

# Generative Models

*Discriminative vs Generative*

Understanding Machine Learning

# Archetype: Naïve Bayes

probabilistic model:

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

Bayes' rule                      constant                      to be estimated

approach:

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

# Independence Assumption

(naïve) assumption: conditional independence of features given target

$$P(X_j | Y, X_1, \dots, X_{j-1}, X_{j+1}, \dots, X_p) = P(X_j | Y)$$

$$\Rightarrow P(Y | X_1, \dots, X_p) = \frac{P(Y) \prod_{j=1}^p P(X_j | Y)}{P(X_1, \dots, X_p)}$$


- 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

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

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

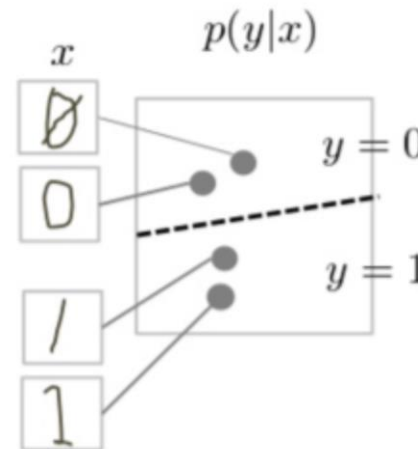
generative models: predict joint probability  $P(Y, \mathbf{X})$  (what allows to create new data samples) or directly generate new data samples

or just  $P(\mathbf{X}) \rightarrow$  unsupervised (or self-supervised) learning

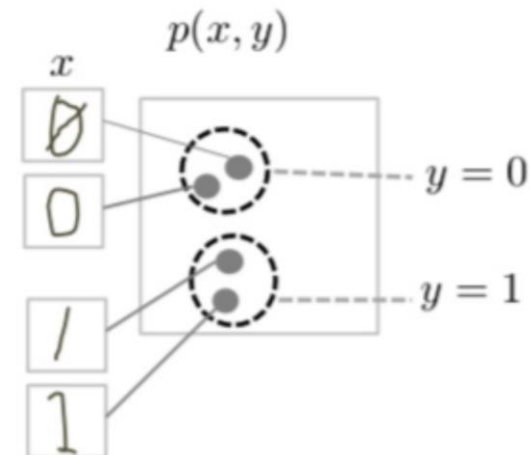
discriminative models: predict conditional probability (or probability distribution for regression)  $P(Y|\mathbf{X})$  or directly output (label for classification, real value for regression)

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,  $\log \left( \frac{P(y_i=1|x_i)}{P(y_i=0|x_i)} \right)$ , 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)

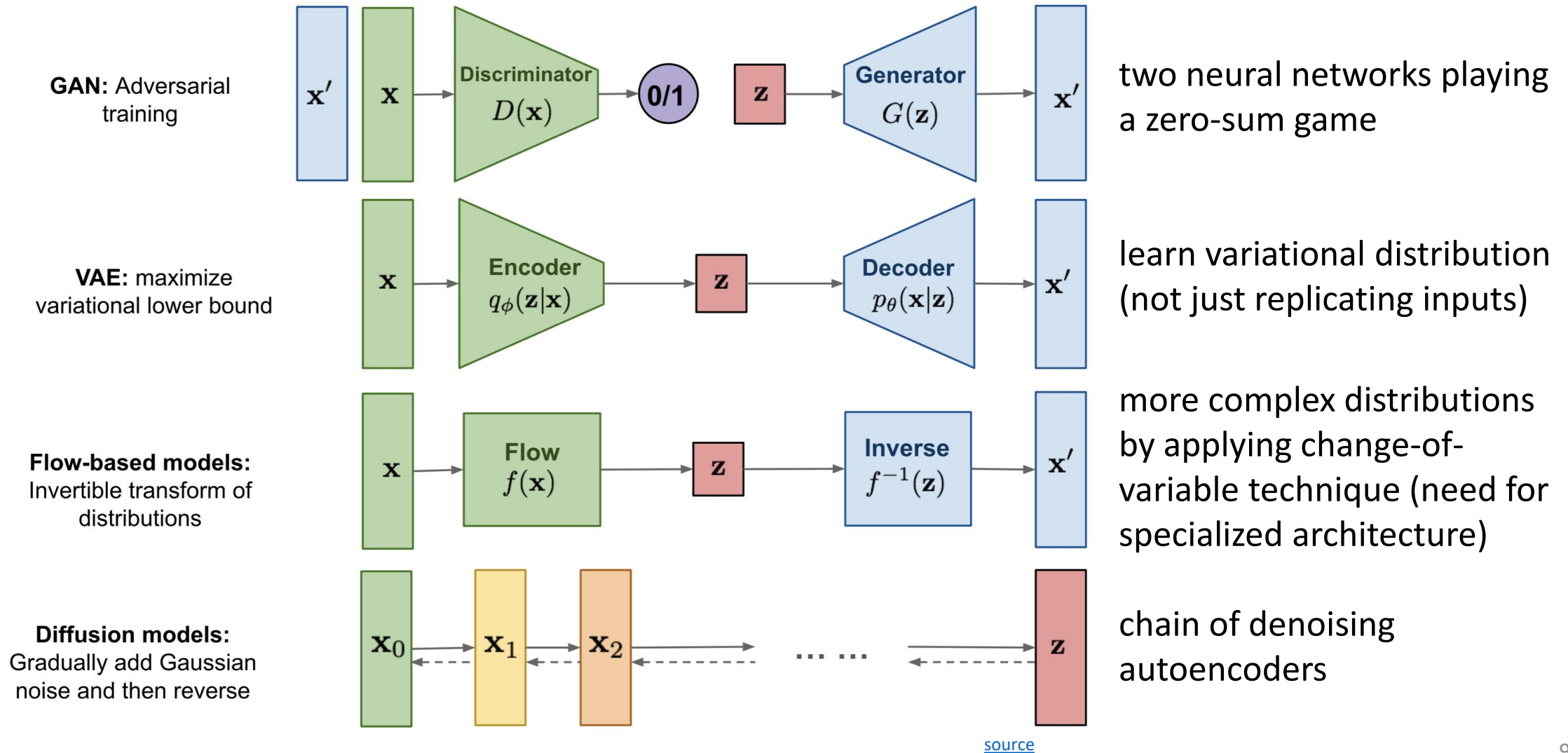
but generative methods do more than discriminative ones: model full data distribution

→ 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



# Different Types of Generative Models



# 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

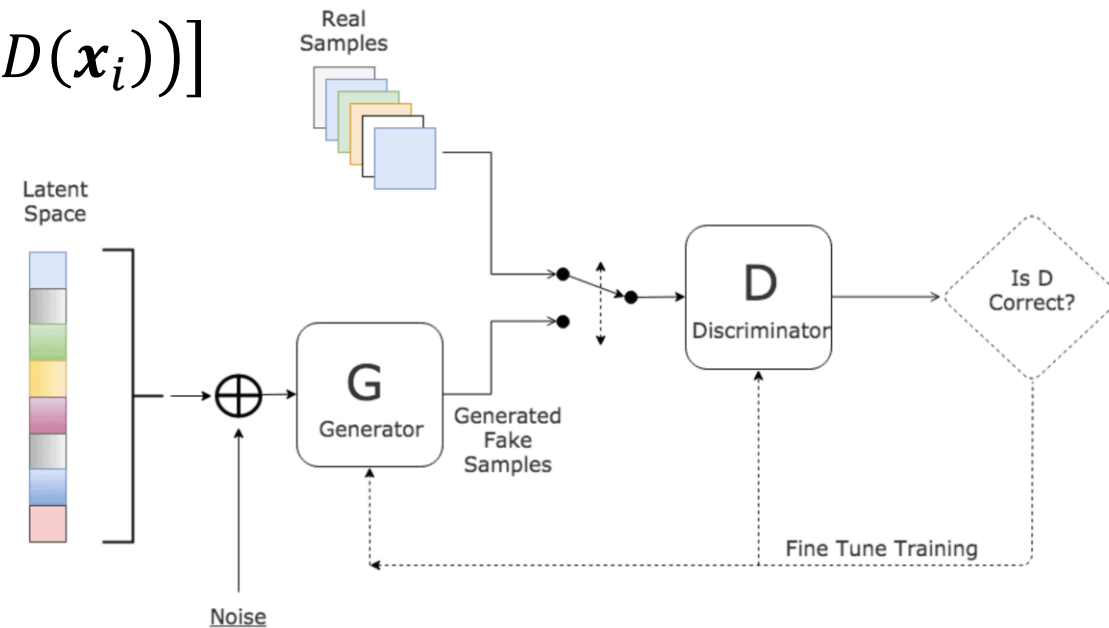
idea:  $G$  not trained directly to minimize reconstruction error of real samples, but to fool  $D \rightarrow$  self-supervised approach

# Formulation

common loss for generator and discriminator:

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

- 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  $D = 0.5$

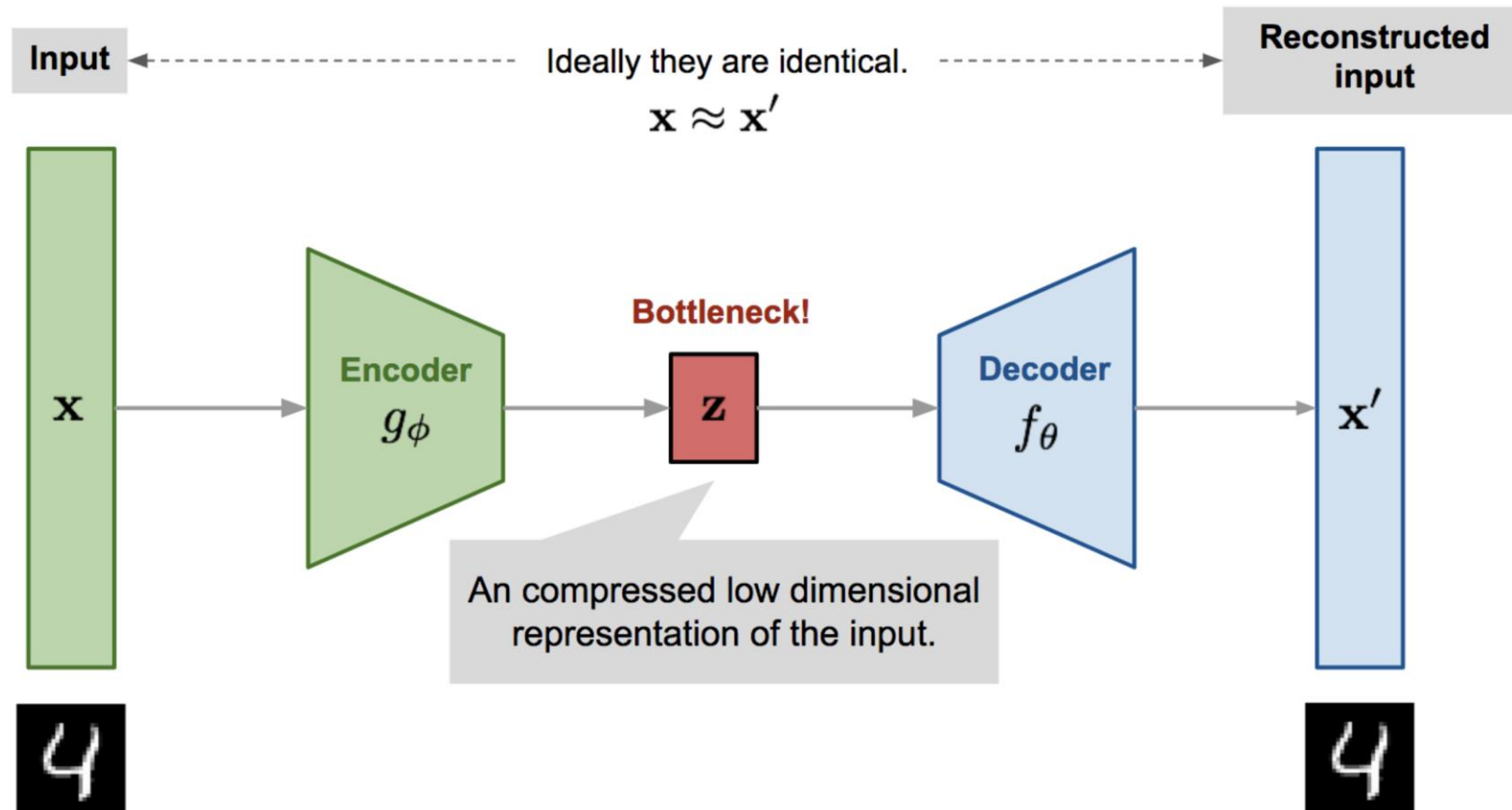
issue: potentially unstable training

# Variational Autoencoders (VAE)

# Recap: Autoencoder

(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

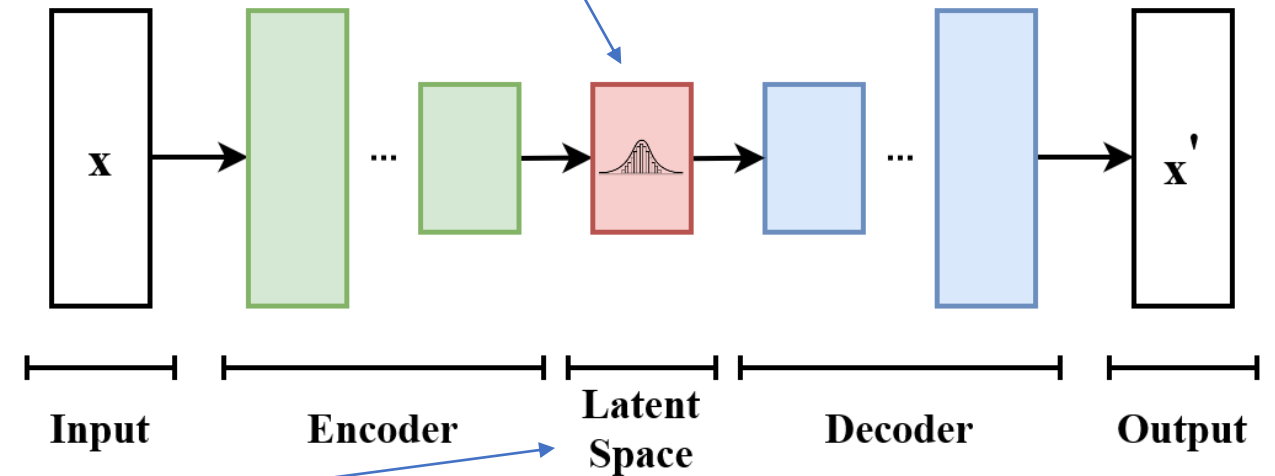
to be precise: parametrized variational distribution of latent encoding variables  $\mathbf{z}$

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

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

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

from which to sample



from wikipedia

Variational Bayesian Method



# Encoder and Decoder Networks

encoder: find posterior  $p_{\theta}(\mathbf{z}|\mathbf{x})$   
unfortunately, generally intractable

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

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

→ amortized inference:

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

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

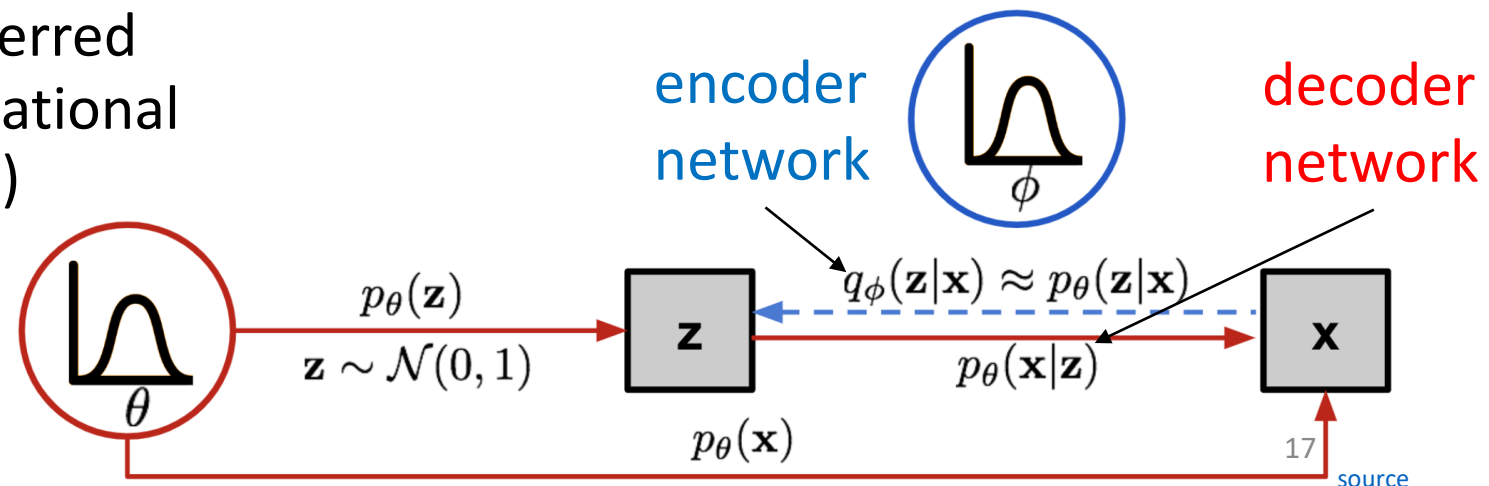
1. sample  $\mathbf{z}_i$  (from Gaussian)

2. generate  $\mathbf{x}_i$  (similar to real data)

→ maximize:  $p_{\theta}(\mathbf{x}_i) = \int p_{\theta}(\mathbf{x}_i|\mathbf{z}) p_{\theta}(\mathbf{z}) d\mathbf{z}$

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

in VAE: network weights  $\theta$



# VAE Loss: ELBO

VAE loss function to be minimized according to network weights:

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

maximize likelihood of observed data (minimize reconstruction error)

and

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

can be interpreted as regularizer

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

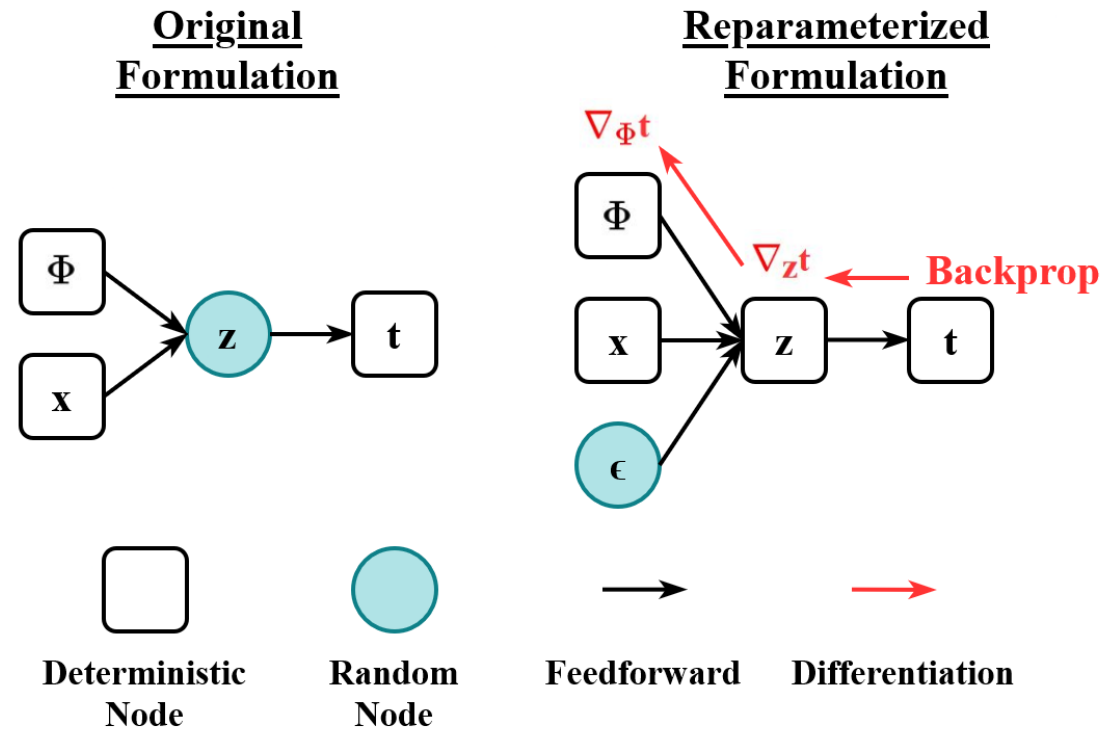
$$\ln p_{\boldsymbol{\theta}}(\mathbf{x}_i) \geq \ln p_{\boldsymbol{\theta}}(\mathbf{x}_i) - \underbrace{D_{KL} \left( q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_i) || p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}_i) \right)}_{\text{non-negative}} = E_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}_i)}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

# Reparameterization Trick

→ gradient descent according to  $\theta$  and  $\phi$

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

→ reparameterization to the rescue:  
express randomness in  $\mathbf{z}$  by independent auxiliary variable  $\epsilon$

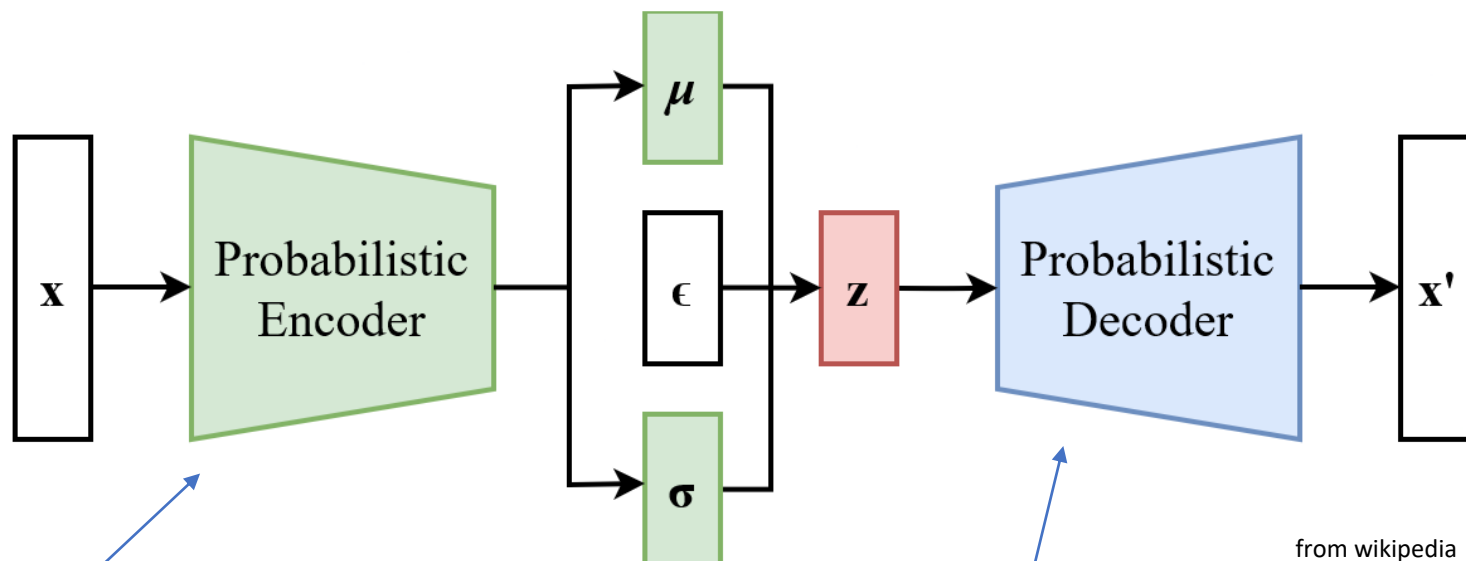


from wikipedia

# Gaussian Approximation

e.g.,  $q_\phi$  as multivariate Gaussian with diagonal covariance structure

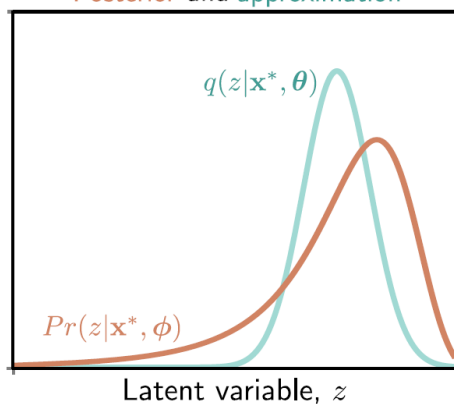
→ learn mean and variance



from wikipedia

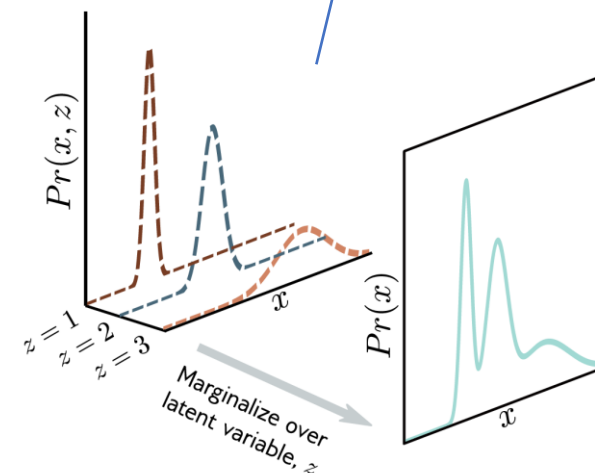
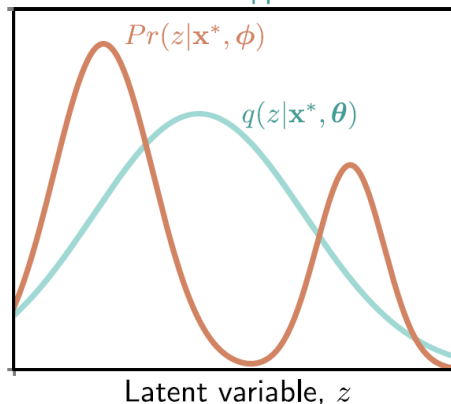
good approximation:

Posterior and approximation



poor approximation:

Posterior and approximation



[source](#)

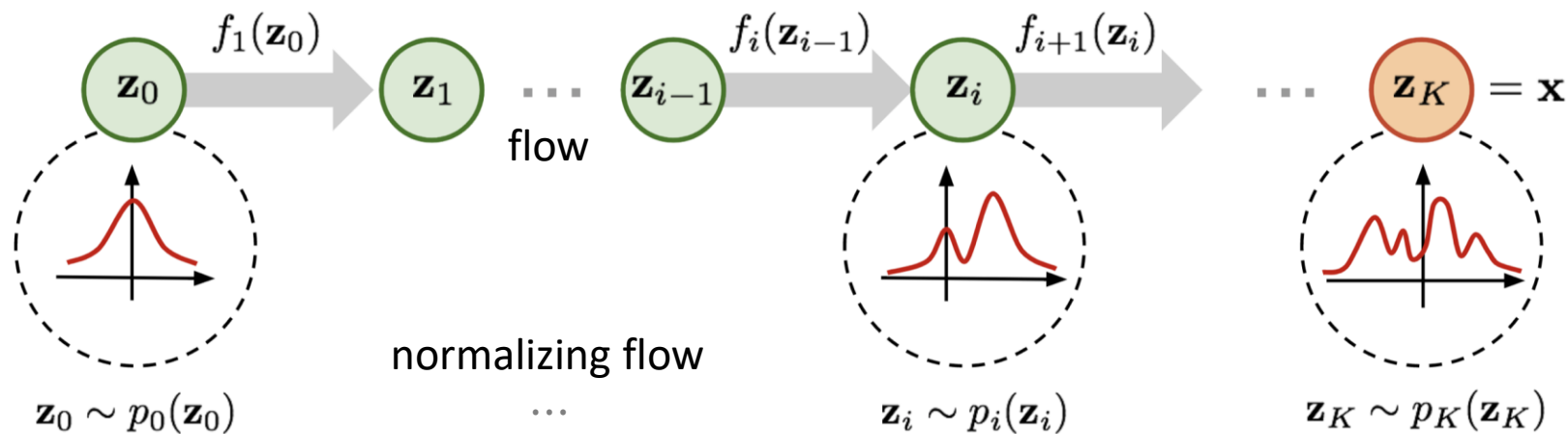
# Flow-Based Methods

# 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)

log-likelihood:

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



# Usage in Generative Models

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

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

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

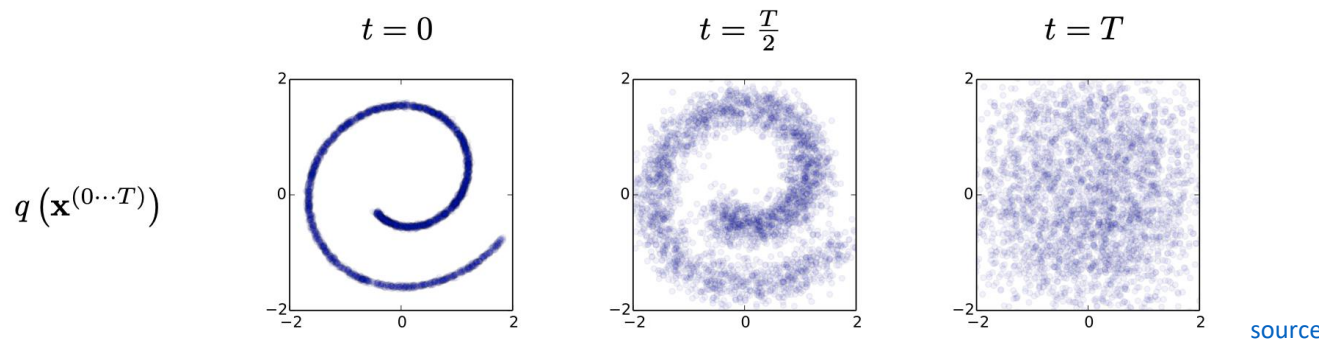
can be interpreted as special case of hierarchical VAE (one latent variable generates another) with fixed encoder and latent space of same size as the data → more sophisticated latent space than just Gaussian mixture in VAE

# Forward Process

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

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

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



# Reparametrization

conditional Gaussian distributions at each  $t$ :

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

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

$$\begin{aligned}\mathbf{x}_t &= \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon \\ q(\mathbf{x}_t | \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I})\end{aligned}$$

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

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

# 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)  $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$  and  $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$

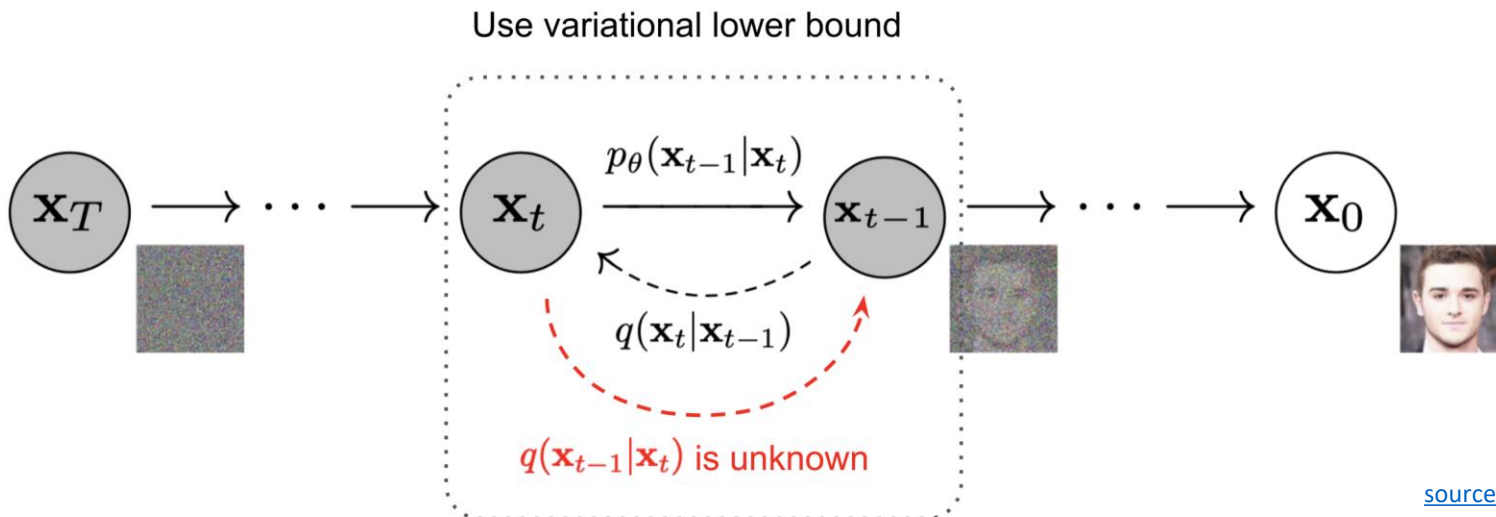
→ corresponds to VAE loss: maximizing ELBO

time-dependent Gaussian parameters:

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

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

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



# Noise Prediction

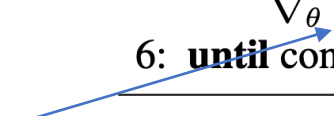
reparametrization allows to learn added noise instead of Gaussian parameters:

---

**Algorithm 1** Training

---

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



---

**Algorithm 2** Sampling

---

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

---

L2-loss (MSE) between true and predicted Gaussian noise at time step  $t$   
use position embeddings (as network parameters are shared across time)

[source](#)

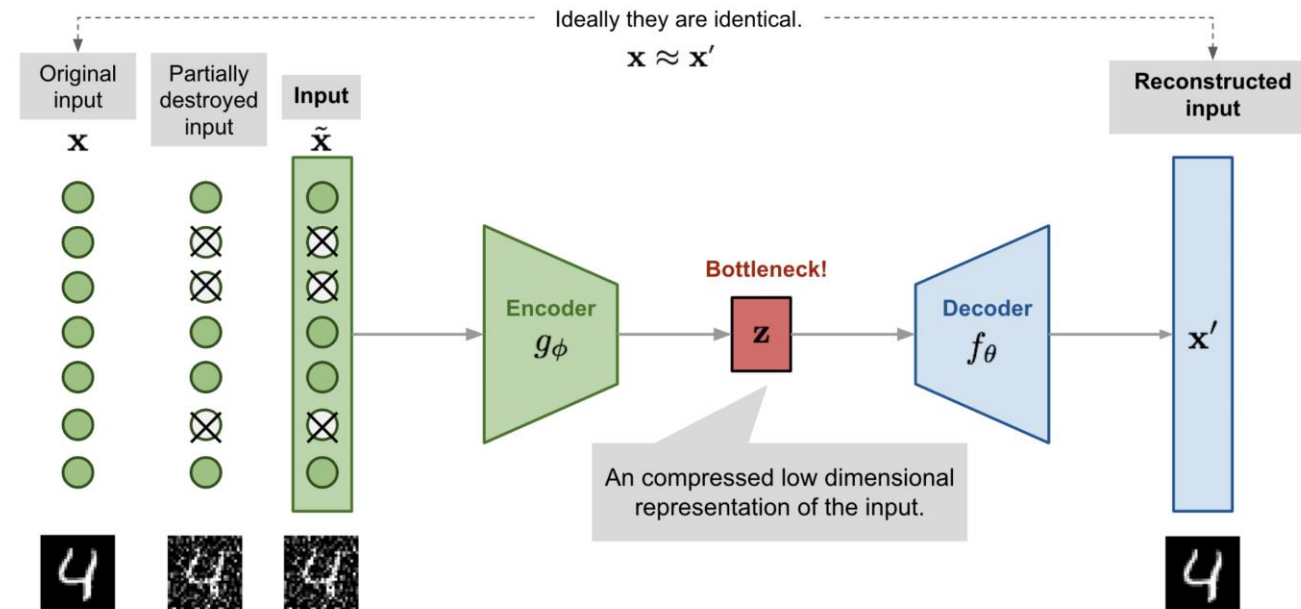
diffusion models can be interpreted as chain of denoising autoencoders (also connected to score-based generative modeling via Langevin dynamics)

# Denoising Autoencoder

goal: avoid overfitting and improve robustness of plain autoencoder

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

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

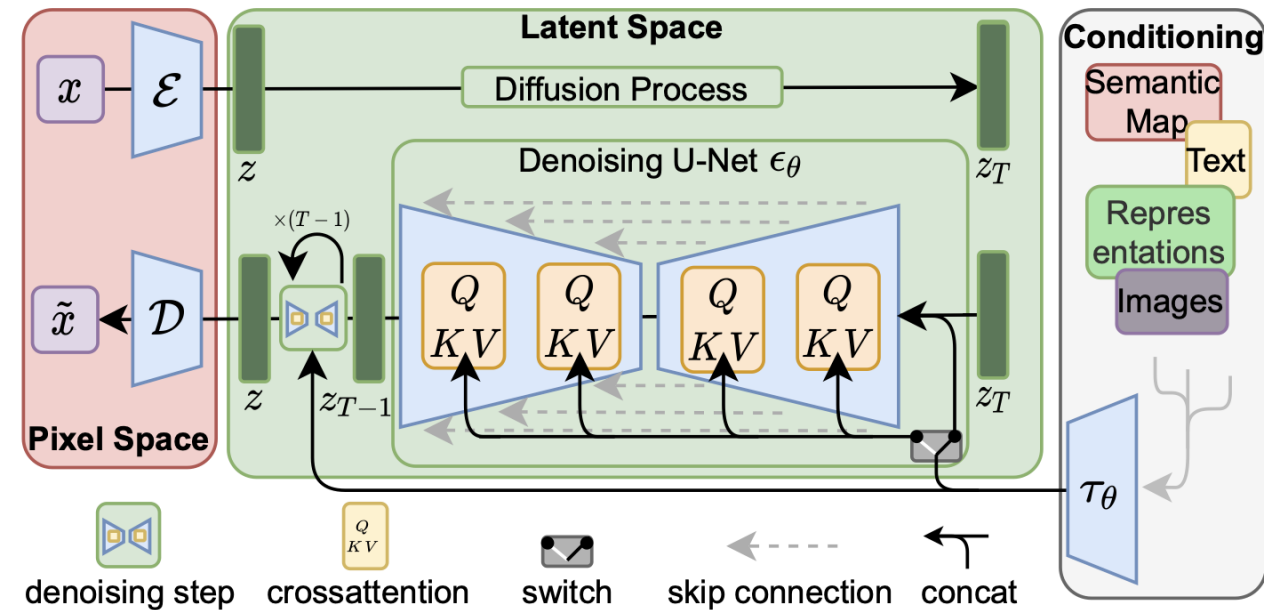
# Latent Diffusion Model

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 (skip connections)
- but also (vision) transformers possible (e.g., [DiT](#))



[source](#)

use of attention mechanism for flexible conditioning



# Conditioned Generation

# Conditional GANs

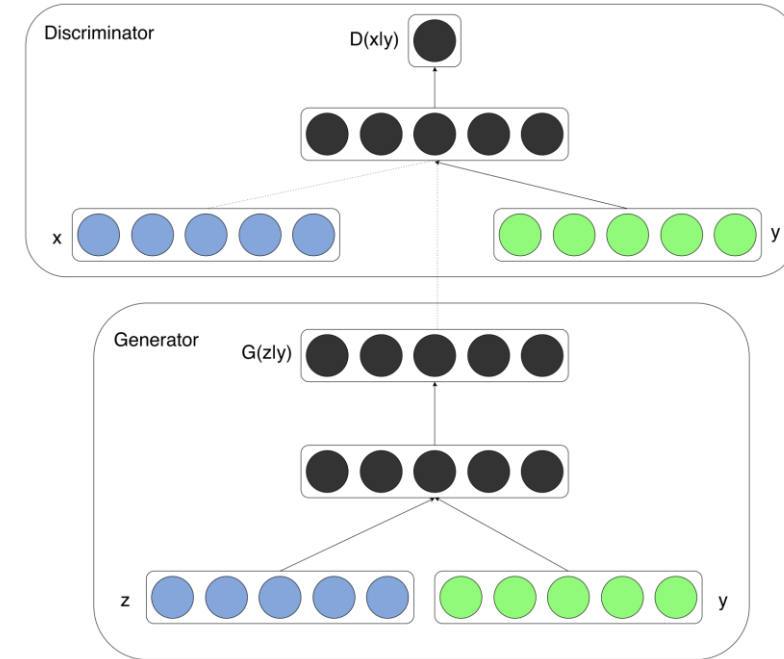
as discussed so far, generative methods give no control over what kind of data is generated (limited usability)

→ 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

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



[source](#)

# Guided Diffusion

ways to condition on class information in diffusion process:

- classifier guidance: perturbation of class-conditional diffusion model by separately trained classifier model  $p_\theta(y|x_t)$ 
  - $\hat{\mu}_\theta(x_t|y) = \mu_\theta(x_t|y) + s \cdot \Sigma_\theta(x_t|y) \cdot \nabla_{x_t} \log p_\theta(y|x_t)$
  - 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:

$$\hat{\epsilon}_\theta(x_t|y) = \epsilon_\theta(x_t|\emptyset) + s \cdot (\epsilon_\theta(x_t|y) - \epsilon_\theta(x_t|\emptyset))$$

guidance scale: hyperparameter for tradeoff between sample quality and diversity

similar idea as softmax temperature in auto-regressive LLMs

tradeoff between diversity (unconditioned) and fidelity (guidance)



“Pembroke Welsh corgi”

[source](#)

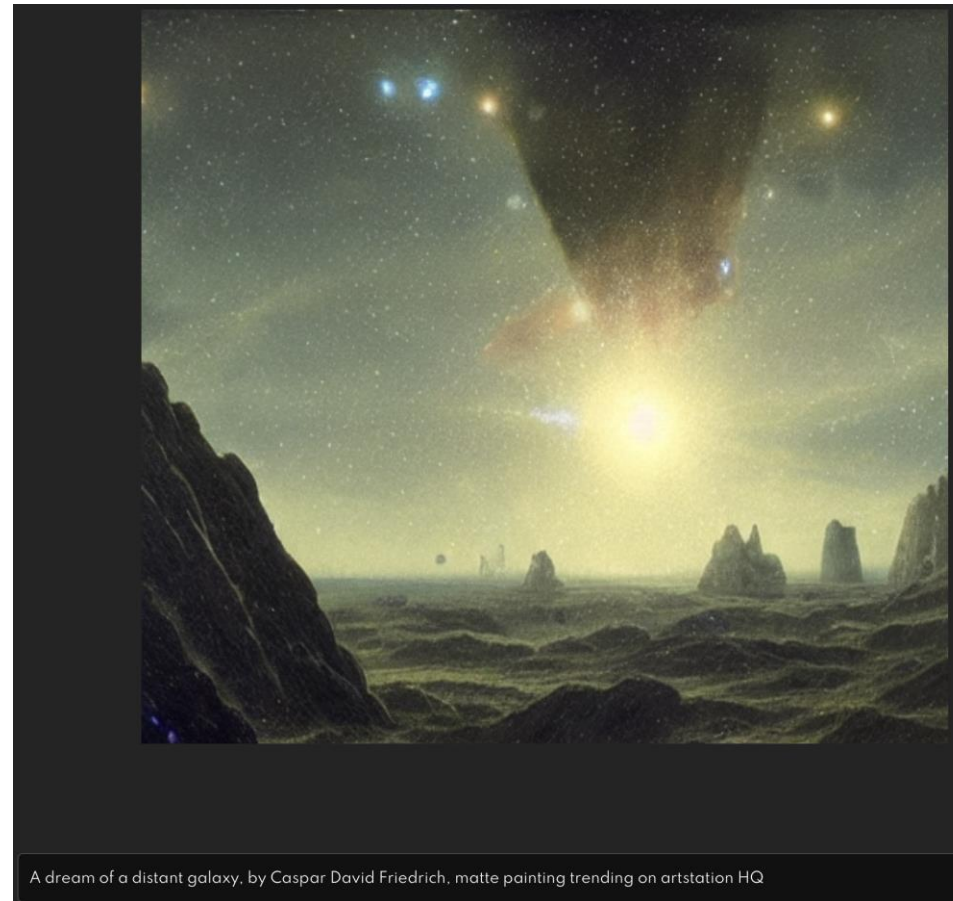
# Applications

plenty of implementations available

text-to-image:  
[DALL-E 2](#), [Stable Diffusion](#),  
[GLIDE](#), [ImageGen](#), ...

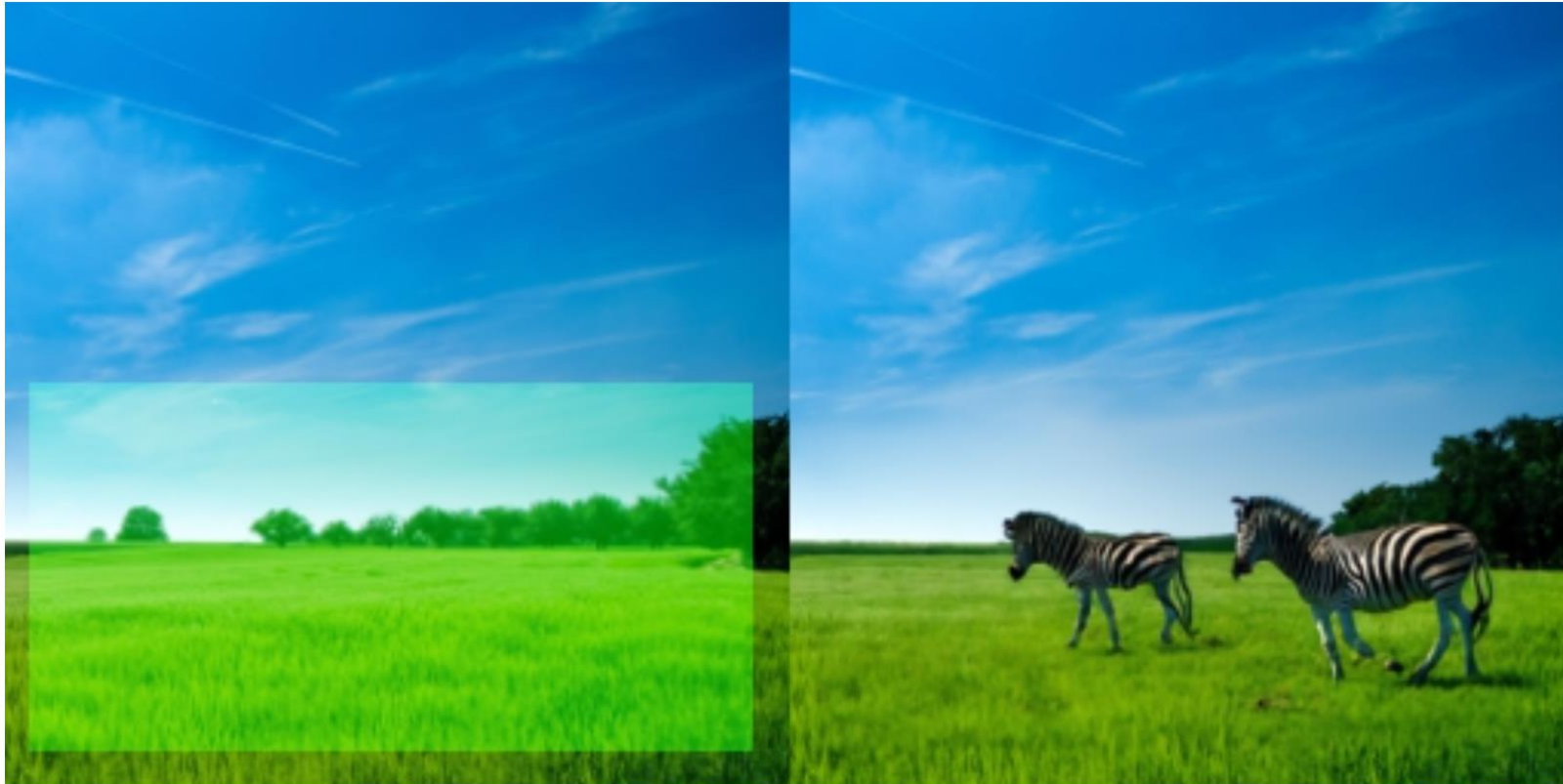
also text-to-speech (e.g.,  
[VALL-E](#)), text-to-video  
(e.g., [Make-A-Video](#)), ...

web app for Stable Diffusion: [DreamStudio](#)



# Inpainting

GLIDE example:



“zebras roaming in the field”

[source](#)

# Literature

papers:

- [variational autoencoder](#)
- [normalizing flows](#)
- [GAN](#)
- [denoising diffusion](#), [latent diffusion](#)



HWARZENEGGER



# 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

