider it hand fixed here.
    - α_t is chosen so that $\alpha_t > \cdots > \alpha_T$, e. g. $T = 2000, \alpha_1 = 1 - 10^{-4}, \alpha_T = 1 - 10^{-2}$ with a linear increase

Denoising Diffusion Probabilistic Models
Forward (diffusion) process

- The forward diffusion process is then defined as

- $\mathbf{x_0} \sim \mathbf{q}(\mathbf{x_0})$,
- $q(x_1, \dots, x_T | x_0) = \prod_{t=1}^{T} q(x_t | x_{t-1})$,
- $\mathcal{N}(x_t; \sqrt{\alpha_t}\, x_{t-1}; (1 - \alpha_t) I)$ $\quad \forall t \in \{1, \dots, T\}$
  - $x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t}\epsilon$, with $\epsilon \sim N(O, I)$
- $\alpha_t \in [0,1]$ is a variance hyperparameter, $\alpha_t > \cdots > \alpha_T$

Denoising Diffusion Probabilistic Models
Forward process – Diffusion kernel

- Property: the forward process can be sampled at any time t in closed form
  - $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_O, (1-\bar{\alpha}_t)I)$ – this is called the **diffusion kernel**
  - with $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$
- 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 $x_t \sim q(x_t|x_0)$ (this is called ancestral sampling) and this is the main formula for the forward process

$$x_t = \sqrt{\bar{\alpha}_t}x_O + \sqrt{(1-\bar{\alpha}_t)}\epsilon, \text{ with } \epsilon \sim \mathcal{N}(0, I), \forall t \sim \mathcal{U}(\{1, \dots, T\})$$

  - -

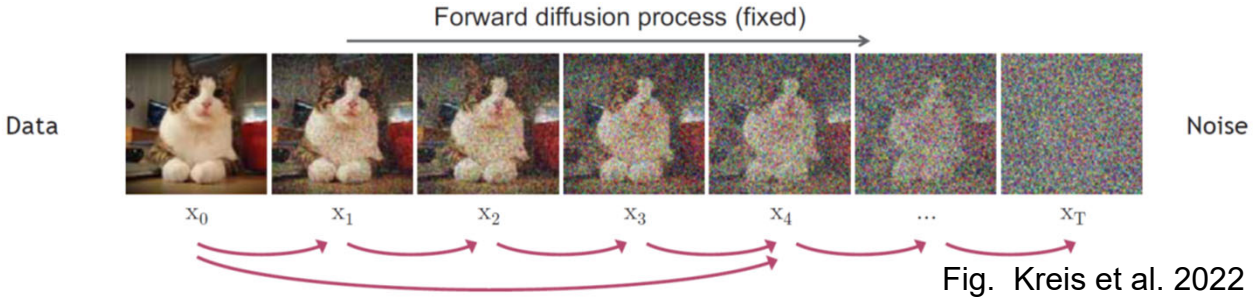


Fig. Kreis et al. 2022

Denoising Diffusion Probabilistic Models
Forward process - Illustration

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

Samples
$x_0 \sim q(x_0)$

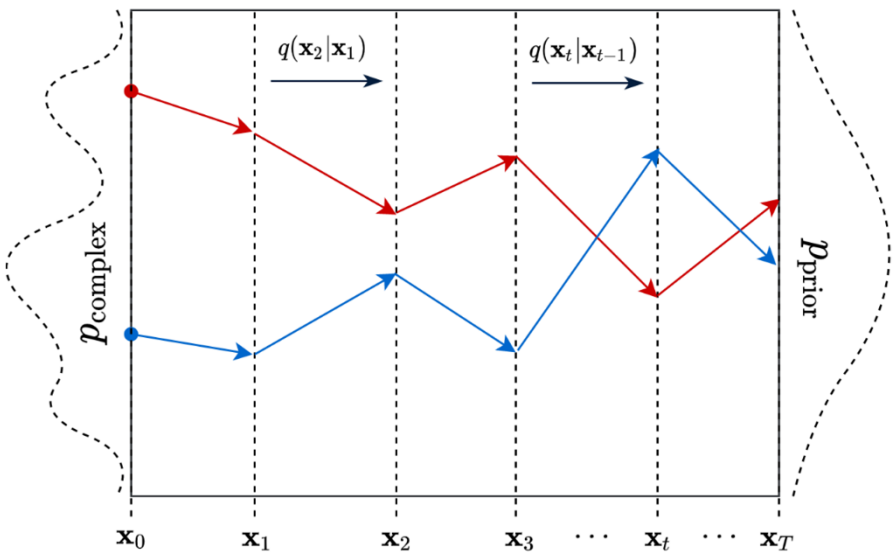


Fig. Ayan Das 2021

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

Denoising Diffusion Probabilistic Models
Reverse denoising process

- The reverse distribution is
  - $q(x_0, \dots, x_T) = q(x_{0:T}) = q(x_T) \prod_{t=1}^{T} q(x_{t-1}|x_t)$
  - The $q(x_{t-1}|x_t)$ are complex multimodal distributions, they are approximated as normal distributions $p_\theta(x_{t-1}|x_t)$
- The reverse factorization is then
  - $p_\theta(x_0, \dots, x_T) = p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$
  - We can then generate a data sample x_0 by first sampling a noise vector from a prior distribution $x_T \sim p(x_T)$ 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 $p_\theta(x_0|x_1)$

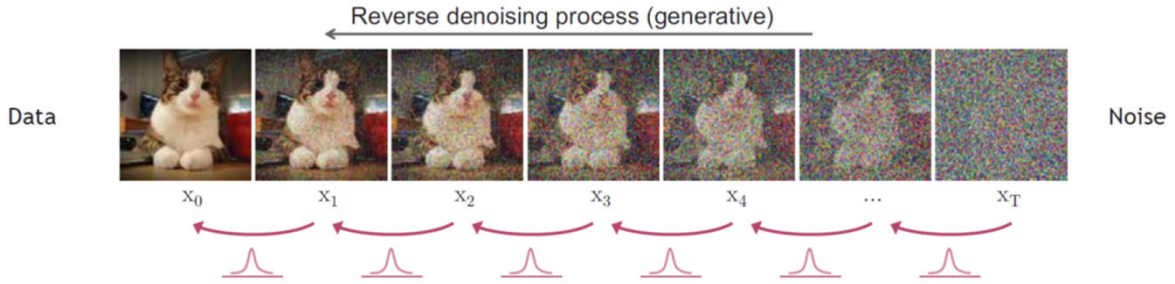


Fig. Kreis et al. 2022

Denoising Diffusion Probabilistic Models
Reverse denoising process

- 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 can be learned as $\sigma_t(x_t, t)$, but in (Ho et al. 2020) it is simply set to $1 - \alpha_t$

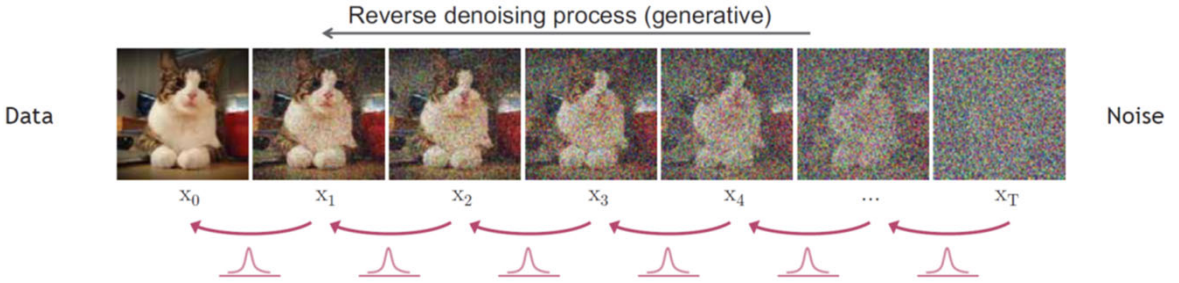


Fig. Kreis et al. 2022

Denoising Diffusion Probabilistic Models
Training and sampling algorithms

- Training algorithm

Repeat

Sample

$x_0 \sim q(x_0)$

$t \sim \text{Uniform}\ [1, T]$

Draw a sample

$$x_t = \sqrt{\bar{\alpha}_t} x_O + \sqrt{(1 - \bar{\alpha}_t)}\epsilon, \text{ with } \boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$$

Take gradient descent on

$$\nabla_\theta \|\epsilon - \epsilon_\theta(x_t, t)\|^2$$

Until convergence

Denoising Diffusion Probabilistic Models
Training and sampling algorithms

- Inference (generation) take the following simple forms

$x_T \sim N(O, I)$
for $t = T$ *to* 1 *do*
  $z \sim N(0, I)$
  Update according to

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma(t)z \text{ with } z \sim N(O, I)$$

end for
return x_0

Fig. Ho et al 2020

Denoising Diffusion Probabilistic Models Implementation

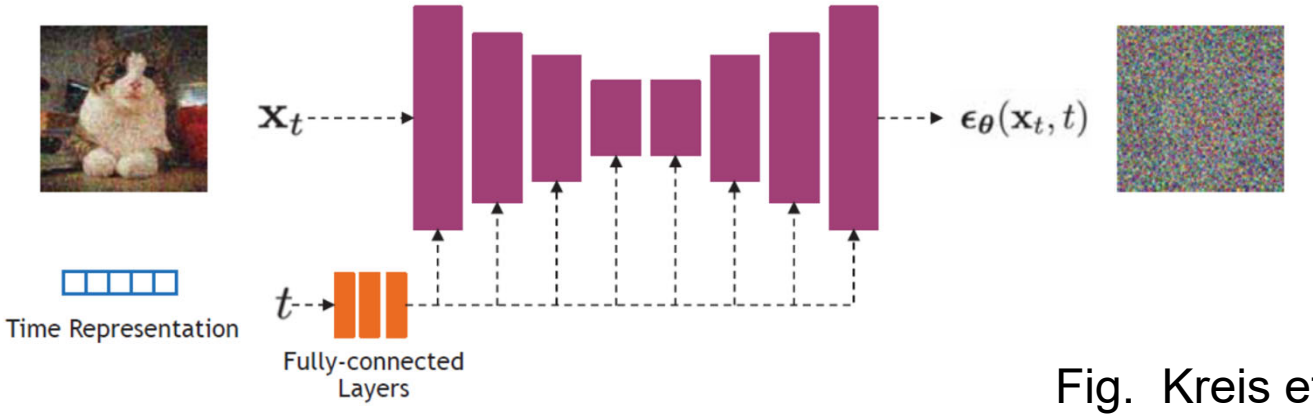


Fig. Kreis et al. 2022

- $\epsilon_\theta(x_t, t)$ 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
  - $T = 1000, \beta_1 = 10^{-4}, \beta_T = 0.02, \beta_t$ increases 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 dimensional 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)

Diffusion models

- References (in Red – recommended references)
  - Tutorial / survey papers
    - S. Chan, Tutorial on diffusion models for imaging and vision, https://arxiv.org/pdf/2403.18103 <<<< a gentle introduction illustrated with examples
    - 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
  - 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 2022 tutorial, https://cvpr2022-tutorial-diffusion-models.github.io/
    - Video by Sohroush Mehraban: https://www.youtube.com/watch?v=r4V0vLhYZIQ&list=PLHYTrZz5TkDmORMaTUg59lx-jvaB0K6mF&index=11
  - Reference 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