Generative Models

Discriminative vs Generative

Understanding Machine Learning

Archetype: Naïve Bayes

probabilistic model:

$$P(Y|X_1,\cdots,X_p) = \frac{P(Y,X_1,\cdots,X_p)}{P(X_1,\cdots,X_p)} = \frac{P(Y)P(X_1,\cdots,X_p|Y)}{P(X_1,\cdots,X_p)} \propto P(Y)P(X_1,\cdots,X_p|Y)$$

Bayes' rule

constant

to be estimated

approach:

1. estimate $P(Y,\boldsymbol{X})$ → generative model (can be used to generate new samples)
2. calculate $P(Y|\boldsymbol{X})$ from $P(Y,\boldsymbol{X})$ → used for discriminative task (classification)

Independence Assumption

(naïve) assumption: conditional independence of features given target

$$P\left(X_j \middle| Y, X_1, \cdots, X_{j-1}, X_{j+1}, \cdots, X_p\right) = P\left(X_j \middle| Y\right)$$

$$\Longrightarrow P\left(Y \middle| X_1, \cdots, X_p\right) = \frac{P(Y) \prod_{j=1}^{p} P\left(X_j \middle| Y\right)}{P\left(X_1, \cdots, X_p\right)}$$

→ independent feature contributions (ignoring feature correlations)

→ robust against curse of dimensionality

Estimation of Feature Contributions

separate estimations of $P(X_j|Y)$ for each feature

requires assumption of distributions (e.g., Gaussian naïve Bayes) or non-parametric methods (kernel density estimation)

Gaussian feature likelihoods: $$P(x_{ij}|y) = \frac{1}{\sqrt{2\pi\sigma_{y,j}^2}} \exp\left(-\frac{(x_{ij}-\mu_{y,j})^2}{2\sigma_{y,j}^2}\right)$$

parameter estimation (e.g., mean and variance of Gaussians) can be done with maximum likelihood method (y known in training)

→ no Bayesian methods needed

Maximum a Posteriori Classification

$$\hat{y}_i = \underset{y}{\operatorname{argmax}} P(y) \prod_{j=1}^{p} P\left(x_{ij} | y\right)$$

despite potentially inaccurate probability estimates (due to naïve independence assumption), good identification of correct class via maximum probability

→ bad for regression tasks (if independence assumption is too naïve, i.e., features are correlated)

Generative vs Discriminative Models

generative models: predict joint probability $P(Y, \boldsymbol{X})$ (what allows to create new data samples) or directly generate new data samples

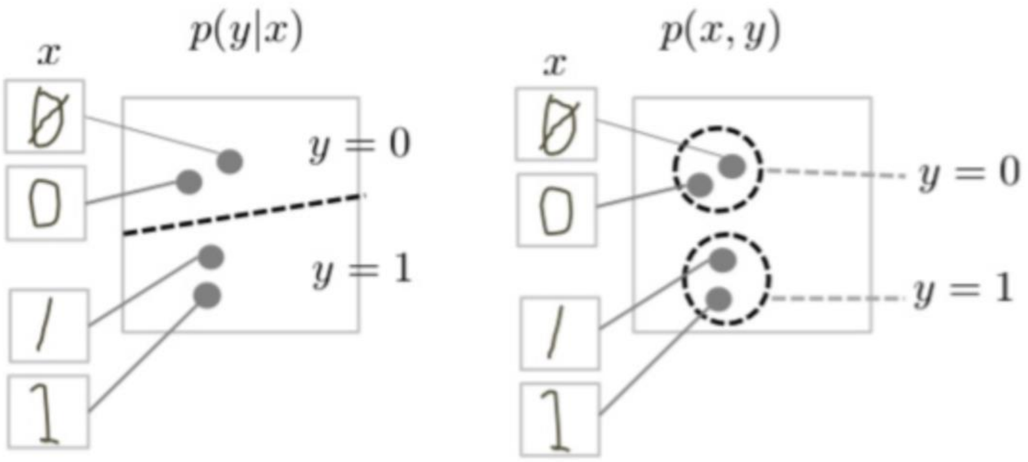
or just $P(\boldsymbol{X})$ → unsupervised (or self-supervised) learning

discriminative models: predict conditional probability (or probability distribution for regression) $P(Y|\boldsymbol{X})$ or directly output (label for classification, real value for regression)

task of generative models more difficult: model full data distribution rather than merely find patterns in inputs to distinguish outputs

discriminative model

generative model



source

Naïve Bayes and Logistic Regression

generative-discriminative pair of classification algorithms

- binary case: logit of naïve Bayes' outputs, $\log\left(\frac{P(y_i=1|\boldsymbol{x}_i)}{P(y_i=0|\boldsymbol{x}_i)}\right)$, corresponds to output of logistic regression's linear predictor
- for discrete inputs or Gaussian naïve Bayes: naïve Bayes can be reparametrized as linear classifier

for discriminative task: identical in asymptotic limit (infinite training samples) if independence assumption holds (otherwise naïve Bayes less accurate)

naïve Bayes has greater bias but lower variance than logistic regression → to be preferred for scarce training data (if bias, i.e., independence assumption, correct)

Data Generation

generative models can be used for discriminative tasks (although potentially inferior to direct discriminative methods)

but generative methods do more than discriminative ones: model full data distribution

→ allows generation of new data samples (can be images, text, video, audio, code like SQL or Python, proteins, materials, time series, structured data, …)

large (auto-regressive) language models examples of generative models

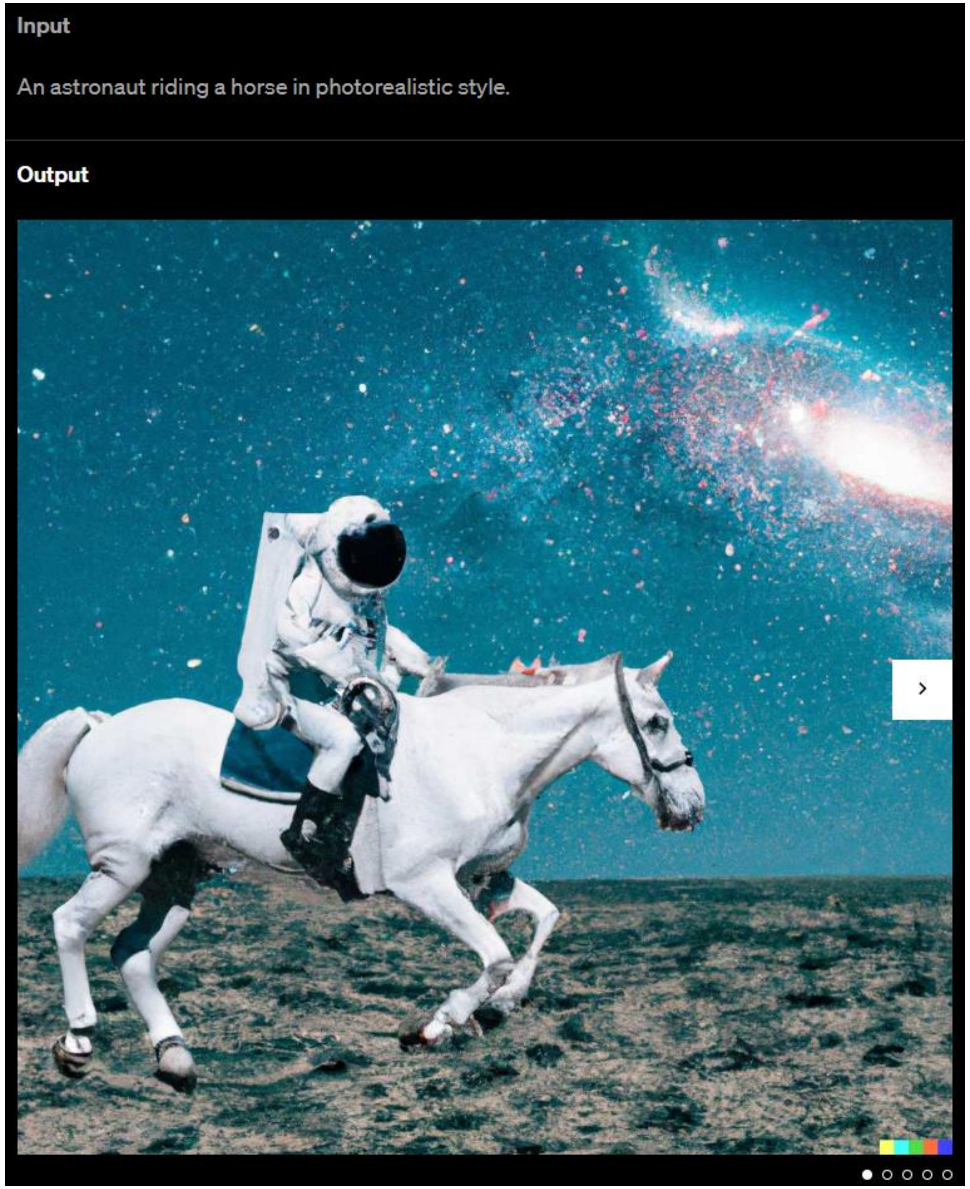
Generative AI

Depending on the application, there are currently two dominant approaches for generative AI:

- text generation: LLMs
- image synthesis: diffusion models (usually conditioned on text by transformers)

Image Synthesis

example: DALL-E 2



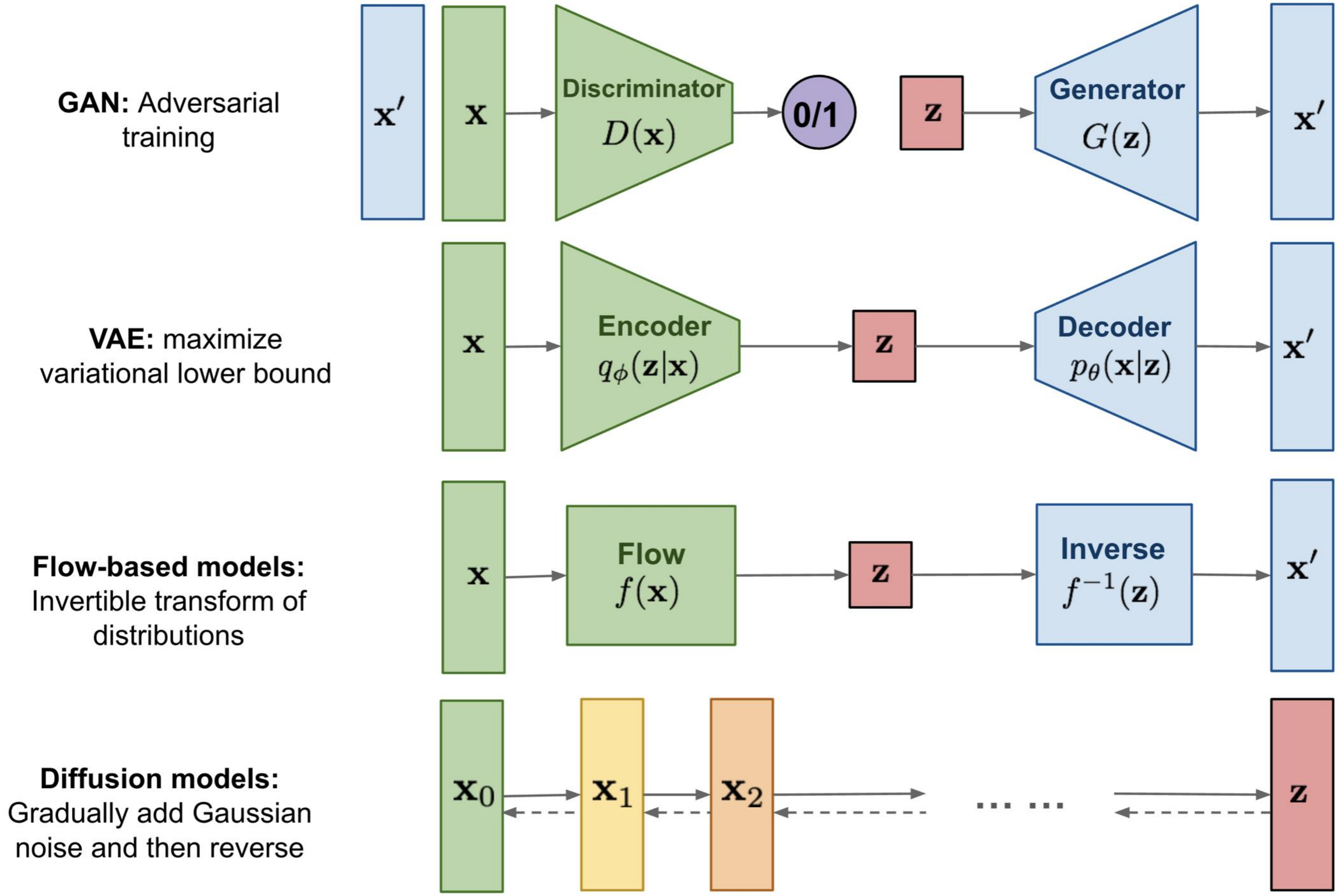
idea: generate new images as variations of training data

condition generation on text prompts: text-to-image

trade-off between diversity and fidelity

SOTA: (guided) diffusion models

Different Types of Generative Models



two neural networks playing a zero-sum game

learn variational distribution (not just replicating inputs)

more complex distributions by applying change-of-variable technique (need for specialized architecture)

chain of denoising autoencoders

source

Generative Adversarial Networks (GAN)

Indirect Training via Discriminator

two neural networks playing a zero-sum game:

- the generator network G generating new (fake) samples
- the discriminator network D trying to distinguish between real and fake samples

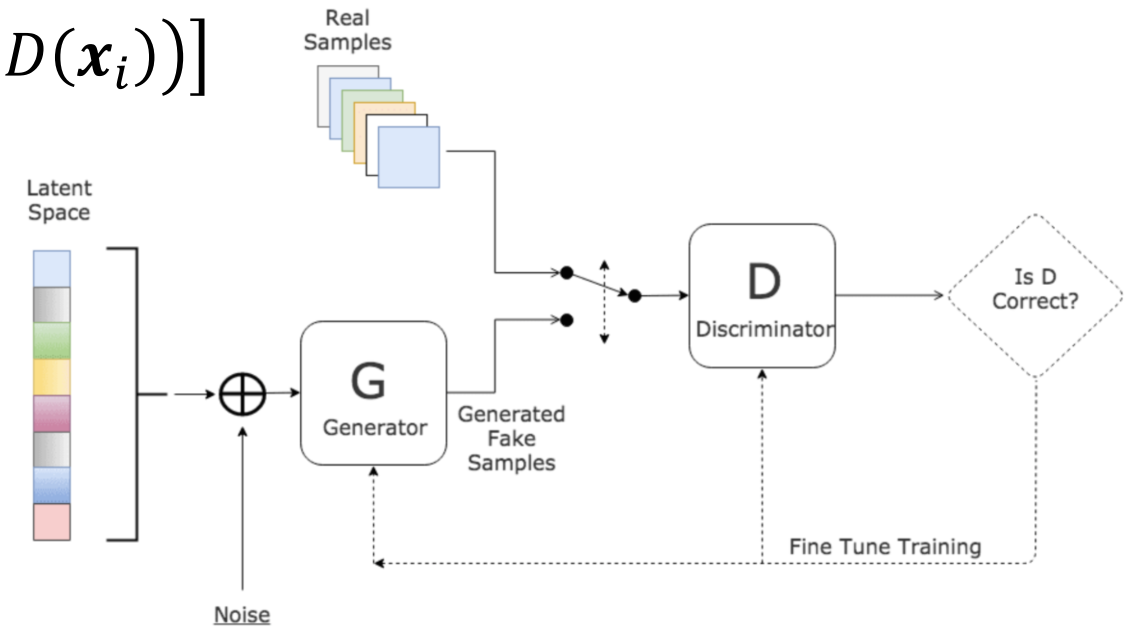
idea: G not trained directly to minimize reconstruction error of real samples, but to fool D → self-supervised approach

Formulation

common loss for generator and discriminator:

$$L(\boldsymbol{x}_i) = E_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\ln D(\boldsymbol{x}_i)] + E_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}\big[\ln\big(1 - D(\boldsymbol{x}_i)\big)\big]$$

- G trying to minimize
- D trying to maximize



generator: decomposition into latent space (parameters of generator network) and noise (sampled from, e.g., Gaussian distribution)

source

Properties

implicit generative model: do not estimate likelihood function

for optimal D, GAN loss quantifies similarity between generative data distribution p_g and real data distribution p_r by Jensen-Shannon divergence

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}\left(p||\frac{p+q}{2}\right) + \frac{1}{2} D_{KL}\left(q||\frac{p+q}{2}\right)$$

for optimal values of both G and D: $p_g = p_r$ and $D = 0.5$

issue: potentially unstable training

Variational Autoencoders (VAE)

Recap: Autoencoder

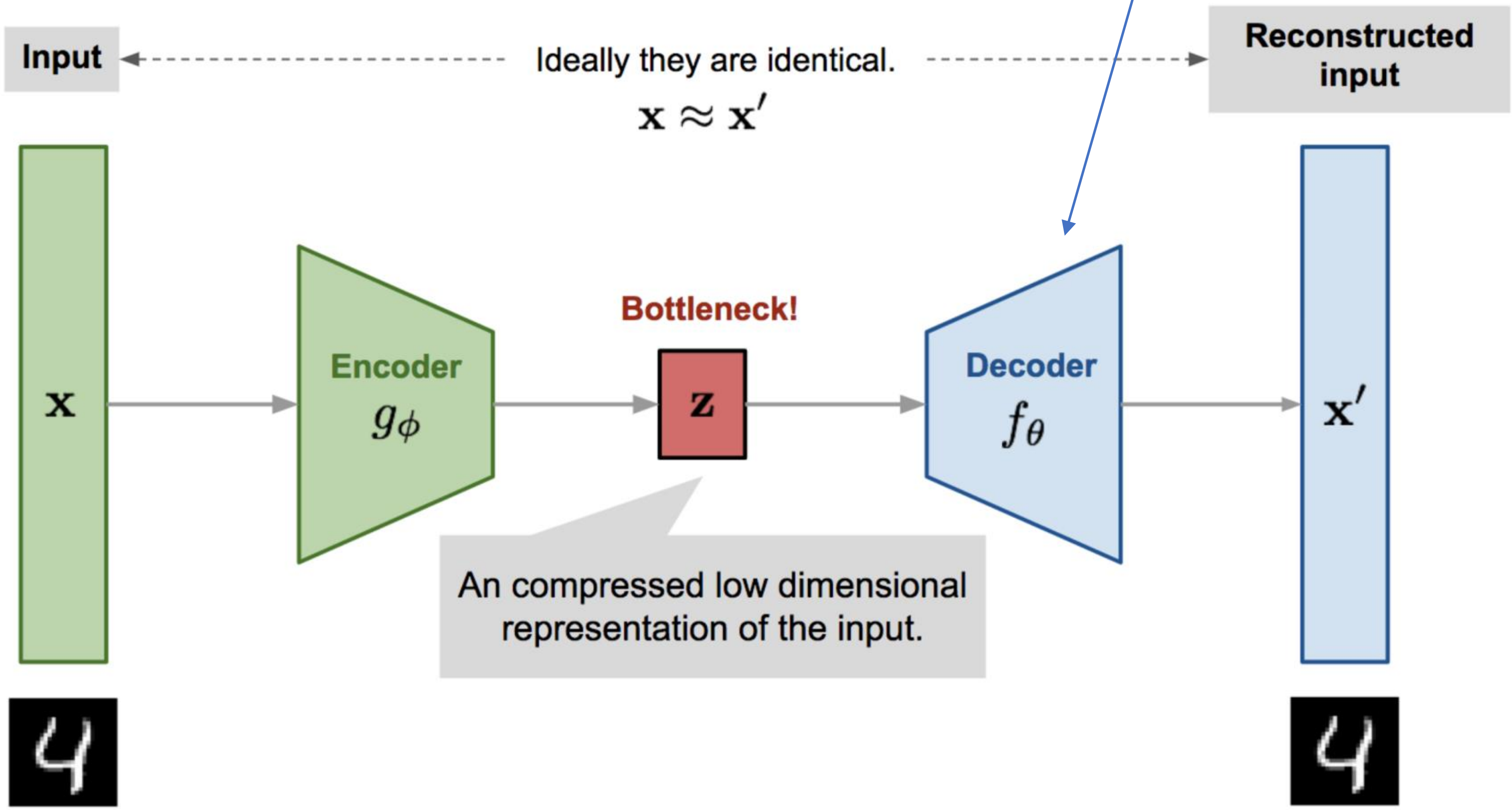
up-sampling, for example, by transposed convolutions

(deep) encoder network

(deep) decoder network

learned together by minimizing differences between original input and reconstructed input (expressed as losses)

compressed intermediate representation: dimensionality reduction



source

Autoencoder Architecture for Generative Tasks

goal: generation of variations of input data rather than compressed representation

→ learn variational distribution instead of identity function

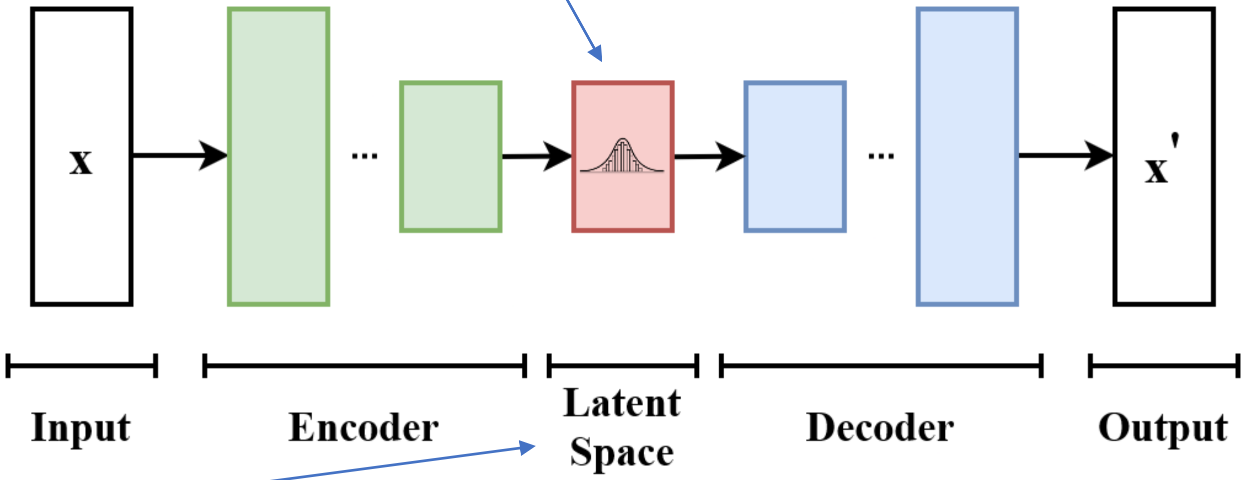
to be precise: parametrized variational distribution of latent encoding variables $\boldsymbol{z}$

prior (simple distribution, in usual VAE: Gaussian): $p_{\boldsymbol{\theta}}(\boldsymbol{z})$

posterior: $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{\int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})d\boldsymbol{z}}$

$p_{\boldsymbol{\theta}}(\boldsymbol{x})$: mixture of Gaussians

from which to sample



from wikipedia

Variational Bayesian Method

Encoder and Decoder Networks

encoder: find posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$

unfortunately, generally intractable

→ approximate by $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$

VAE: $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ expressed by neural network with weights $\boldsymbol{\phi}$

→ amortized inference:
$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ learned in training, $\boldsymbol{z}$ inferred from $\boldsymbol{x}$ in prediction (sharing variational parameters across all data points)

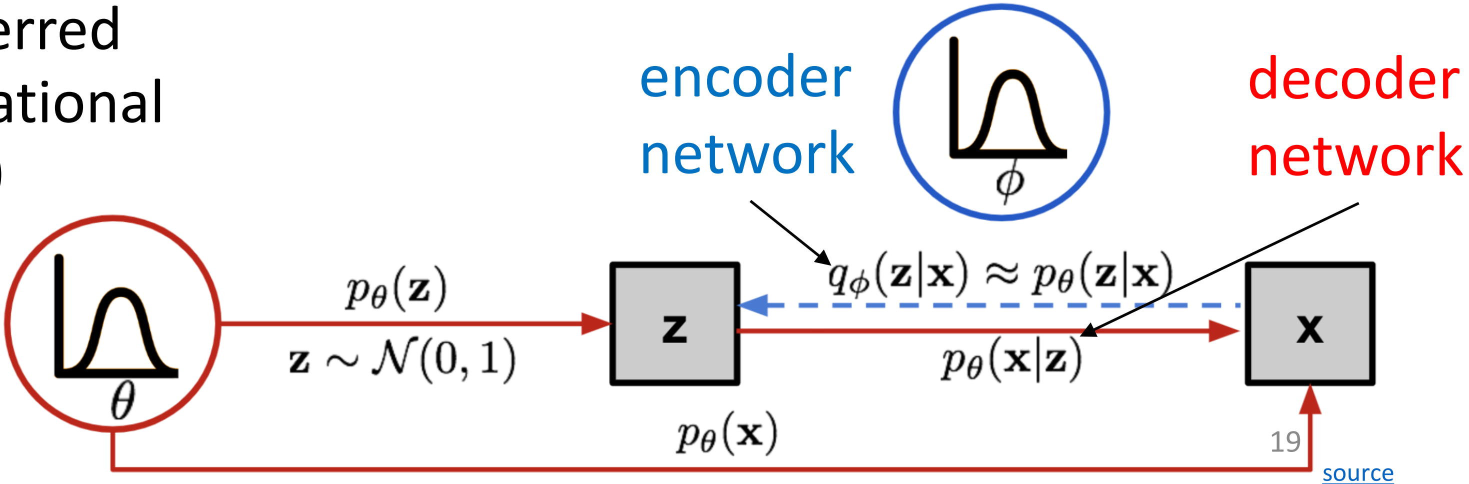
in VAE: network weights $\boldsymbol{\theta}$

decoder: generate new sample $\boldsymbol{x}_i$

1. sample $\boldsymbol{z}_i$ (from Gaussian)
2. generate $\boldsymbol{x}_i$ (similar to real data)

→ maximize: $p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z})\, p_{\boldsymbol{\theta}}(\boldsymbol{z}) dz$

(integral over $\boldsymbol{z}$ expensive → use only likely codes $\boldsymbol{z}$ given input $\boldsymbol{x}$: need for encoder)



source

VAE Loss: ELBO

VAE loss function to be minimized according to network weights:

$$L(\boldsymbol{x}_i; \boldsymbol{\theta}, \boldsymbol{\phi}) = -\ln p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) + D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i) || p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_i)\right)$$

maximize likelihood of observed data (minimize reconstruction error)

and

minimize difference of approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)$ to exact posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_i)$

can be interpreted as regularizer

corresponds to maximizing evidence lower bound (ELBO), i.e., maximizing lower bound of probability to generate real data sample:

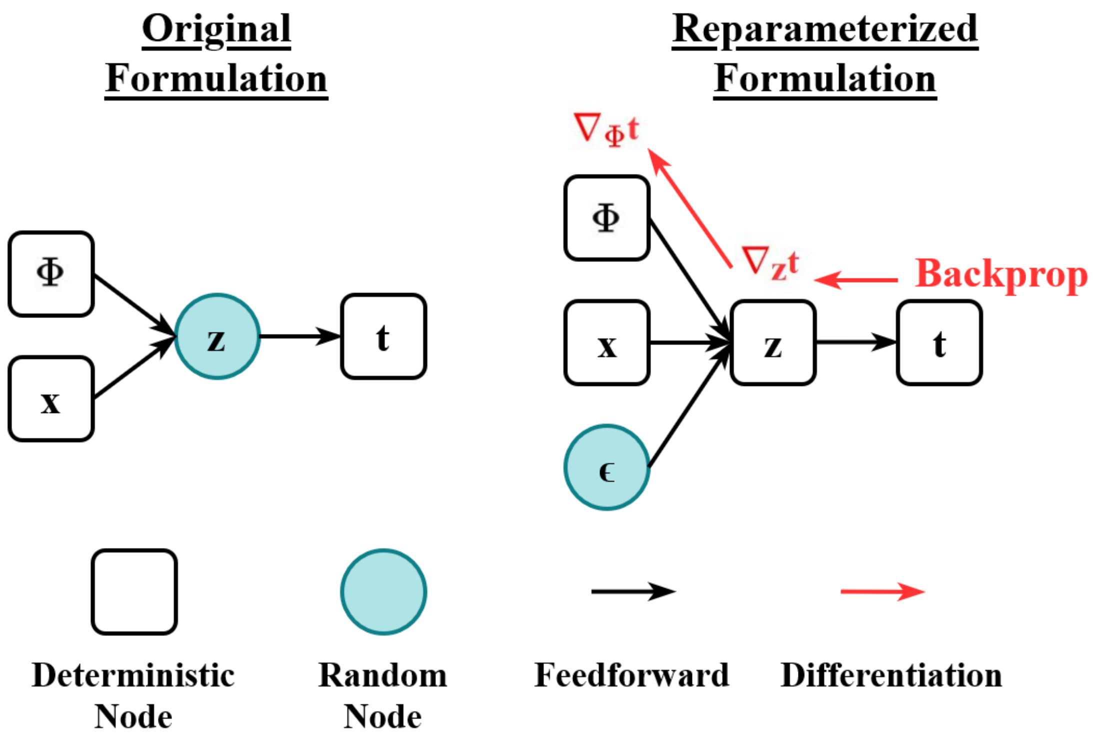
$$\ln p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \geq \ln p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \underbrace{D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i) || p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_i)\right)}_{\text{non-negative}} = E_{\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)}\left[\ln \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)}\right]$$

Reparameterization Trick

→ gradient descent according to $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$

issue: not readily possible for $\boldsymbol{\phi}$
(expecatation over $\boldsymbol{z}$, which is sampled from $q_{\boldsymbol{\phi}}$)

→ reparametrization to the rescue:
express randomness in $\boldsymbol{z}$ by independent auxiliary variable $\boldsymbol{\varepsilon}$

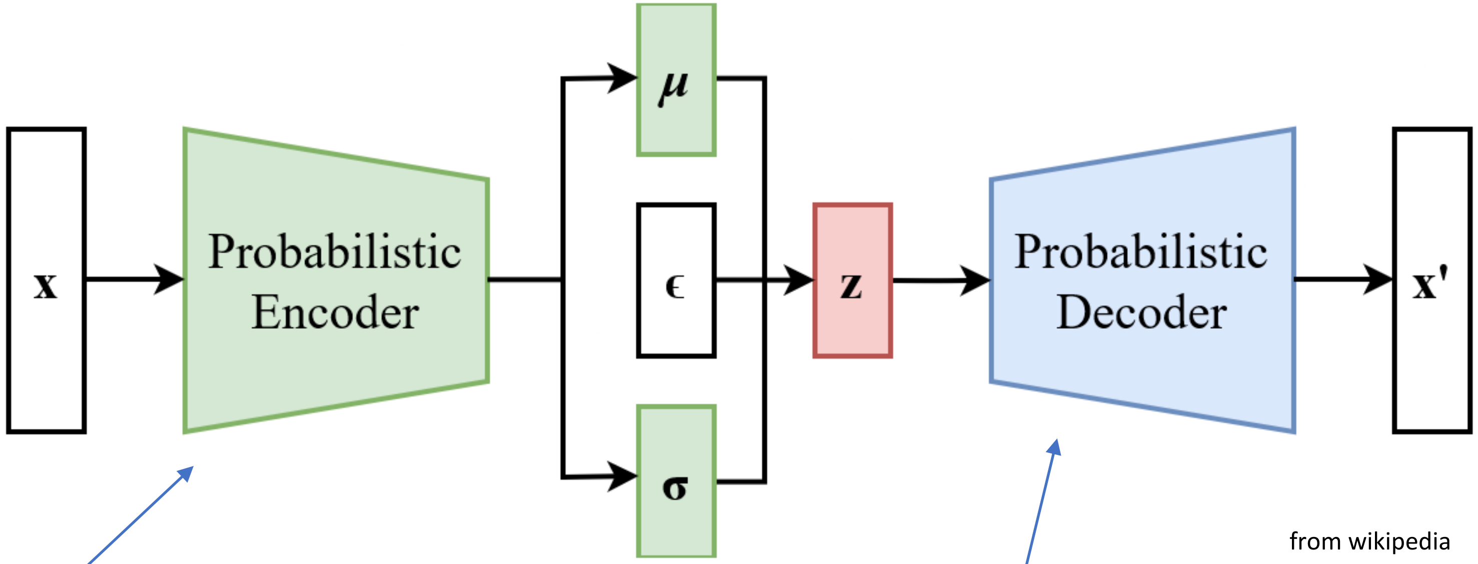


from wikipedia

Gaussian Approximation

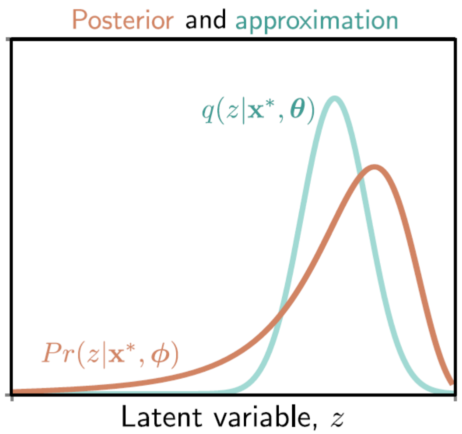
e.g., q_ϕ as multivariate Gaussian with diagonal covariance structure

→ learn mean and variance

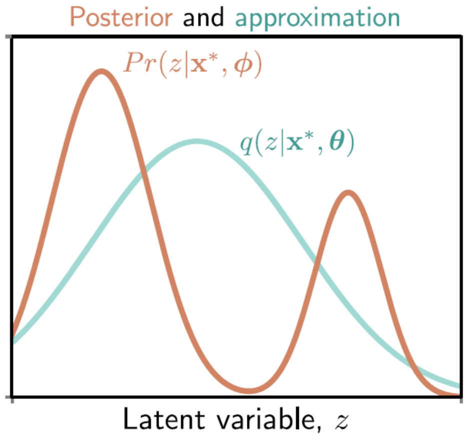


from wikipedia

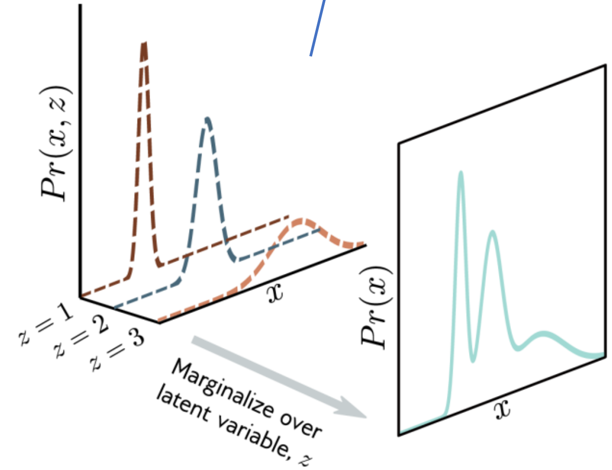
good approximation:



poor approximation:



source



Flow-Based Methods

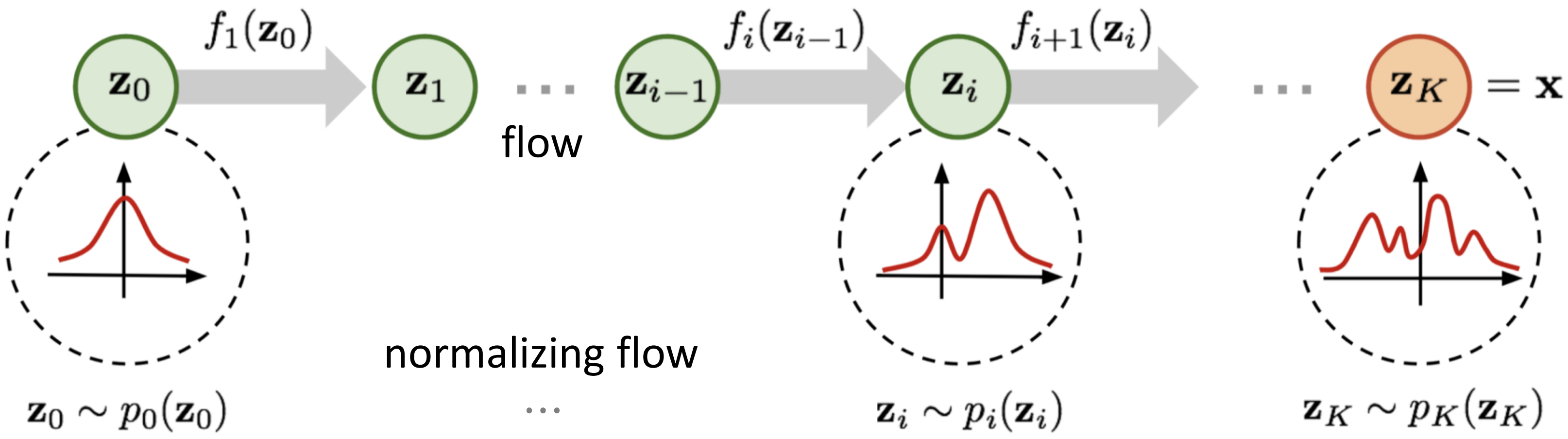
Normalizing Flows

idea: mapping of a simple probability distribution (often, standard normal distribution) into a complex one by sequence of invertible transformations (repeatedly applying the change-of-variable technique)

$$f(\boldsymbol{z}') = f(\boldsymbol{z}) \left| \det \frac{\delta f^{-1}}{\delta \boldsymbol{z}'} \right| = f(\boldsymbol{z}) \left| \det \frac{\delta f}{\delta \boldsymbol{z}} \right|^{-1}$$

log-likelihood:

$$\ln p_K(\boldsymbol{z}_K) = \ln p_0(\boldsymbol{z}_0) - \sum_{k=1}^{K} \ln \left| \det \frac{\delta f_k}{\delta \boldsymbol{z}_{k-1}} \right|$$



source

Usage in Generative Models

training: estimate maximum likelihood of normalizing flow (log-likelihood of last slide) by gradient descent (learn parameters θ of transformations f_θ^{-1}, e.g., to let $p_0(\boldsymbol{z})$ be Gaussian)

inference: sample from simple distribution $p_0(\boldsymbol{z})$ and transform it back to data distribution $p_K(\boldsymbol{x})$ via f_θ

advantages:

- instead of simple distributions like Gaussians, allow more complex latent encodings: real-world distributions usually much more complicated
- exact likelihood estimation (VAEs and diffusion models only return lower bound): allows density estimation (e.g., for anomaly detection)

Invertible Neural Networks

neural networks representing invertible/bijective functions can be used for normalizing flow transformations

- forward transformation to generate samples
- backward transformation to evaluate likelihoods

need for specialized architectures to construct reversible transform (e.g., affine coupling layers)

Diffusion Models

Idea

training: distort training data by successively adding random noise, then learn to reverse this process (denoising)

generation: sample random noise and run through the learned denoising process

advantages: easy to train, produce high-quality/realistic samples

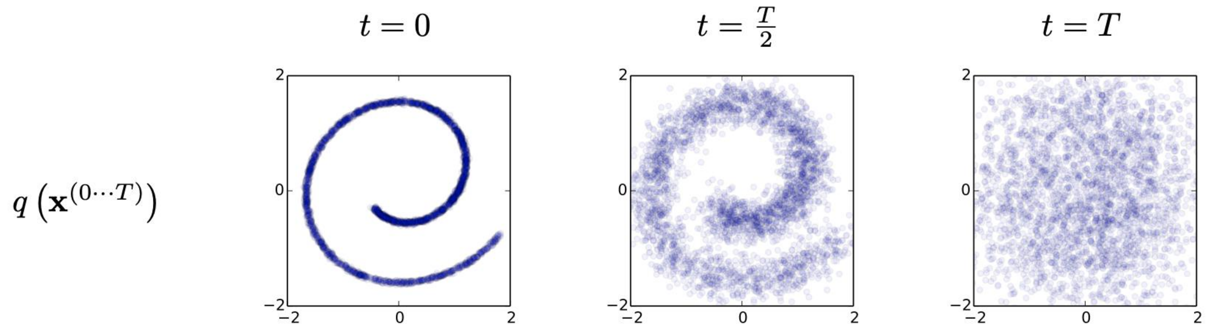
can be interpreted as special case of hierarchical VAE (one latent variable generates another) with fixed encoder and latent space of same size as the data → more sophisticated latent space than just Gaussian mixture in VAE

Forward Process

Markov chain of diffusion steps to slowly add Gaussian noise to data (inspired by non-equilibrium thermodynamics):

$$q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0) = \prod_{t=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}), \quad q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}\left(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \mathbf{I}\right)$$

- with variance schedule $\beta_1, \dots, \beta_T$ (hyperparameters, increasing with t)
- large T and small β_t → same functional form for forward and reverse processes, ending up with isotropic Gaussian distribution for $\boldsymbol{x}_T$



source

Reparametrization

conditional Gaussian distributions at each t:

sample $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ and set $\boldsymbol{x}_t = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\epsilon$

nice property: possible to directly sample $\boldsymbol{x}_t$ conditioned on $\boldsymbol{x}_0$ (no need to apply q repeatedly)

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

$$q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}\left(\boldsymbol{x}_t;\, \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\mathbf{I}\right)$$

with $\quad \alpha_t = 1-\beta_t, \quad \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$

conditioning on $\boldsymbol{x}_0$ also allows to handle $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$

Reverse Process

to generate new data samples, one needs to learn to reverse the diffusion process (starting from pure noise): neural network learning to gradually denoise data

overall loss as sum of losses for each time step t

for each t: D_{KL} between two Gaussians (closed form) $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$ and $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$

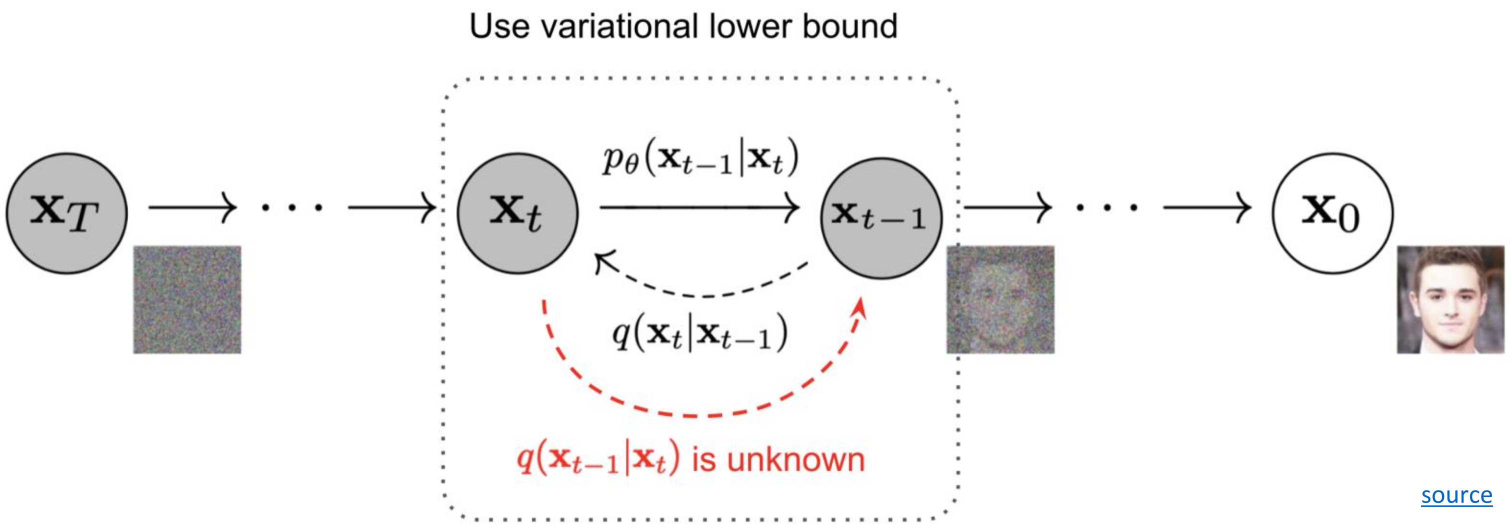
→ corresponds to VAE loss: maximizing ELBO

time-dependent Gaussian parameters:

$$p(\boldsymbol{x}_T) = \mathcal{N}(\boldsymbol{x}_T; 0, \mathbf{I})$$

$$p_\theta(\boldsymbol{x}_{0:T}) = p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$$

$$p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}\left(\boldsymbol{x}_{t-1};\ \boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \boldsymbol{\Sigma}_\theta(\boldsymbol{x}_t, t)\right)$$



source

Noise Prediction

reparametrization allows to learn added noise instead of Gaussian parameters:

Algorithm 1 Training

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

Algorithm 2 Sampling

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

source

L2-loss (MSE) between true and predicted Gaussian noise at time step t

use position embeddings (as network parameters are shared across time)

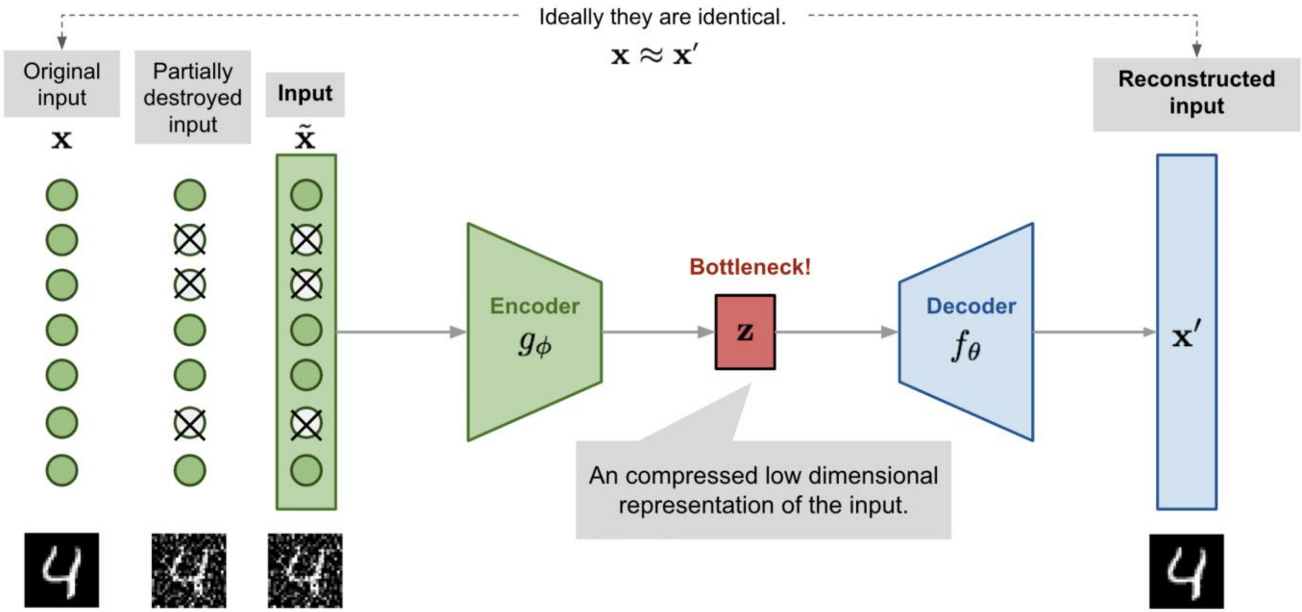
diffusion models can be interpreted as chain of denoising autoencoders (also connected to score-based generative modeling via Langevin dynamics)

Denoising Autoencoder

goal: avoid overfitting and improve robustness of plain autoencoder

learn to remove noise of distorted input $\tilde{\boldsymbol{x}}$ → restore original input $\boldsymbol{x}$

similar to dropout



source

differences of diffusion models to typical denoising autoencoders:

- no bottleneck (care about output here, not internal representation): latent space with high dimensionality (same as original data)
- handle many different noise levels with single set of shared parameters

Latent Diffusion Model

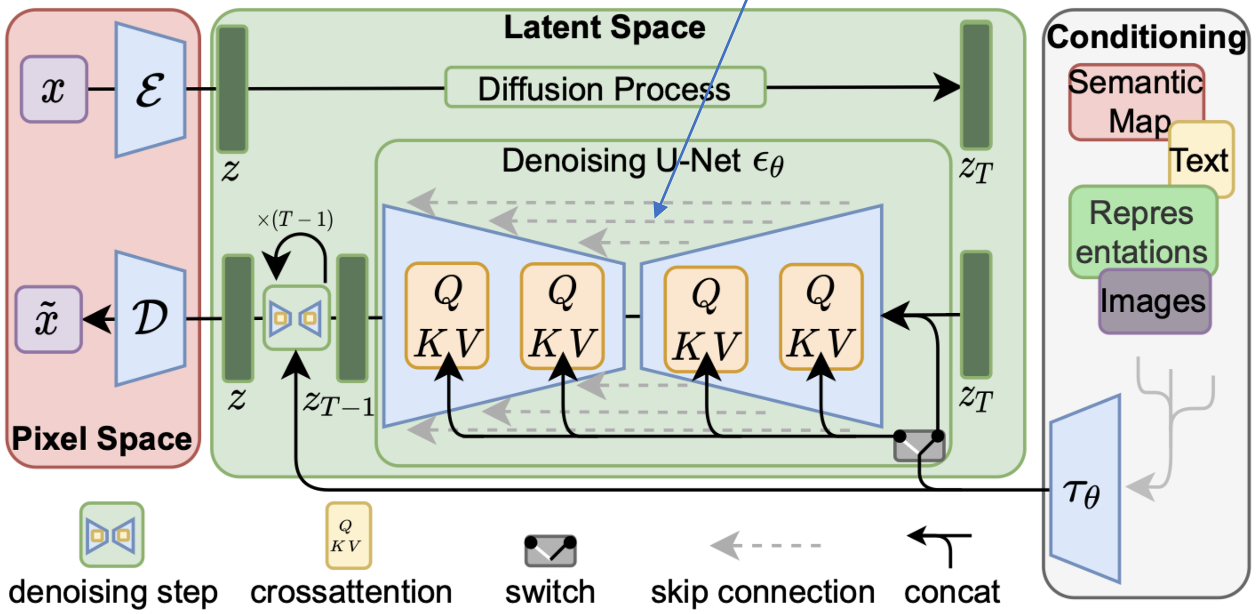
add noise to latent representation rather than raw data

→ significant speedup

diffusion models highly flexible in terms of architecture: only require same input and output dimensionality (autoencoder-like)

- often (convolutional) U-Net architectures
- but also (vision) transformers possible (e.g., DiT)

skip connections between layers operating at the same scale



source

use of attention mechanism for flexible conditioning

Conditioned Generation

Conditional GANs

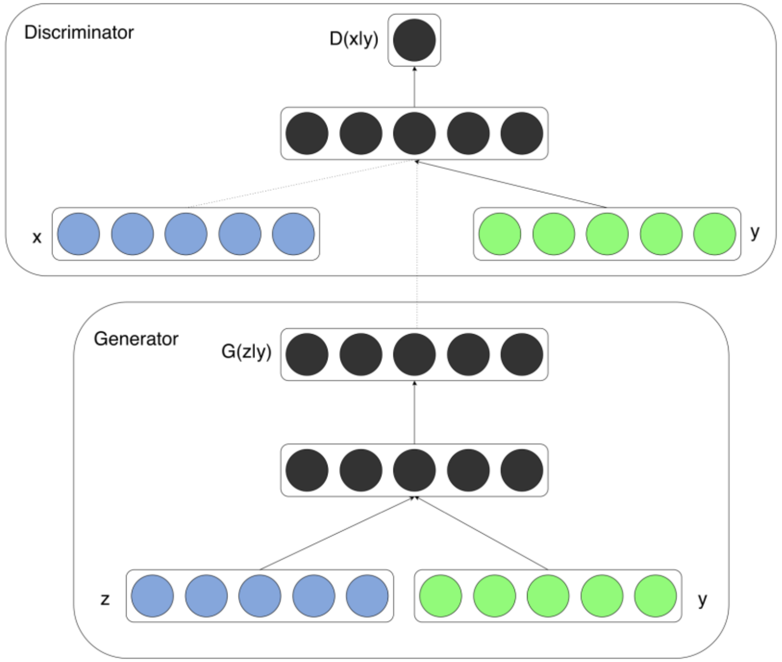
as discussed so far, generative methods give no control over what kind of data is generated (limited usability)

→ need for conditional approach (e.g., conditioning on describing text)

example GANs:

transform usual GAN to conditional model by feeding extra information y (e.g., class labels) as additional input layer into both generator and discriminator

$$L(\boldsymbol{x}_i) = E_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\ln D(\boldsymbol{x}_i|y_i)] + E_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}\left[\ln\left(1 - D(\boldsymbol{x}_i|y_i)\right)\right]$$



source

Guided Diffusion

similar idea as softmax temperature in auto-regressive LLMs

tradeoff between diversity (unconditioned) and fidelity (guidance)

ways to condition on class information in diffusion process:

- classifier guidance: perturbation of class-conditional diffusion model by separately trained classifier model $p_\theta(y|\boldsymbol{x}_t)$
  - $\hat{\boldsymbol{\mu}}_\theta(\boldsymbol{x}_t|y) = \boldsymbol{\mu}_\theta(\boldsymbol{x}_t|y) + s \cdot \boldsymbol{\Sigma}_\theta(\boldsymbol{x}_t|y) \cdot \nabla_{\boldsymbol{x}_t} \log p_\theta(y|\boldsymbol{x}_t)$
  - guidance can also be free-form text, e.g., from CLIP model
- classifier-free guidance: randomly replace label in class-conditional diffusion model with null label during training

  extrapolate in direction of conditioned model during sampling:

  $\hat{\epsilon}_\theta(\boldsymbol{x}_t|y) = \epsilon_\theta(\boldsymbol{x}_t|\emptyset) + s \cdot \left(\epsilon_\theta(\boldsymbol{x}_t|y) - \epsilon_\theta(\boldsymbol{x}_t|\emptyset)\right)$

guidance scale: hyperparameter for tradeoff between sample quality and diversity

"Pembroke Welsh corgi"

source

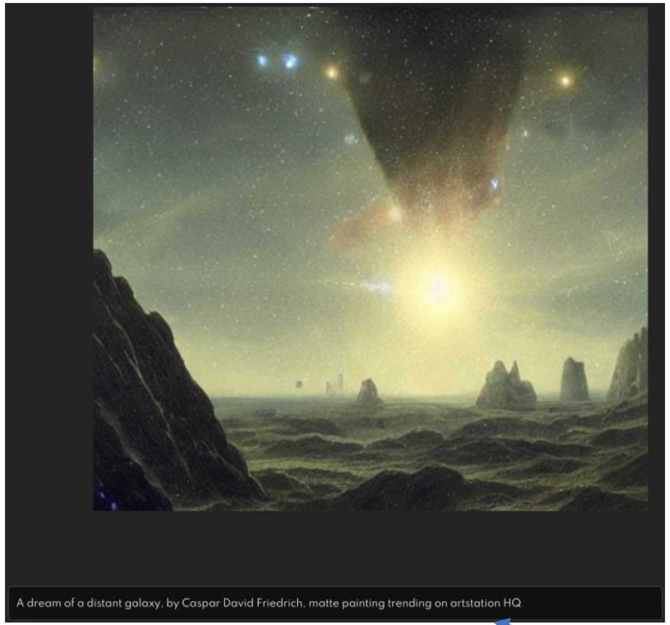
Text-to-Image

plenty of applications: DALL-E, Stable Diffusion, ImageGen, Midjourney, ...

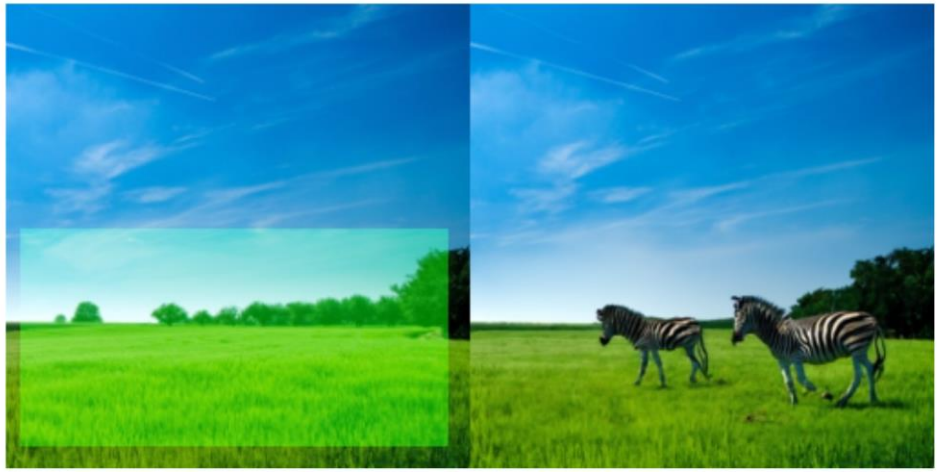
rather "translations"

also text-to-speech (VALL-E, Speech T5, …) (and speech recognition, e.g., Whisper), text-to-video (Make-A-Video, Lumiere, Sora, …)

web app for Stable Diffusion: DreamStudio



inpainting example (GLIDE):



"zebras roaming in the field"

source

prompt

Literature

papers:

- variational autoencoder
- normalizing flows
- GAN
- denoising diffusion, latent diffusion

Movie-like Intelligence

emergent capabilities of complex systems difficult to foresee

mini examples in contemporary ML:

- large language models
- multi-agent reinforcement learning

philosophical: emotions and consciousness in humans may also have occurred as emergent capabilities (But that does not mean the same will happen with AI.)

ideas for paths toward general intelligence:

- reward is enough
- small-world/scale-free networks