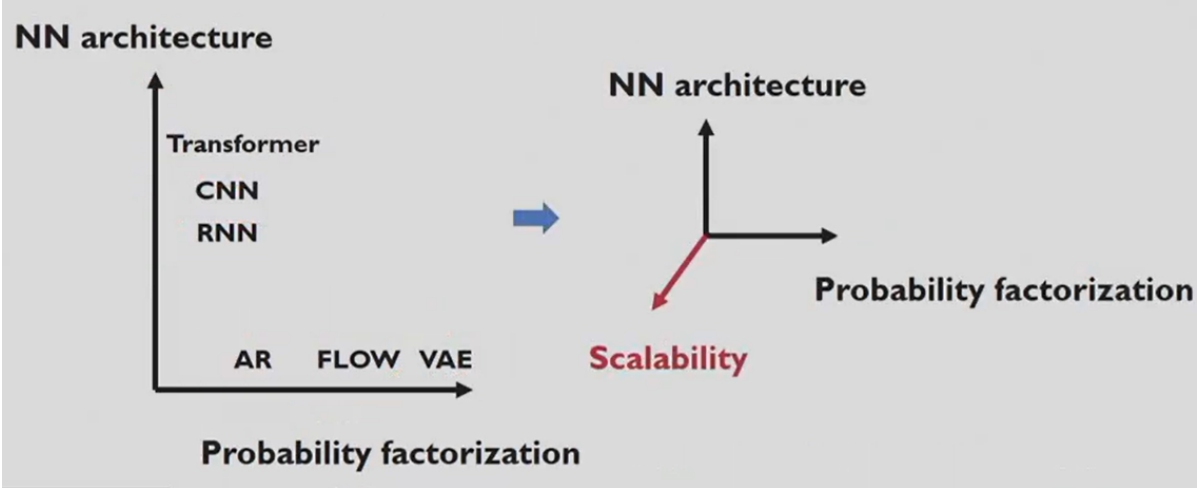
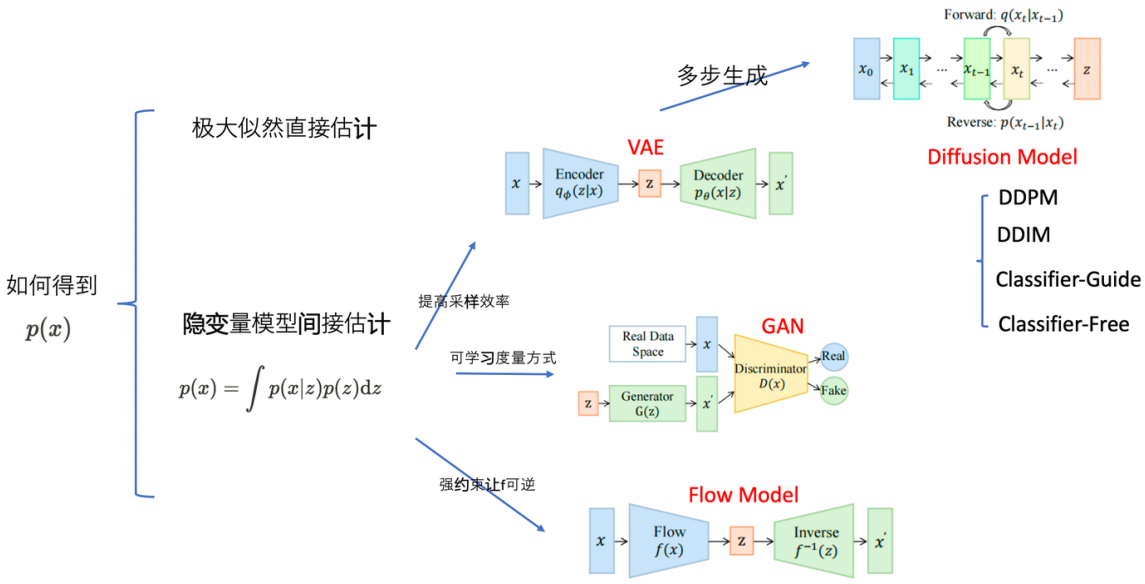
Notes for Deep Generative Model Learning

(the whole notebook was written by JEd. And the DGM here merely focus on the instances on image generation)





0 - Pre-knowledge

- Transformer
- Resnet
- RNN & LSTM
- Maximum Likelihood Estimation (MLE)
- Divergence
- Basic Machine Learning (*e.g.* Gradients)
- Monte Carlo Estimation
- Stochastic Differential Equation (SDE) (? not necessary better to know)
- CLIP

1 - Auto-Regressive Model (ARM)

Principle

Suppose there is a high-dimensional random variable X, where x_i represents the little pixel points within it. According to the chain rule, we know that $p(x)$ can be calculated by this fomular:

$$p(x) = p(x_1) \prod_{d=2}^{D} p(x_d | X_{<d})$$

Under the autoregressive assumption, generation problem has become a **sequence problem**.

Examples

Entire Linear Regression

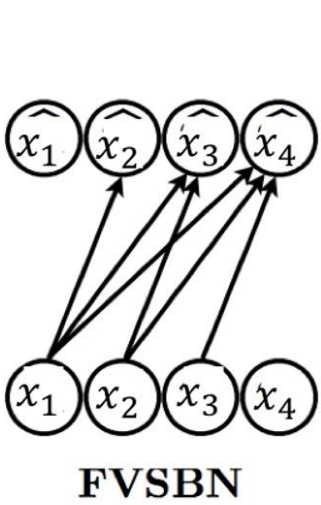
The MINIST Task, we assume

$$p(x) = p_{\text{CPT}}(x_1; \alpha^{(1)}) p_{\text{logit}}(x_2 | x_1; \alpha^{(2)}) p_{\text{logit}}(x_3 | x_1, x_2; \alpha^{(3)}) \cdots p_{\text{logit}}(x_D | x_{<D}; \alpha^{(D)})$$

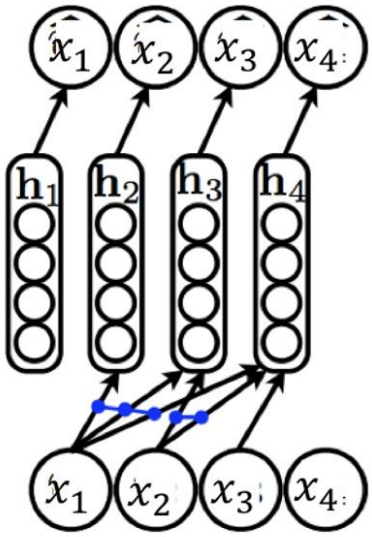
$$\text{where,} \quad p_{\text{CPT}}(X_1 = 0; \alpha^{(1)}) = \alpha^{(1)}, p_{\text{CPT}}(X_1 = 0; \alpha^{(1)}) = 1 - \alpha^{(1)}$$

$$p_{\text{logit}}(X_2 = 1 | x_1; \alpha^{(2)}) = \sigma(\alpha_0^{(2)} + \alpha_1^{(2)} x_1)$$

Without NN, in this kind of modeling of black and white images, the pixel computation entirely relies on linear regression.



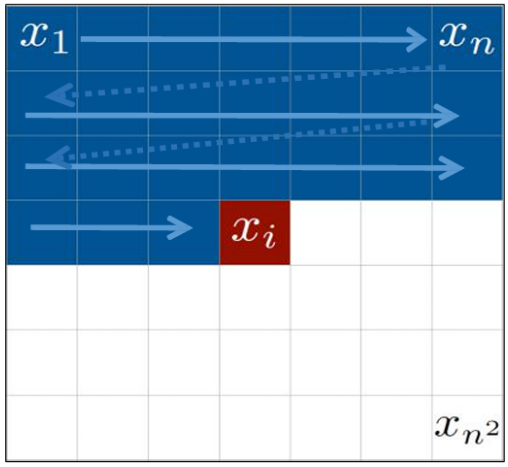
FVSBN



NADE

For multi-channel images, autoregressive generation follows the order of RGB. The probability for the i-th pixel could become:

$$p(x_i) = p\left(x_{i,R} | x_{<i}\right) p\left(x_{i,G} | x_{<i}, x_{i,R}\right) p\left(x_{i,B} | x_{<i}, x_{i,R}, x_{i,G}\right)$$



RNN & PixelRNN (×)

- Recap of RNN Recurrent Neural Network Regularization
- PixelCNN Network Structure
- “Masked Convolution” & “BlindSpot”

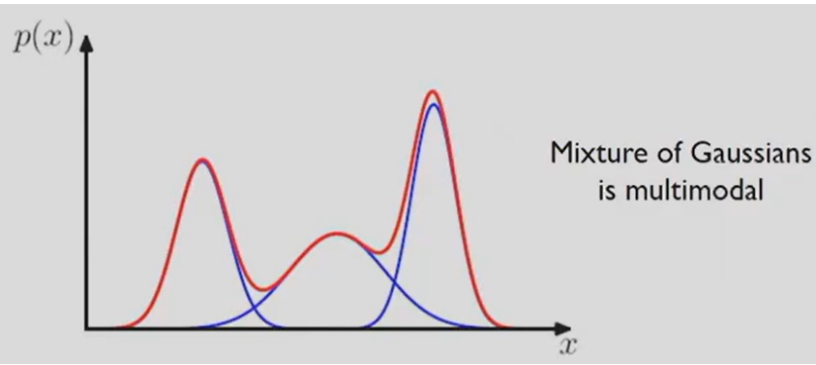
> 由于像素本身的性质，对图片生成进行scaling或者rotation的操作是得额外学习的。

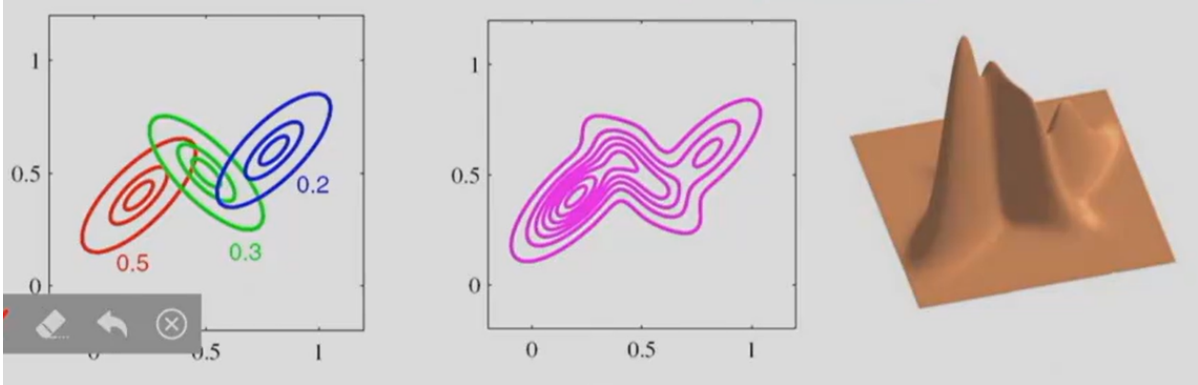
2 - Variational Auto-Encoders (VAE)

Mixture of Gaussians

- A simple family of multi-modal distributions
- treat unimodal Gaussians as **basis (or component) distributions**
- superpose multiple Gaussians via **convex combination**

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \sigma_k^2)$$





MLE for Mixture of Gaussians

$$\theta = (\pi, \mu, \Sigma)$$

$$\mathcal{L}(\mu, \Sigma) = \log p(\mathcal{D}|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \right)$$

Since this is quite complicated, we can estimate it through a heuristic procedure (Expectation-Maximization Algorithm).

$$\frac{\partial \mathcal{L}}{\partial \mu_k} = 0$$

$$\Rightarrow \quad \sum_{n=1}^{N} \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)} \Sigma_k^{-1}(x_n - \mu_k) = 0$$

$$\Rightarrow \quad \mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n, N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

$$\frac{\partial \mathcal{L}}{\partial \Sigma_k} = 0$$

$$\Rightarrow \quad \Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T$$

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = 0$$

$$\Rightarrow \quad \sum_{n=1}^{N} \frac{\mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)} + \lambda = 0$$

$$\Rightarrow \quad \pi_k = \frac{N_k}{N}$$

Since, the set of couple conditions is

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n$$

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T$$

$$\pi_k = \frac{N_k}{N}$$

And the key factor to get them coupled is

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

Then, we can iterate the function.

> - **E-step**: estimate the responsibilities
>
> $$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

- **M-step**: re-estimate the parameters

$$\mu_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})x_n$$

$$\Sigma_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})(x_n-\mu_k)(x_n-\mu_k)^T$$

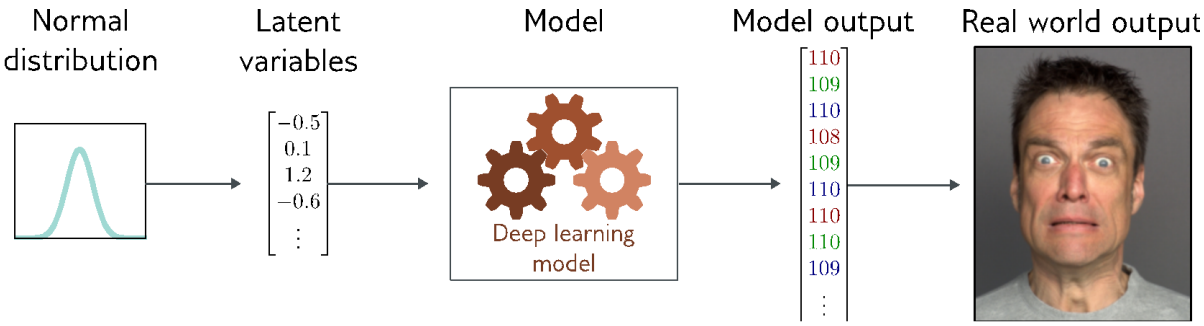
$$\pi_k = \frac{N_k}{N}$$

So, **initialization plays a key role to succeed!**

Latent variable model

Latent variable models take an indirect approach to describing a probability distribution $P(x)$ over a multi-dimensional variable x. Instead of directly writing the expression for $P(x)$, they model a joint distribution $P(x, z)$ of the data x and an unobserved hidden or latent variable z. They then describe the probability of $P(x)$ as a marginalization of this joint probability so that:

$$P(x) = \int P(x,z)\mathrm{d}z = \int P(x|z)P(z)\mathrm{d}z, \text{ where } P(z) \text{ is the prior.}$$



The advantages of latent model

- Expressiveness power (e.g, single modal to muki modain MoG)
- Dimensionality reduction, structures and interpretability
- Conditional sampling and manipulation

Latent Representation is very powerful.(credit to Lecture 7-8 From Autoencoder to VAE 110mins, 了解Sora背后的原理，来学习深度生成模型！(第2讲))

- **Power of Latent Representation**
  - t-SNE visualisation on MNIST: PCA vs. Autoencoder

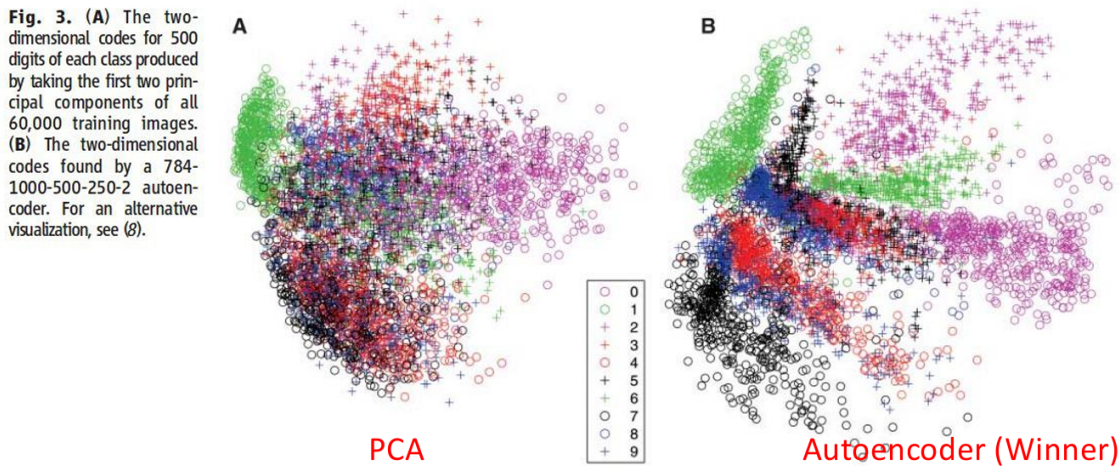


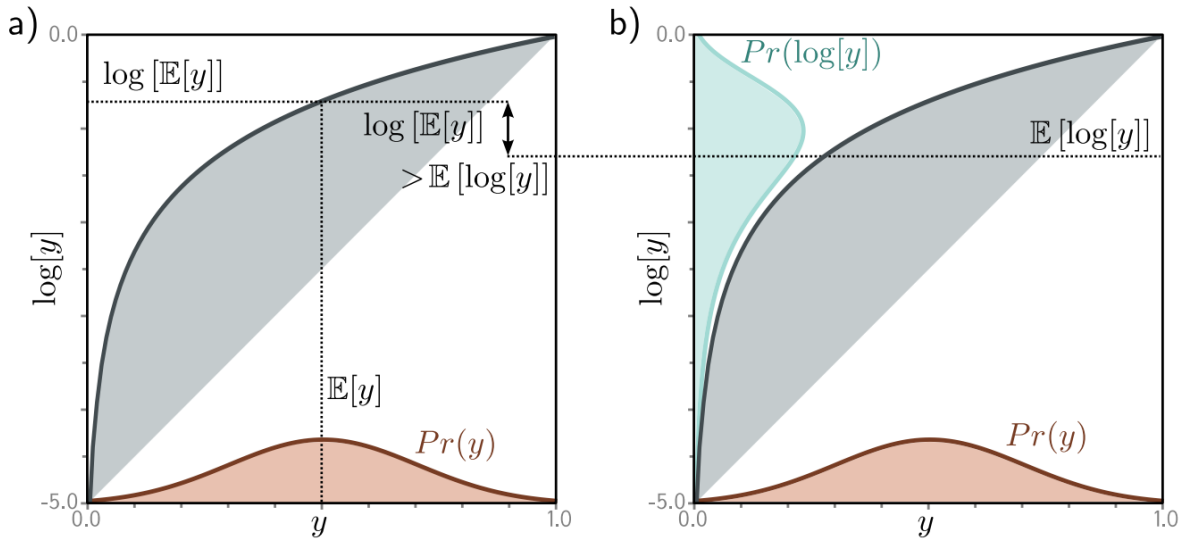
Fig. 3. **(A)** The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. **(B)** The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).

2006 Science paper by Hinton and Salakhutdinov

Evidence lower bound (ELBO)

- Jensen's inequality

$$\log \mathbb{E}_{p(x)}[x] \geq \mathbb{E}_{p(x)}[\log x]$$



- Now, let's proceed with some derivations.

$$\begin{aligned}
\log p(x) &= \log \int p(x,z)\mathrm{d}z \\
&= \log \int \frac{p(x,z)q_\phi(z|x)}{q_\phi(z|x)}\mathrm{d}z \\
&= \log \mathbb{E}_{q_\phi(z|x)}\left[\frac{p(x,z)}{q_\phi(z|x)}\right] \\
&\geq \mathbb{E}_{q_\phi(z|x)} \log\left[\frac{p(x,z)}{q_\phi(z|x)}\right] \\
&= \mathrm{ELBO}[\theta,\phi]
\end{aligned}$$

Here, $q_\phi(z|x)$ is a flexible approximate variational distribution with parameters ϕ that we seek to optimize. Additionally, in practice, θ is the parameter of the distribution $q(z)$. And the ELBO can also be written as:

$$\mathrm{ELBO}[\theta,\phi] = \int q(z|\theta) \log\left[\frac{P(x,z|\phi)}{q(z|\theta)}\right]\mathrm{d}z$$

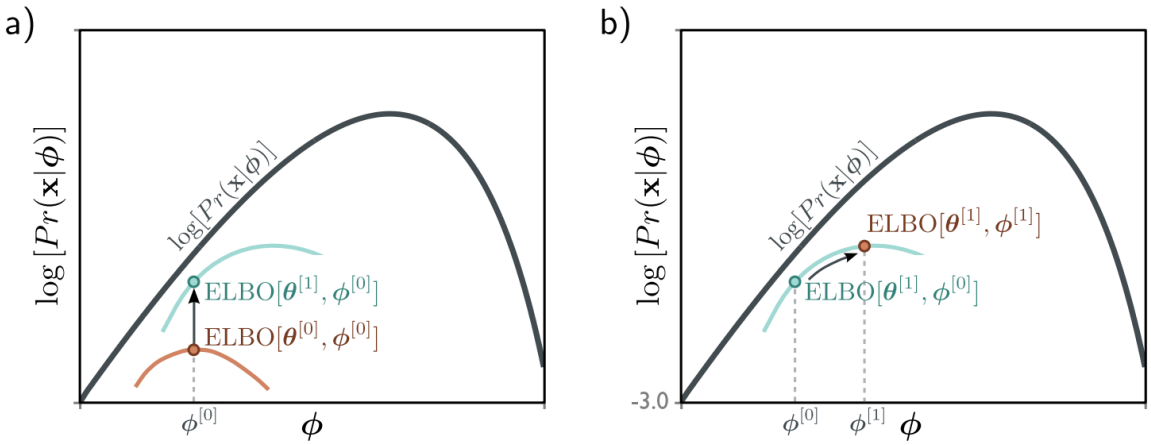


Figure 17.6 Evidence lower bound (ELBO). The goal is to maximize the log-likelihood $\log[Pr(\mathbf{x}|\boldsymbol{\phi})]$ (black curve) with respect to the parameters $\boldsymbol{\phi}$. The ELBO is a function that lies everywhere below the log-likelihood. It is a function of both $\boldsymbol{\phi}$ and a second set of parameters $\boldsymbol{\theta}$. For fixed $\boldsymbol{\theta}$, we get a function of $\boldsymbol{\phi}$ (two colored curves for different values of $\boldsymbol{\theta}$). Consequently, we can increase the log-likelihood by either improving the ELBO with respect to a) the new parameters $\boldsymbol{\theta}$ (moving from colored curve to colored curve) or b) the original parameters $\boldsymbol{\phi}$ (moving along the current colored curve).

Here are some notes about the ELBO:

- The ELBO is tractable because $p(x, z)$ and $q_\phi(z|x)$ are defined manually.
- The EBLO is valid for any q, but we optimize p to obtain a tight lower bound.
- Maximizing ELBO is equivalent to minimize $KL(q_\phi(z|x) \,||\, p(z|x))$, Variational Inference(VI).

$$
\begin{aligned}
\log p(x) &= \log \int p(x,z)\mathrm{d}z \\
&= \mathbb{E}_{q_\phi(z|x)}[\log p(x)] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{p(z|x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z|x)}\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p(z|x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z|x)}\right] + D_{\mathrm{KL}}(q_\phi(z|x) \,||\, p(z|x)) \\
&\geq \mathbb{E}_{q_\phi(z|x)} \log\left[\frac{p(x,z)}{q_\phi(z|x)}\right] (\text{tips: } D_{\mathrm{KL}} \geq 0)
\end{aligned}
$$

- The ELBO is *tight* when, for a fixed value of ϕ, the ELBO and the likelihood function coincide.

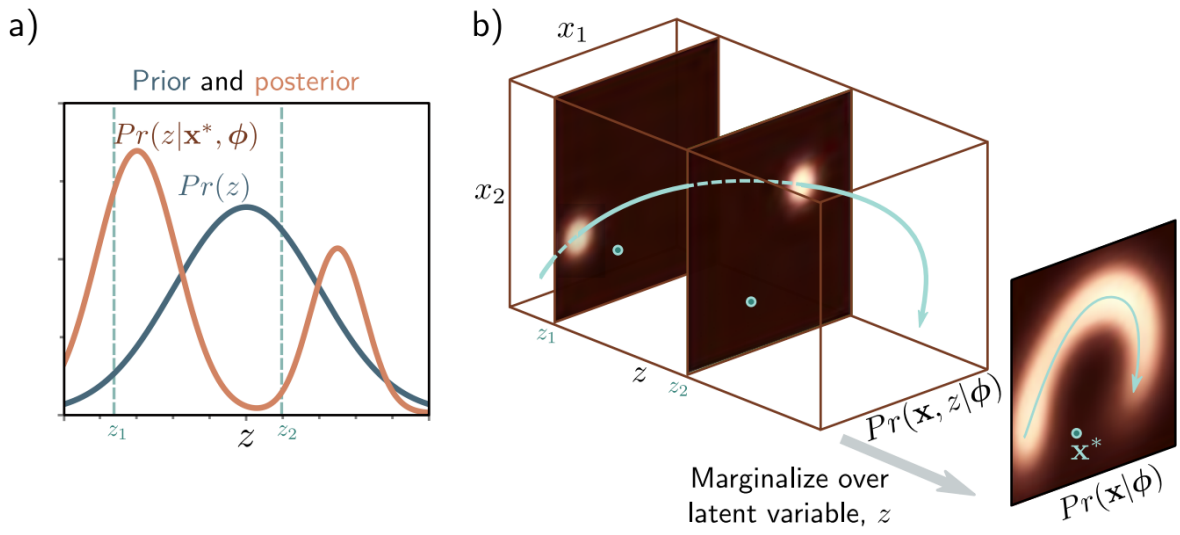


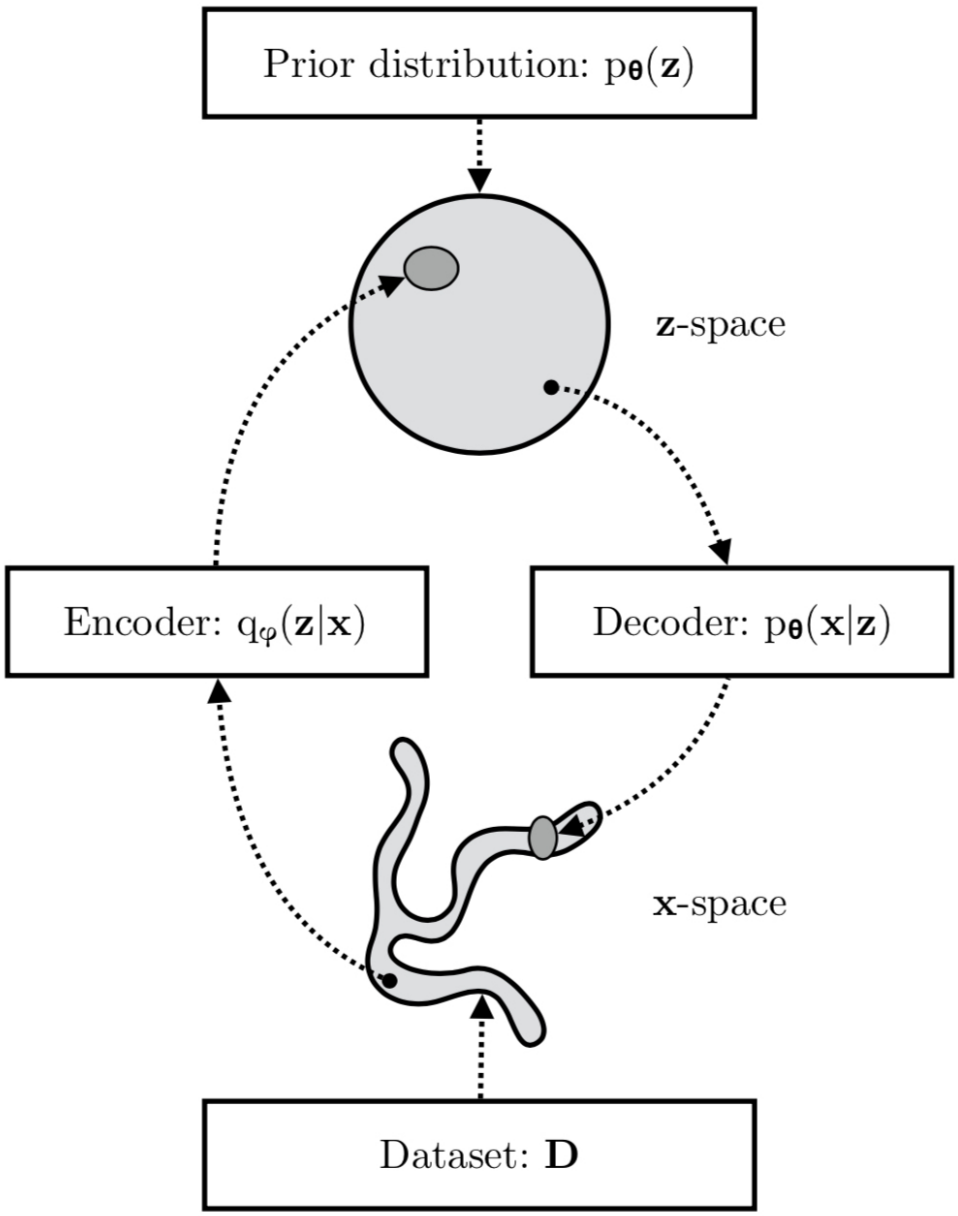
Figure 17.7 Posterior distribution over latent variable. a) The posterior distribution $Pr(z|\mathbf{x}^*, \boldsymbol{\phi})$ is the distribution over the values of the latent variable z that could be responsible for a data point $\mathbf{x}^*$. We calculate this via Bayes' rule $Pr(z|\mathbf{x}^*, \boldsymbol{\phi}) \propto Pr(\mathbf{x}^*|z, \boldsymbol{\phi})Pr(z)$. b) We compute the first term on the right-hand side (the likelihood) by assessing the probability of $\mathbf{x}^*$ against the symmetric Gaussian associated with each value of z. Here, it was more likely to have been created from z_1 than z_2. The second term is the prior probability $Pr(z)$ over the latent variable. Combining these two factors and normalizing so the distribution sums to one gives us the posterior $Pr(z|\mathbf{x}^*, \boldsymbol{\phi})$.

Variational Inference (×)

Variational inference methods in Bayesian inference and machine learning are techniques which are involved in approximating intractable integrals.

https://www.cs.cmu.edu/~epxing/Class/10708-15/notes/10708_scribe_lecture13.pdf

https://ashkush.medium.com/variational-inference-gaussian-mixture-model-52595074247b



Variational approximation

The ELBO is tight when $q(z|\theta)$ is the posterior $P(z|x,\phi)$.

In principle, we can compute the posterior using Bayes' rule:

$$P(z|x,\phi) = \frac{P(x|z,\phi)P(z)}{P(x|\phi)}$$

Here,

- $P(z|x,\phi)$ is the **posterior probability**, representing the distribution of the latent variable z given the observed data x and parameters ϕ.
- $P(x|z,\phi)$ is the **likelihood**, indicating the probability of observing data x given the latent variable z and parameters ϕ.
- $P(z)$ is the **prior probability**, expressing the initial belief or distribution about the latent variable z before seeing the data x.
- $P(x|\phi)$ is the **marginal likelihood** or **evidence**, which can be obtained by integrating the joint probability $P(x,z|\phi)$ over all possible values of z, i.e., $P(x|\phi) = \int P(x|z,\phi)P(z)dz$. This step effectively marginalizes out z, making it independent of z but dependent on x and ϕ.

But in practice, this is intractable because we can't evaluate the data likelihood in the denominator.

One solution is to make a variational approximation: we choose a simple parametric form for $q(z|\theta)$ and use this to approximate the true posterior. We can choose a multivariate normal distribution with mean μ and diagonal covariance Σ. This will not always match the posterior well but often performs well. During training, we will find the normal distribution that is "closest" to the true posterior $P(z|x)$.

Since the optimal choice depends on the data example x, the variational approximation should do the same, so we choose:

$$q(z|x,\theta) = \text{Norm}_z\left[g_\mu[x,\theta], g_\Sigma[x,\theta]\right]$$

where $g[x,\theta]$ is a second neural network with parameters θ that predicts the mean μ and variance Σ of the normal variational approximation.

Structure of VAE

According to *Monte Carlo estimate*, For any function $a[\cdot]$ we have:

$$\mathbb{E}_z[a[z]] = \int a[z]q(z|x,\theta)\mathrm{d}z \approx \frac{1}{N}\sum_{n=1}^{N} a[z_n^*]$$

where z_n^* is the n-th sample from $q(z|x,\theta)$.

And thus, we can better know how to calculate ELBO from the perspective of algorithm:

$$\begin{aligned}\text{ELBO} &\approx \frac{1}{N}\sum_{n=1}^{N}\log\left[P(x|z_n^*,\phi)\right] - D_{\text{KL}}\left[q(z|x,\theta)\,||\,P(z)\right] \\ &= \frac{1}{N}\log\left[\prod_{n=1}^{N}P(x|z_n^*,\phi)\right] - D_{\text{KL}}\left[q(z|x,\theta)\,||\,P(z)\right]\end{aligned}$$

Notice that the second term is the KL-divergence between the variational distribution $q(z|x,\theta) = \text{Norm}_z[\mu,\Sigma]$ and the prior $P(z) = \text{Norm}_z[0,I]$. **The KL-divergence between two normal distributions can be calculated in closed form**. It is given by:

$$D_{\text{KL}}\left[q(z|x,\theta)\,||\,P(z)\right] = \frac{1}{2}\left(\text{tr}[\Sigma] + \mu^T\mu - D_z - \log\left[\det[\Sigma]\right]\right)$$

where D_z is the dimensionality of the latent space. So we have the loss function:

$$\text{ELBO} \approx \frac{1}{N}\sum_{n=1}^{N}\log\left[P(x|z_n^*,\phi)\right] - \frac{1}{2}\left(\text{tr}[\Sigma] + \mu^T\mu - D_z - \log\left[\det[\Sigma]\right]\right) \quad (*)$$

and our objective is:

$$\arg\max_{\phi,\theta}\frac{1}{N}\sum_{n=1}^{N}\log\left[P(x|z_n^*,\phi)\right] - D_{\text{KL}}\left[q(z|x,\theta)\,||\,P(z)\right] \quad (**)$$

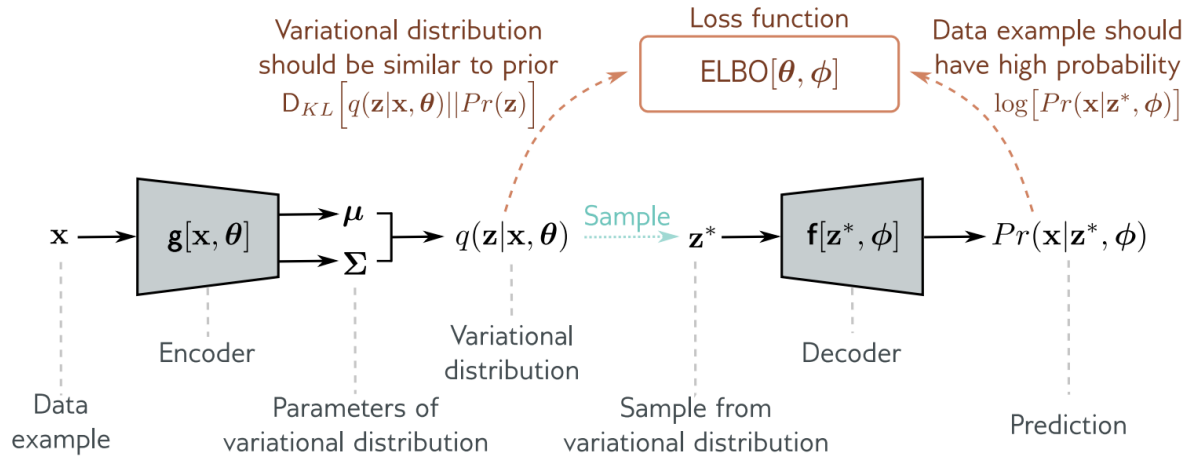


Figure 17.9 Variational autoencoder. The encoder $\mathbf{g}[\mathbf{x},\boldsymbol{\theta}]$ takes a training example $\mathbf{x}$ and predicts the parameters $\boldsymbol{\mu},\boldsymbol{\Sigma}$ of the variational distribution $q(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})$. We sample from this distribution and then use the decoder $\mathbf{f}[\mathbf{z},\boldsymbol{\phi}]$ to predict the data $\mathbf{x}$. The loss function is the negative ELBO, which depends on how accurate this prediction is and how similar the variational distribution $q(\mathbf{z}|\mathbf{x},\boldsymbol{\theta})$ is to the prior $Pr(\mathbf{z})$ (equation 17.21).

It should now be clear why this is called a *variational autoencoder*. It is variational because it computes a Gaussian approximation to the posterior distribution. It is an autoencoder because it starts with a data point x, computes a lower-dimensional latent vector z from this, and then uses it to recreate the data point x as closely as possible. In this context, the mapping from the data to the latent variable by the network $g[x,\theta]$ is called the *encoder*, and the mapping from the latent variable to the data by the network $f[x,\theta]$ is called the *decoder*.

$$\begin{aligned}
\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\frac{p(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] && \text{(Chain Rule of Probability)}\\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\frac{p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] && \text{(Split the Expectation)}\\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\parallel p(\boldsymbol{z}))}_{\text{prior matching term}} && \text{(Definition of KL Divergence)}
\end{aligned}$$

Policy Gradient (×)

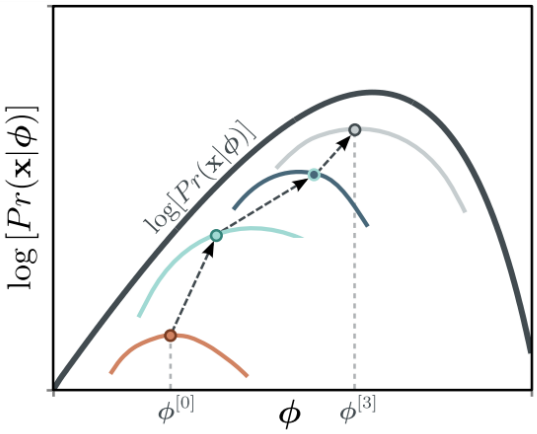
The Reparameterization Trick

The reparameterization trick rewrites a random variable as a deterministic function of a noise variable; this allows for the optimization of the non-stochastic terms through gradient descent. For example, samples from a normal distribution $x \sim \mathcal{N}(x;\mu,\sigma^2)$ with arbitrary mean μ and variance σ^2 can be rewritten as:

$$x = \mu + \sigma\epsilon \quad \text{with } \epsilon \sim \mathcal{N}(\epsilon;0,I)$$

By the reparameterization trick, sampling from an arbitrary Gaussian distribution can be performed by sampling from a standard Gaussian, scaling the result by the target standard deviation, and shifting it by the target mean.

Figure 17.10 The VAE updates both factors that determine the lower bound at each iteration. Both the parameters ϕ of the decoder and the parameters θ of the encoder are manipulated to increase this lower bound.



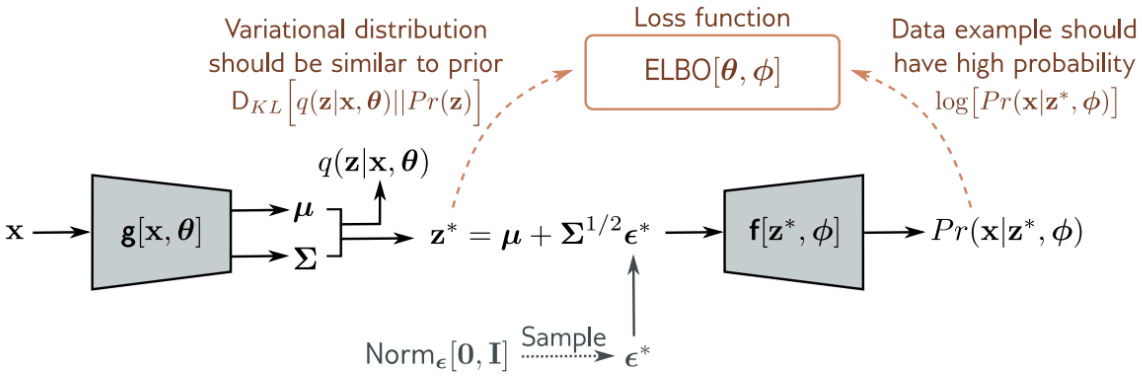


Figure 17.11 Reparameterization trick. With the original architecture (figure 17.9), we cannot easily backpropagate through the sampling step. The reparameterization trick removes the sampling step from the main pipeline; we draw from a standard normal and combine this with the predicted mean and covariance to get a sample from the variational distribution.

AEs & VAE variants (×)

> 在深度学习的早期，自动编码器（Autoencoder, AE）被广泛用于无监督学习中，尤其是用于降维和特征学习。自动编码器的工作机制是通过一个编码器将输入数据压缩为一个潜在表示，再通过解码器从该潜在表示重建原始数据。虽然自动编码器在重构数据时有一定的能力，但它并不擅长生成新的数据点。这是因为传统的自动编码器将输入数据映射到一个**固定的潜在空间**，没有明确地对潜在空间的分布进行建模。
>
> 所以从结果上来讲，AE的泛化能。

- Vanilla Autoencoder
- Denoising Autoencoder
- Sparse Autoencoder
- Contractive Autoencoder
- Stacked Autoencoder
- Variational Autoencoder (VAE)

- Convolutional VAE
- Conditional VAE
- β-VAE
- IWAE

Representation learning

- Ladder VAE
- Progressive + Fade-in VAE

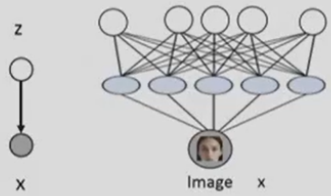
Hierarchical representation learning

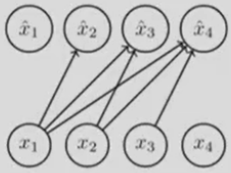
- VAE in speech
- Temporal Difference VAE (TD-VAE)

Temporal representation learning

VAE VS. ARM

VAE vs. Autoregressive models





- Model $p(x) = \int p(x, z)dz$
- Learning: ELBO, approx. MLE
- Inference:
  - Sampling: decoder, fast
  - Likelihood: approximate (affect MLE learning)
  - Feature: encoder, fast

- Model: $p(x) = \prod p(x_i|x_{<i})$
- Learning: MLE
- Inference:
  - Sampling: sequential
  - Likelihood: exact and fast (affect MLE learning)
  - Feature: no

Where we are?

北京大学 PEKING UNIVERSITY

- Autoregressive Models
  $p(x_1, x_2, \dots, x_n) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \dots p(x_n|x_1, \dots, x_{n-1})$
  - Provide tractable likelihoods
  - No direct mechanism for learning features
  - Slow generation – Wavenet: 1 second audio takes 90 mins (200K samples)

- Variational Autoencoders
  $p(X) = \sum_Z p(X|Z)p(Z)$ or $p(X) = \int_Z p(X|Z)p(Z)dZ$
  - Can learn feature representations (via latent variables Z)
  - Have intractable marginal likelihoods.
  - Optimising a lower bound – it is not maximising the likelihood … we don't know the gap.

Question: Can we design a latent variable model with tractable likelihoods?
Yes! We can use normalising flow models. (Today)

Markovian Hierarchical VAE

A Hierarchical Variational Autoencoder (HVAE) is a generalization of a VAE that extends to multiple hierarchies over latent variables.

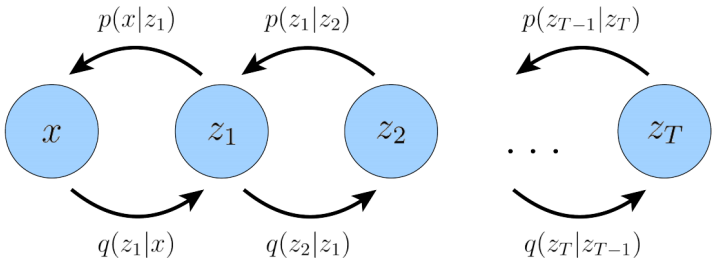


Figure 2: A Markovian Hierarchical Variational Autoencoder with T hierarchical latents. The generative process is modeled as a Markov chain, where each latent $\boldsymbol{z}_t$ is generated only from the previous latent $\boldsymbol{z}_{t+1}$.

Intuitively, and visually, this can be seen as simply stacking VAEs on top of each other, as depicted in Figure 2; another appropriate term describing this model is a Recursive VAE. Mathematically, we represent the joint distribution and the posterior of a Markovian HVAE as:

$$p(x, z_{1:T}) = p(z_T)p_\theta(x|z_1)\prod_{t=2}^{T} p_\theta(z_{t-1}|z_t)$$

$$q_\phi(z_{1:T}|x) = q_\phi(z_1|x)\prod_{t=2}^{T} q_\phi(z_t|z_{t-1})$$

And the ELBO can be extended to be:

$$\begin{aligned}
\log p(x) &= \log \int p(x, z_{1:T})\mathrm{d}z_{1:T} \\
&= \log \int \frac{p(x, z_{1:T})q_\phi(z_{1:T})}{q_\phi(z_{1:T})}\mathrm{d}z_{1:T} \\
&= \log \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\frac{p(x, z_{1:T})}{q_\phi(z_{1:T}|x)}\right] \\
&\geq \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log \frac{p(x, z_{1:T})}{q_\phi(z_{1:T}|x)}\right] \\
&= \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log \frac{p(z_T)p_\theta(x|z_1)\prod_{t=2}^{T} p_\theta(z_{t-1}|z_t)}{q_\phi(z_1|x)\prod_{t=2}^{T} q_\phi(z_t|z_{t-1})}\right]
\end{aligned}$$

3 - Normalizing Flow Model (×)

4 - Generative Adversarial Networks (GAN)

High likelihood = good sample quality?

Case 1: Optimal generative model will give best sample quality andhighest test log-likelihood

For imperfect models, achieving high log-likelihoods might not alwaysimply good sample quality, and vice-versa (Theis et al., 2016)

- **Case 2:** Great test log-likelihoods, poor samples. E.g., For a discrete noise mixture model $p_\theta(\mathbf{x}) = 0.01 p_{\text{data}}(\mathbf{x}) + 0.99 p_{\text{noise}}(\mathbf{x})$
  - 99% of the samples are just noise
  - Taking logs, we get a lower bound

$$\begin{aligned}\log p_\theta(\mathbf{x}) &= \log[0.01 p_{\text{data}}(\mathbf{x}) + 0.99 p_{\text{noise}}(\mathbf{x})] \\ &\geq \log 0.01 p_{\text{data}}(\mathbf{x}) = \log p_{\text{data}}(\mathbf{x}) - \log 100\end{aligned}$$

  - For expected likelihoods, we know that
    - Lower bound

$$E_{p_{\text{data}}}[\log p_\theta(\mathbf{x})] \geq E_{p_{\text{data}}}[\log p_{\text{data}}(\mathbf{x})] - \log 100$$

    - Upper bound (via non-negativity of KL)

$$E_{p_{\text{data}}}[\log p_{\text{data}}(\mathbf{x}))] \geq E_{p_{\text{data}}}[\log p_\theta(\mathbf{x})]$$

  - As we increase the dimension of $\mathbf{x}$, absolute value of $\log p_{\text{data}}(\mathbf{x})$ increases proportionally but $\log 100$ remains constant. Hence, $E_{p_{\text{data}}}[\log p_\theta(\mathbf{x})] \approx E_{p_{\text{data}}}[\log p_{\text{data}}(\mathbf{x})]$ in very high dimensions

- **Case 3:** Great samples, poor test log-likelihoods. E.g., Memorizing training set
  - Samples look exactly like the training set (cannot do better!)
  - Test set will have zero probability assigned (cannot do worse!)
- The above cases suggest that it might be useful to disentangle likelihoods and samples
- **Likelihood-free learning** consider objectives that do not depend directly on a likelihood function

Likelihood is not a gold metric for everything!
We can learn a family of powerful models without MLE!

Structure of GAN

Loss Function

GAN's architecture consists of two neural networks:

1. **Generator(G)**: creates synthetic data from random noise to produce data so realistic that the discriminator cannot distinguish it from real data.

   The Generator is a deep neural network (DNN) that takes random noise as input to generate realistic data samples, learning the underlying data distribution by adjusting its parameters through backpropagation (BP). Its loss function is:

$$\begin{aligned}\mathcal{L}_G &= \mathbb{E}_{x \sim p_z}[\log(1 - D(G(z)))] \\ G^* &= \min_G \mathbb{E}_{x \sim p_z}[\log(1 - D(G(z)))]\end{aligned}$$

   Where,

   - $\mathcal{L}_G$ measure how well the generator is fooling the discriminator.
   - z is random noise sampled from the noise distribution p_z.

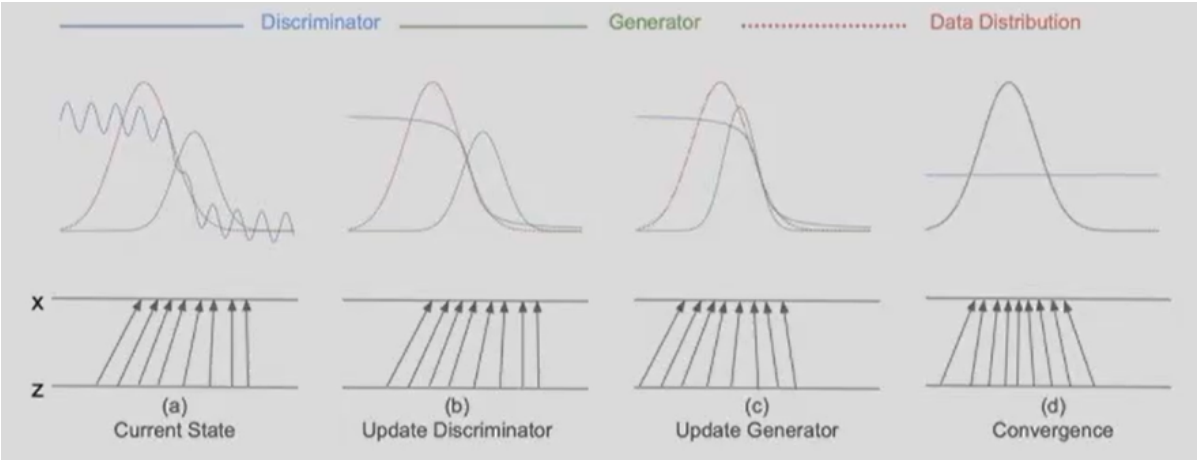
2. **Discriminator(D)**: acts as a critic, evaluating whether the data it receives is real or fake.

   The Discriminator acts as a binary classifier, distinguishing between real and generated data. This loss incentivizes the discriminator to accurately categorize generated samples as fake and real samples with the following equation:

$$\mathcal{L}_D = -\mathbb{E}_{x\sim p_{data}}[\log D(x)] - \mathbb{E}_{x\sim p_z}[\log(1 - D(G(x)))]$$
$$D^* = \max_D \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim p_z}[\log(1 - D(G(x)))]$$

   Where,

   - $\mathcal{L}_D$ assesses the discriminator's ability to discern between produced and actual samples.
   - x is the real data sample from the distribution p_{data}.



The training process of a GAN (Generative Adversarial Network) is a minimax game, where the goals of the generator and the discriminator are opposed:

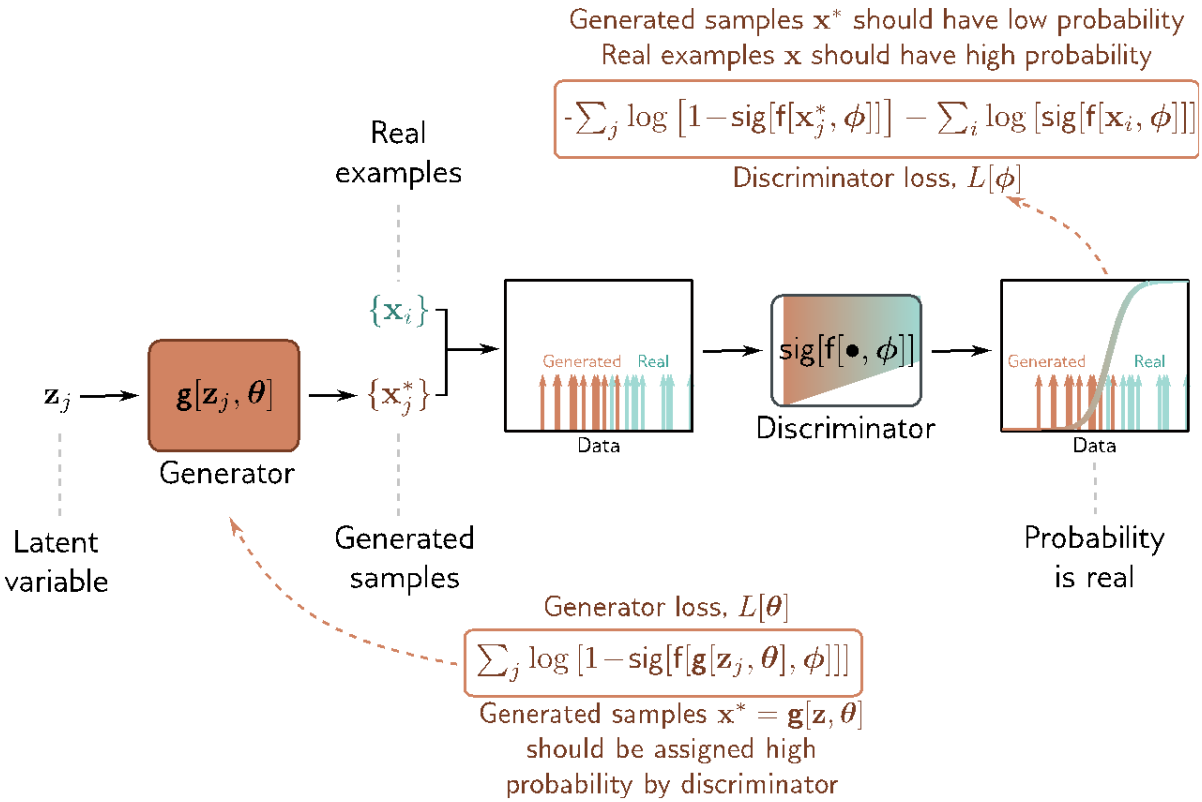
$$\min_G \max_D V(D,G) = \min_G \max_D \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{x\sim p_z}[\log(1 - D(G(x)))]$$

Optimal disciminator:

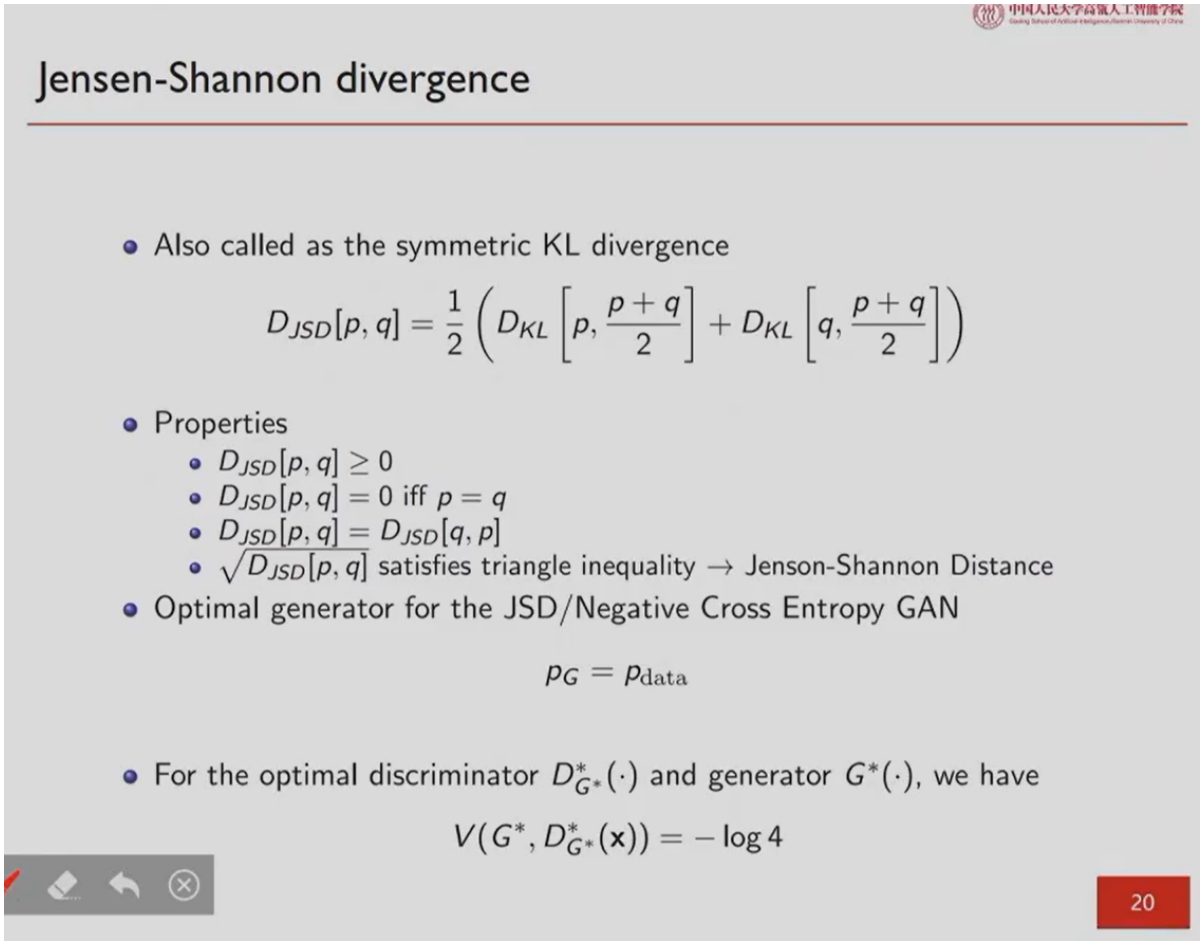
$$D_G^* = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

For the optimal disciminator $D_G^*(\cdot)$, we have

$$\begin{aligned}
V(G, D_G^*(x)) &= \mathbb{E}_{x\sim p_{data}}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}\right] + \mathbb{E}_{x\sim p_z}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}\right] \\
&= \mathbb{E}_{x\sim p_{data}}\left[\log \frac{p_{data}(x)}{\frac{p_{data}(x)+p_G(x)}{2}}\right] + \mathbb{E}_{x\sim p_z}\left[\log \frac{p_{data}(x)}{\frac{p_{data}(x)+p_G(x)}{2}}\right] - \log 4 \\
&= D_{\mathrm{KL}}\left[p_{data}, \frac{p_{data}(x) + p_G(x)}{2}\right] + D_{\mathrm{KL}}\left[p_G, \frac{p_{data}(x) + p_G(x)}{2}\right] - \log 4 \\
&= 2D_{\mathrm{JSD}}\left[p_{data}, p_G\right] - \log 4
\end{aligned}$$



Jesen-Shannon divergence



Evaluation

GANs are assessed based on the ***quality, diversity, and realism*** of their generated samples. Common evaluation metrics include:

Fréchet Inception Distance (FID) – Measures similarity to real images.

Inception Score (IS) – Evaluates the variety and clarity of generated images.

Evaluation of Generative Models: Sample Quality • Known Ground Truth • SRGAN •-MSE PSNR SSIM • Unknown Ground Truth • DRIT • StarGAN •-Classification •-IS FID KID •-LPIPS • Human Evaluation • Ranking v.s. Contrast • Tools: AMT

Lecture 16-17 Evaluation - Sampling Quality 50-80min

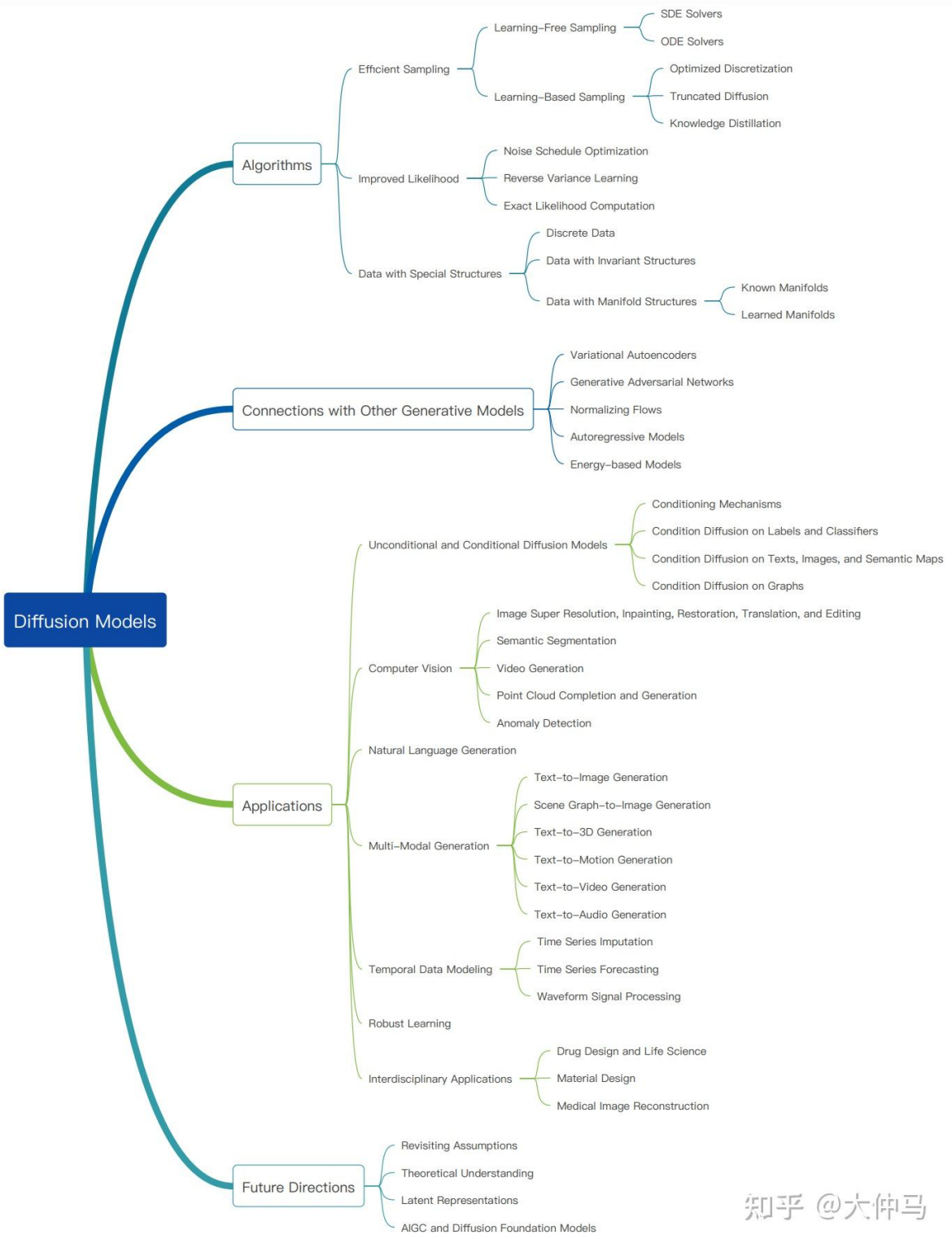
Optimization challenges (略)

- **Training instability** – Balancing the generator and discriminator is tricky.
- **Mode collapse** – The generator may produce only a few types of outputs instead of diverse samples.
- **High computational cost** – Requires powerful hardware for training.

Further Variants (×)

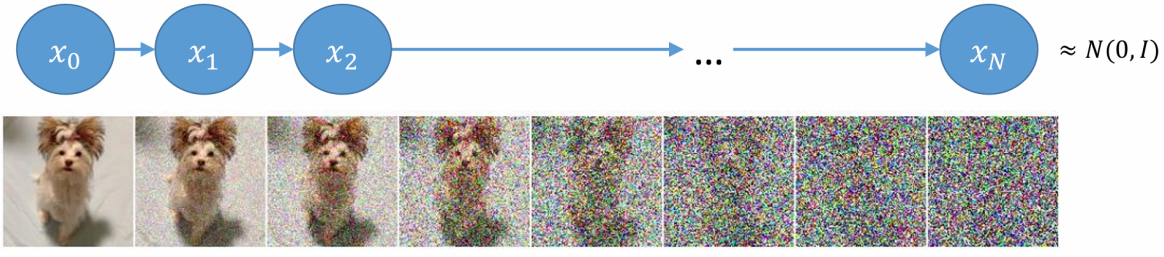
5 - Energy-Based Model (EBM) (×)

6 - Diffusion Probabilistic Model (DPM)



An Overview From Markov Chain

Transition of diffusion: $q(x_n|x_{n-1}) = N(\sqrt{\alpha_n}x_{n-1}, \beta_n I) \quad \alpha_n = 1 - \beta_n$



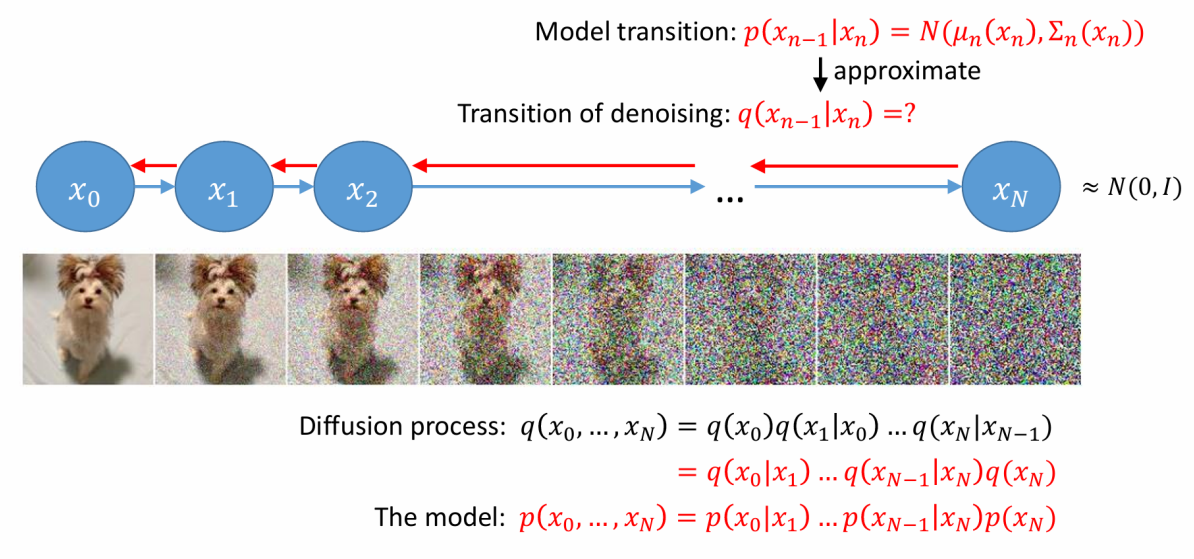
Diffusion process: $q(x_0, \dots, x_N) = q(x_0)q(x_1|x_0) \dots q(x_N|x_{N-1})$

Demo Images from *Song et al. Score-based generative modeling through stochastic differential equations, ICLR 2021.*

Diffusion process gradually injects noise to data

Described by a Markov chain:

$$q(x_0, \cdots, x_N) = q(x_0)q(x_1|x_0) \cdots q(x_N|x_{N-1})$$



Diffusion process in the reverse direction ⇔ denoising proces

Reverse factorization (We need an approximate diffusion process in the reverse direction):

$$q(x_0, \cdots, x_N) = q(x_0|x_1) \cdots q(x_{N-1}|x_N)q(x_N)$$

> 1. 从右向左是encoder过程，是一个无参数的 $q(x_t|x_{t-1})$，即这是一个纯粹人为的过程（比如从原始的清晰图片每次按照高斯分布进行一个映射）。
> 2. 从左往右是decoder过程，是一个带参数的 $p_\theta(x_{t-1}|x_t)$ 。其并非像VAE那样直接预测 $\hat{x}$ ，而是预测高斯噪声，在decode过程中逐渐减去高斯噪声，还原出清晰的图像。
> 3. 在diffusion model 中隐变量 z （在图中是 $x_N(\text{or usually } x_T)$ ）和原始图片的维度是一样大的（但是理论上还是简单的（因为很像高斯噪声））。

Let's take a recap of two formulation of MHVAE:

> $$p(x, z_{1:T}) = p(z_T)p_\theta(x|z_1)\prod_{t=2}^{T} p_\theta(z_{t-1}|z_t)$$
>
> $$q_\phi(z_{1:T}|x) = q_\phi(z_1|x)\prod_{t=2}^{T} q_\phi(z_t|z_{t-1})$$

The VDM posterior is the same as the MHVAE posterior, but can now be rewritten as:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

with the assumption of:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I)$$

this assumption can be rewritten into $x_t = \alpha_t x_{t-1} + \beta_t \varepsilon_t, \varepsilon_t \in \mathcal{N}(0, I), \alpha_t, \beta_t > 0 \wedge \alpha_t^2 + \beta_t^2 = 1$, with β_t close to o. In the case,

$$
\begin{aligned}
x_t &= \alpha_t x_{t-1} + \beta_t \varepsilon_t \\
&= \alpha_t(\alpha_{t-1} x_{t-2} + \beta_{t-1}\varepsilon_{t-1}) + \beta_t \varepsilon_t \\
&= \cdots \\
&= (\alpha_t \cdots \alpha_1) x_0 + (\alpha_t \cdots \alpha_2)\beta_1 \varepsilon_1 + (\alpha_t \cdots \alpha_3)\beta_2 \varepsilon_2 + \cdots + \alpha_t \beta_{t-1}\varepsilon_{t-1} + \beta_t \varepsilon_t \\
\text{key!} &= (\alpha_t \cdots \alpha_1) x_0 + \sqrt{1 - (\alpha_t \cdots \alpha_1)^2}\bar{\varepsilon}_t, \bar{\varepsilon}_t \sim \mathcal{N}(0, I)
\end{aligned}
$$

And,

$$
\begin{gathered}
p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \\
\text{where, } p(x_T) = \mathcal{N}(x_T; 0, I)
\end{gathered}
$$

This means that, our encoder distributions $q(x_t|x_{t-1})$ are no longer parameterized by ϕ, as they are completely modeled as Gaussians with defined mean and variance parameters at each timestep. Therefore, in a VDM, we are only interested in learning conditionals $p_\theta(x_{t-1}|x_t)$. After optimizing the VDM, the sampling procedure is as simple as sampling Gaussian noise from $p(x_T)$ and iteratively running the denoising transitions for T steps to generate a novel x_0.

ELBO can be derived as:

$$
\begin{aligned}
\log p(x) &= \log \int p(x_{0:T}) \mathrm{d}x_{1:T} \\
&= \log \int \frac{p(x_{0:T}) q(x_{1:T}|x_0)}{q(x_{1:T}|x_0)} \mathrm{d}x_{1:T} \\
&= \log \mathbb{E}_{q(x_{1:T}|x_0)} \left[\frac{p(x_{0:T})}{q(x_{1:T}|x_0)} \right] \\
&\geq \mathbb{E}_{q(x_{1:T}|x_0)} \log \left[\frac{p(x_{0:T})}{q(x_{1:T}|x_0)} \right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)} \log \left[\frac{p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)}{\prod_{t=1}^{T} q(x_t|x_{t-1})} \right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)} \log \left[\frac{p(x_T) p_\theta(x_0|x_1) \prod_{t=1}^{T-1} p_\theta(x_t|x_{t+1})}{q(x_T|x_{T-1}) \prod_{t=1}^{T-1} q(x_t|x_{t-1})} \right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)} \log \left[\frac{p(x_T) p_\theta(x_0|x_1)}{q(x_T|x_{T-1})} \right] + \mathbb{E}_{q(x_{1:T}|x_0)} \log \left[\frac{\prod_{t=1}^{T-1} p_\theta(x_t|x_{t+1})}{\prod_{t=1}^{T-1} q(x_t|x_{t-1})} \right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)} \log \left[p_\theta(x_0|x_1) \right] + \mathbb{E}_{q(x_{1:T}|x_0)} \log \left[\frac{p(x_T)}{q(x_T|x_{T-1})} \right] + \mathbb{E}_{q(x_{1:T}|x_0)} \sum_{t=1}^{T-1} \log \left[\frac{p_\theta(x_t|x_{t+1})}{q(x_t|x_{t-1})} \right]
\end{aligned}
$$

$$\log p(\boldsymbol{x}) = \log \int p(\boldsymbol{x}_{0:T}) d\boldsymbol{x}_{1:T} \tag{34}$$

$$= \log \int \frac{p(\boldsymbol{x}_{0:T}) q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} d\boldsymbol{x}_{1:T} \tag{35}$$

$$= \log \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\frac{p(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \right] \tag{36}$$

$$\geq \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \right] \tag{37}$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{\prod_{t=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \tag{38}$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1}) \prod_{t=1}^{T-1} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \tag{39}$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \prod_{t=1}^{T-1} p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1}) \prod_{t=1}^{T-1} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \tag{40}$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1})} \right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \prod_{t=1}^{T-1} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \tag{41}$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1})} \right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\sum_{t=1}^{T-1} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \tag{42}$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1})} \right] + \sum_{t=1}^{T-1} \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \tag{43}$$

$$= \mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \right] + \mathbb{E}_{q(\boldsymbol{x}_{T-1}, \boldsymbol{x}_T|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1})} \right] + \sum_{t=1}^{T-1} \mathbb{E}_{q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{x}_{t+1}|\boldsymbol{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \tag{44}$$

$$= \underbrace{\mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)} \left[\log p_{\theta}(\boldsymbol{x}_0|\boldsymbol{x}_1) \right]}_{\text{reconstruction term}} - \underbrace{\mathbb{E}_{q(\boldsymbol{x}_{T-1}|\boldsymbol{x}_0)} \left[D_{\mathrm{KL}}(q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1}) \parallel p(\boldsymbol{x}_T)) \right]}_{\text{prior matching term}} \tag{45}$$

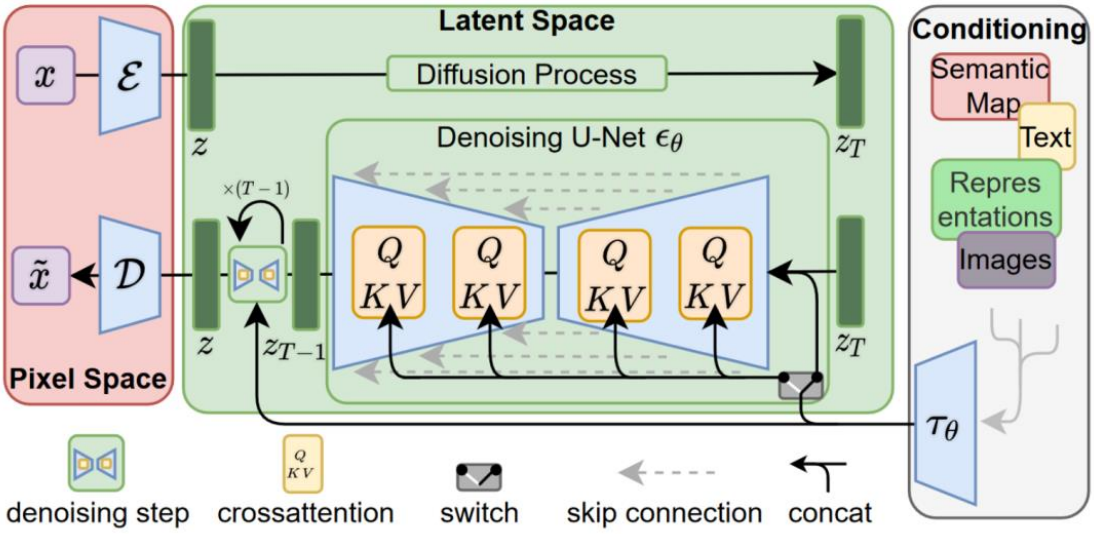
$$- \sum_{t=1}^{T-1} \underbrace{\mathbb{E}_{q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_{t+1}|\boldsymbol{x}_0)} \left[D_{\mathrm{KL}}(q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \parallel p_{\theta}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})) \right]}_{\text{consistency term}}$$

Encoder (Forward Process of DDPM)

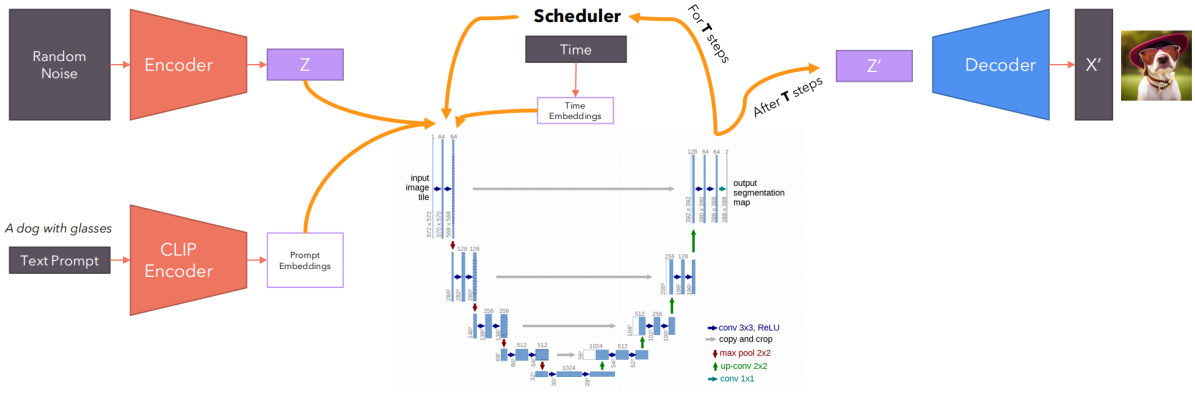
Decoder (Backward Process of DDPM)

DDIM

Architecture of Stable Diffusion (credit to *Umar Jamil*, Coding Stable Diffusion form scratch in PyTorch)



Architecture (Text-To-Image)



7 - Score-based Model

NCSN

目标函数： $L=\frac{1}{2}\mathbb{E}_{P_{data}(x)}\left[||s_\theta(x)-\nabla_x\log P_{data}(x)||_2^2\right]$

加噪： $L=\frac{1}{2}\mathbb{E}_{P_{data}(x),\tilde{x}\sim N(x,\sigma^2 I)}\left[||s_\theta(x+\sigma z)-\nabla_{\tilde{x}}\log q(\tilde{x}|x)||_2^2\right]$

加不同级别的噪声： $L=\frac{1}{S}\sum_{i=1}^{S}\lambda_i\frac{1}{2}\mathbb{E}_{P_{data}(x),\tilde{x}\sim N(x,\sigma_i^2 I)}\left[||s_\theta(x+\sigma_i z,\sigma_i)+\frac{\tilde{x}_i-x}{\sigma_i^2}||_2^2\right]$

同时在NCSN里面，我们曾得到$\frac{\tilde{x}_i-x}{\sigma_i^2}=\frac{z}{\sigma_i}$，于是便有

$$L=\frac{1}{S}\sum_{i=1}^{S}\lambda_i\frac{1}{2}\mathbb{E}_{P_{data}(x),z\sim N(0,I)}\left[||s_\theta(x+\sigma_i z,\sigma_i)+\frac{z}{\sigma_i}||_2^2\right]$$

此时s_θ预测的其实就是$-\frac{z}{\sigma_i}$

采样： $x_{t+1}=x_t+\alpha\nabla_x\log P(x_t)+\sqrt{2\alpha}z_t$

DDPM

加噪过程表示为$q(x_i|x_{i-1})=\mathcal{N}(x_i|\sqrt{1-\beta_i}x_{i-1},\beta_i I)$

跳步加噪表示为$q(x_i|x_0)=\mathcal{N}(x_i|\sqrt{\alpha_i}x_0,(1-\alpha_i)I)$

请注意，在DDPM里面，其实应该是$q(x_i|x_0)=\mathcal{N}(x_i|\sqrt{\bar{\alpha}_i}x_0,(1-\bar{\alpha}_i)I)$。

去噪过程表示为$P(x_{i-1}|x_i)$。重参数化得到采样生成步骤（论文将该方法称为祖先采样）

$$x_{i-1}=\frac{1}{\sqrt{1-\beta_i}}\left(x_i-\frac{\beta_i}{\sqrt{1-\alpha_i}}\epsilon_\theta(x_i,i)\right)+\sigma_i z$$

$$L=||\epsilon_i-\epsilon_\theta(x_i,i)||^2$$

DDPM和NCSN的统一

NCSN

$$L=\frac{1}{S}\sum_{i=1}^{S}\lambda_i\frac{1}{2}\mathbb{E}_{P_{data}(x),z\sim N(0,I)}\left[||s_\theta(x+\sigma_i z,\sigma_i)+\frac{z}{\sigma_i}||_2^2\right]$$

此时s_θ预测的其实就是$-\frac{z}{\sigma_i}$

$q_\sigma(\tilde{x}|x)$表示从初始图像x加噪到$\tilde{x}$，可以写成与DDPM的形式$q(x_i|x_0)$，也就是从x_0加噪到x_i

---

$$x_{i-1}=\frac{1}{\sqrt{1-\beta_i}}\left(x_i-\frac{\beta_i}{\sqrt{1-\alpha_i}}\epsilon_\theta(x_i,i)\right)+\sigma_i z$$

论文将$q(x_i|x_0)$统称为扰动核

我们可以令$-\frac{\epsilon_\theta(x_i,i)}{\sqrt{1-\alpha_i}}=s_\theta(x_i,i)$

DDPM

$$x_{i-1}=\frac{1}{\sqrt{1-\beta_i}}(x_i+\beta_i s_\theta(x_i,i))+\sqrt{\beta_i}z$$

? - Reference

Understanding Deep Learning by Simon J.D. Prince

Generative Adversarial Network (GAN) - GeeksforGeeks

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

Diffusion Probabilistic Models: Theory and Applications - Fan Bao

扩散模型(Diffusion Model)首篇综述-Diffusion Models: A Comprehensive Survey of Methods and Applications - 知乎

苏剑林. (Jun. 13, 2022). 《生成扩散模型漫谈（一）：DDPM = 拆楼 + 建楼》[Blog post]. Retrieved from https://kexue.fm/archives/9119