Course Materials for GEN-AI

Northeastern University

These materials have been prepared and sourced for the course **GEN-AI** at Northeastern University. Every effort has been made to provide proper citations and credit for all referenced works.

If you believe any material has been inadequately cited or requires correction, please contact me at:

Instructor: Ramin Mohammadi
r.mohammadi@northeastern.edu

Thank you for your understanding and collaboration.

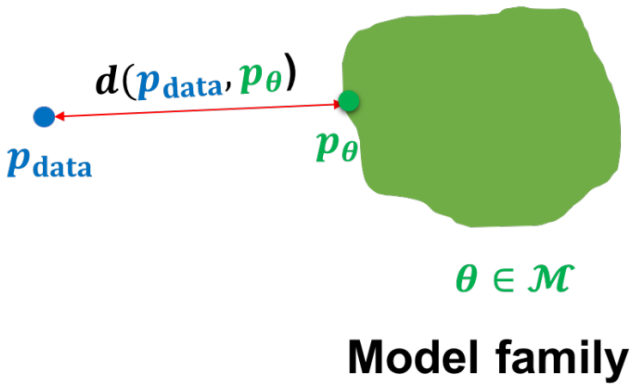
Content sourced from multiple references - see Reference section for details

Diffusion Models

1 Model Family:

$\mathbf{x}^{(j)} \sim p_{\text{data}}$

$j = 1, 2, \ldots, |\mathcal{D}|$



Generative models can be broadly classified based on how they model probability densities or mass functions. Here are the main types:

1. Probability Density / Mass Functions:

- **Autoregressive Models:** Autoregressive models decompose the joint probability of data $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ as a product of conditional probabilities:

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{d} p_\theta(x_i \mid x_{<i}),$$

  where $x_{<i} = (x_1, x_2, \ldots, x_{i-1})$. Examples include PixelCNN and WaveNet.

- **Normalizing Flow Models:** Normalizing flows transform a simple base distribution $p_Z(\mathbf{z})$ (e.g., Gaussian) into a complex data distribution $p_X(\mathbf{x})$ through a sequence of invertible transformations:

$$p_\theta(\mathbf{x}) = p_Z\left(f_\theta^{-1}(\mathbf{x})\right) \left|\det\left(\frac{\partial f_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|,$$

  where f_θ is an invertible mapping parameterized by θ.

- **Latent Variable Models (e.g., Variational Autoencoders):** Latent variable models introduce a set of latent variables $\mathbf{z}$ to model complex data distributions:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})\, d\mathbf{z}.$$

Variational Autoencoders (VAEs) optimize a variational bound on the log-likelihood.

- **Energy-Based Models (EBMs):** EBMs define the data distribution in terms of an energy function $f_\theta(\mathbf{x})$:

$$p_\theta(\mathbf{x}) = \frac{\exp(f_\theta(\mathbf{x}))}{Z(\theta)},$$

where $Z(\theta) = \int \exp(-E_\theta(\mathbf{x}))d\mathbf{x}$ is the partition function that normalizes the distribution.

All the above families are typically trained by minimizing the KL divergence between the data distribution $p_{\text{data}}(\mathbf{x})$ and the model distribution $p_\theta(\mathbf{x})$:

$$D_{\text{KL}}(p_{\text{data}} \| p_\theta) = \int p_{\text{data}}(\mathbf{x}) \log \frac{p_{\text{data}}(\mathbf{x})}{p_\theta(\mathbf{x})} d\mathbf{x}.$$

Minimizing D_{KL} is equivalent to maximizing the likelihood of the data under the model.

2. Sample Generation Processes:

In practice, samples from generative models can be obtained through processes such as:

- **Generative Adversarial Networks (GANs):** GANs generate samples using a generator G that maps a noise variable $\mathbf{z} \sim p(\mathbf{z})$ to the data space. The discriminator D distinguishes between real data and generated samples. The objective function is:

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

where D and G play a minimax game to approximate the data distribution.

- **Two-Sample Tests:** These approaches can approximately optimize f-divergences (e.g., Jensen-Shannon, Kullback-Leibler) or Wasserstein distances to learn the generative model. Examples include methods based on Maximum Mean Discrepancy (MMD) or Sinkhorn distances.

Samples typically have better quality, however they require adversarial training, they suffer from training instability and mode collapse.

3. Score Functions:

- **Energy Based Score Matching**
  - In energy-based generative models, the probability density $p_\theta(\mathbf{x})$ is parameterized through its gradient, known as the **score function**:

$$s_\theta \approx \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}).$$

  - These models use score matching techniques to estimate the score function, which can then be used in **Langevin Dynamics** or other sampling methods to generate samples.

$$= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})\|_2^2 + \text{tr}\left(\nabla_{\mathbf{x}}^2 \log p_\theta(\mathbf{x})\right) \right] + \text{const.}$$

$$= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} f_\theta(\mathbf{x})\|_2^2 + \text{tr}\left(\nabla_{\mathbf{x}}^2 f_\theta(\mathbf{x})\right) \right] + \text{const.}$$

  - **Very flexible model architectures.** But likelihood is intractable, calculating the hessian is expensive.

- **Score Matching**
  - In score-based generative models, the probability density $p_{data}(\mathbf{x})$ is parameterized through its gradient, known as the **score function**:

$$s_\theta \approx \nabla_{\mathbf{x}} \log p_{data}(\mathbf{x}).$$

– These models use score matching techniques to estimate the score function, which can then be used in **Langevin Dynamics** or other sampling methods to generate samples.

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\|\nabla_{\mathbf{x}}\log p_{\text{data}}(\mathbf{x}) - s_\theta(\mathbf{x})\|_2^2\right]$$

$$= \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\frac{1}{2}\|s_\theta(\mathbf{x})\|_2^2 + \text{tr}(\nabla_{\mathbf{x}} s_\theta(\mathbf{x}))\right] + \text{const.}$$

– **Very flexible model architectures.** But likelihood is intractable, calculating the trace of Jacobian is expensive.

- **Noise-Perturbed Score Matching**
  - Given a data distribution $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$, the goal of score matching is to estimate the score function $\nabla_{\mathbf{x}}\log p_{\text{data}}(\mathbf{x})$. In noise-perturbed score matching, a perturbation distribution $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$ is applied to the data distribution, resulting in a noise-perturbed distribution $\tilde{\mathbf{x}} \sim q_\sigma(\tilde{\mathbf{x}})$.
  - The loss function for noise-perturbed score matching is expressed as:

$$\frac{1}{2}\mathbb{E}_{\tilde{\mathbf{x}}\sim p_{\text{data}}}\left[\|s_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}}\log q_\sigma(\tilde{\mathbf{x}})\|_2^2\right],$$

  - where $s_\theta(\tilde{\mathbf{x}})$ is the estimated score function. This loss can be rewritten as:

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x}),\tilde{\mathbf{x}}\sim q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}\left[\|s_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}}\log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\|_2^2\right] + \text{const.}$$

  - Assuming Gaussian perturbations, where $\tilde{\mathbf{x}} = \mathbf{x} + \sigma\mathbf{z}$ and $\mathbf{z} \sim \mathcal{N}(0, I)$, the loss simplifies to:

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}},\mathbf{z}\sim\mathcal{N}(0,I)}\left[\left\|s_\theta(\mathbf{x}+\sigma\mathbf{z}) + \frac{\mathbf{z}}{\sigma}\right\|_2^2\right] + \text{const.}$$

  - **Pros and Cons of Noise-Perturbed Score Matching**
  - **Pros:**
    - Much more scalable than standard score matching.
    - Reduces score estimation to a denoising task.
  - **Con:**
    - Estimates the score of noise-perturbed data rather than the true data distribution:

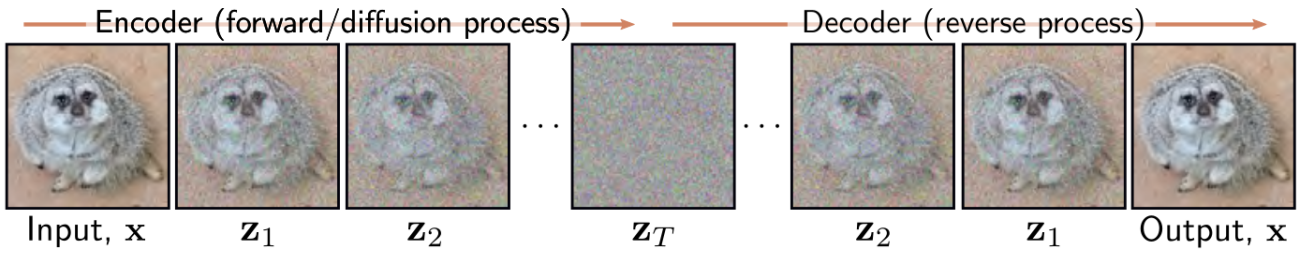
$$s_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}}\log q_\sigma(\mathbf{x}) \neq \nabla_{\mathbf{x}}\log p_{\text{data}}(\mathbf{x}).$$

2 Diffusion Models

2.1 Definition

Diffusion models, similar to normalizing flows, are probabilistic models that establish a nonlinear mapping between latent variables and observed data, where both share the same dimensionality. Like variational autoencoders (VAEs), they approximate the data likelihood using a lower bound, relying on an encoder that maps the data to a latent variable. However, in diffusion models, the encoder is predefined, and the objective is to learn a decoder that reverses this process, enabling the generation of new samples. Diffusion models are straightforward to train and can produce extremely high-quality samples, often surpassing the realism achieved by GANs.

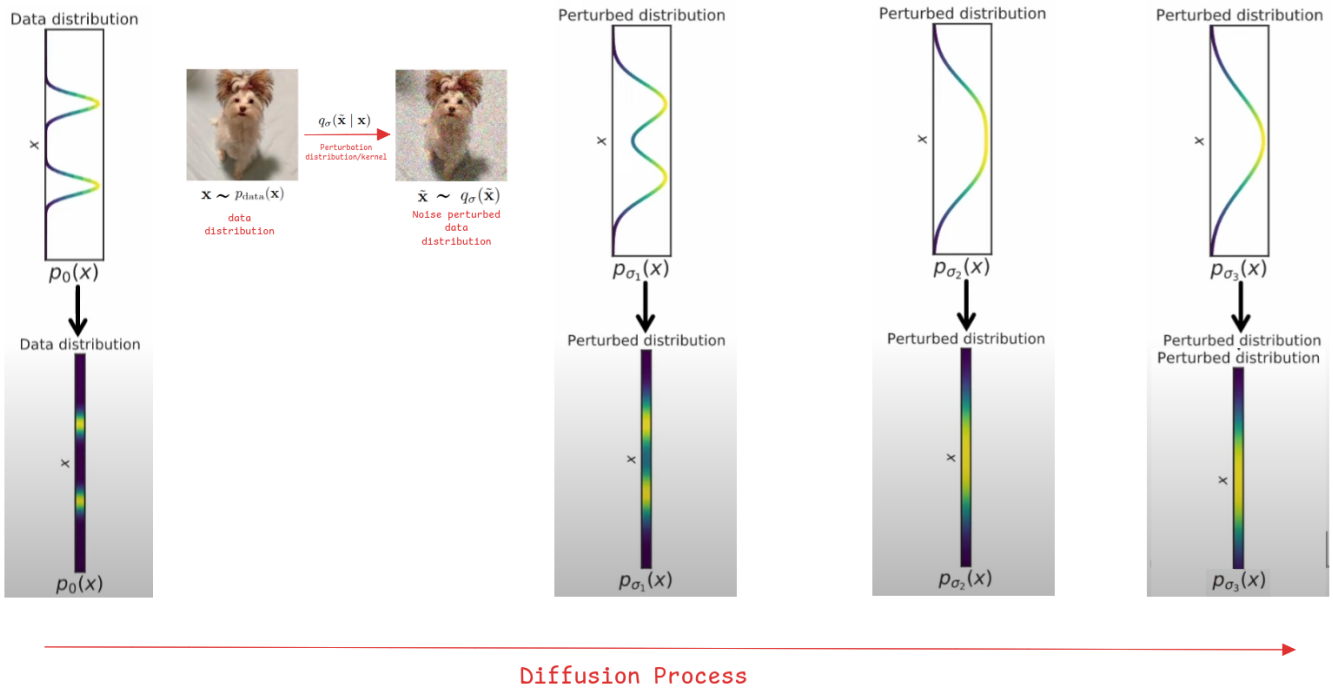
A diffusion model is composed of an encoder and a decoder. The encoder maps a data sample $\mathbf{x}$ into a sequence of intermediate latent variables $\mathbf{z}_1, \ldots, \mathbf{z}_T$. The decoder performs the reverse operation, starting from $\mathbf{z}_T$ and sequentially mapping back through $\mathbf{z}_{T-1}, \ldots, \mathbf{z}_1$ to reconstruct the original data point $\mathbf{x}$. In both the encoder and decoder, these mappings are stochastic rather than deterministic.



The encoder is predefined and progressively mixes the input with white noise (as shown in the figure). After sufficient steps, the conditional distribution $q(\mathbf{z}_T \mid \mathbf{x})$ and the marginal distribution $q(\mathbf{z}_T)$ of the final latent variable approximate the standard normal distribution. Since the encoder process is fixed, all the model parameters are learned in the decoder.

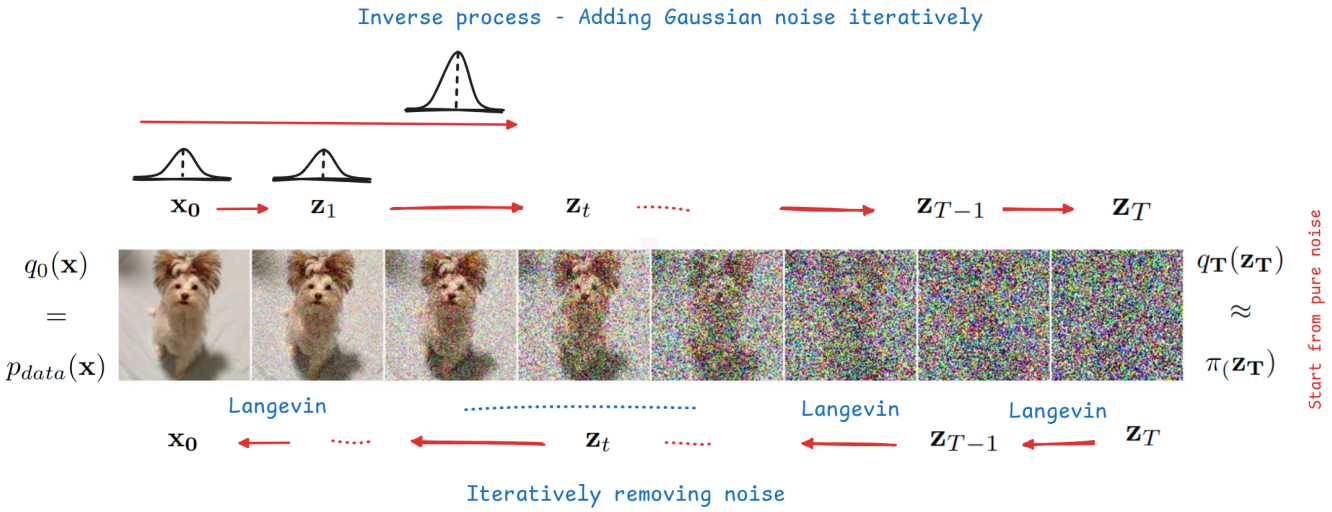
The decoder consists of a series of networks that are trained to reverse each step of the encoder, mapping between adjacent latent variables $\mathbf{z}_t$ and $\mathbf{z}_{t-1}$. The loss function ensures that each network inverts its corresponding encoder step. Consequently, the decoder progressively removes noise from the latent representation until a realistic data sample is produced. To generate a new data sample $\mathbf{x}$, a latent variable is sampled from $q(\mathbf{z}_T)$, and the decoder maps it back to the data space.

2.2 Diffusion Process



An efficient process that destroys the structure of the data transitions from the data distribution to a noise distribution that is simple to sample from. The original data distribution should be continuous (not necessarily Gaussian for images), but the transition kernels should be Gaussian. This ensures that after adding sufficient Gaussian noise to the data distribution, it eventually becomes Gaussian.

3 Forward Diffusion Process - Encoder



The *diffusion process* or *forward process* maps a data sample $\mathbf{x}$ through a sequence of latent variables $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T$, where each variable has the same dimensionality as $\mathbf{x}$. This process is defined as:

$$\mathbf{z}_1 = \sqrt{1-\beta_1}\,\mathbf{x} + \sqrt{\beta_1}\,\boldsymbol{\epsilon}_1$$
$$\mathbf{z}_t = \sqrt{1-\beta_t}\,\mathbf{z}_{T-1} + \sqrt{\beta_t}\,\boldsymbol{\epsilon}_t, \quad \forall t \in \{2, \ldots, T\},$$

where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is Gaussian noise sampled from the standard normal distribution.

In this process:

- The deterministic term $\sqrt{1-\beta_t}$ acts as a scaling factor, progressively attenuating the original data as t increases.
- The stochastic term $\sqrt{\beta_t}\,\boldsymbol{\epsilon}_t$ adds Gaussian noise at each step, blending the data with noise.
- The hyperparameters $\beta_t \in [0, 1]$ control the rate at which the noise is added and are collectively referred to as the *noise schedule*.
- $\mathbf{z}_{t-1}$ is a known vector (a realization from the previous step).
- $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, which is Gaussian noise with zero mean and identity covariance.

Over time, this iterative process gradually transforms the input $\mathbf{x}$ into a pure Gaussian distribution. The forward process can also be equivalently expressed in terms of the cumulative noise schedule.

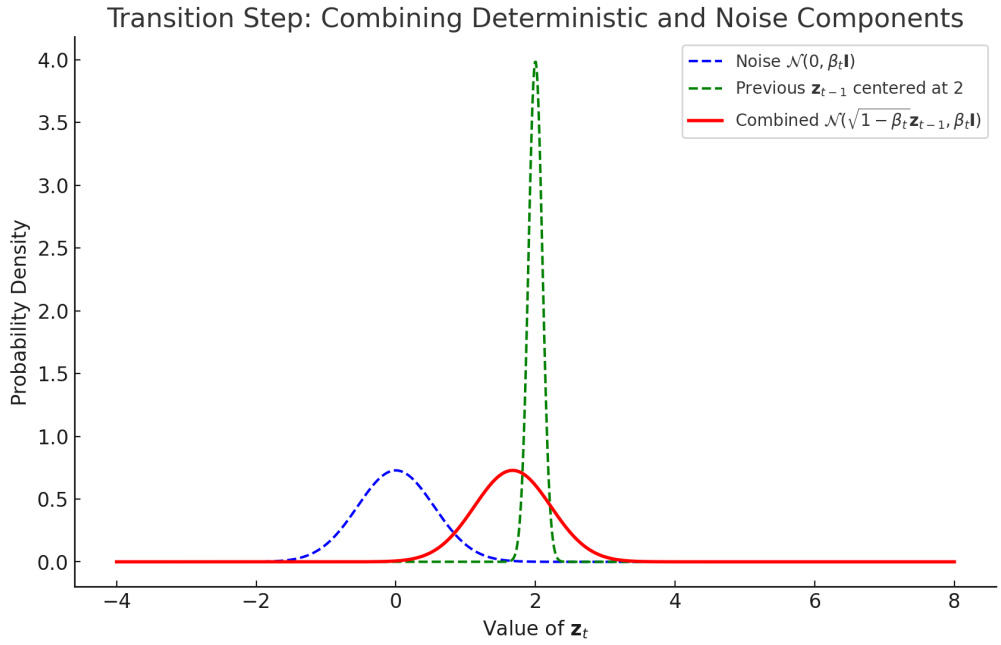
Distribution of Each Term

- The deterministic term $\sqrt{1-\beta_t}\,\mathbf{z}_{t-1}$ is fixed given $\mathbf{z}_{t-1}$, so it has a distribution centered at $\sqrt{1-\beta_t}\,\mathbf{z}_{t-1}$ with zero variance.
- The stochastic term $\sqrt{\beta_t}\,\boldsymbol{\epsilon}_t$: Since $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, scaling it by $\sqrt{\beta_t}$ gives:

  $$\sqrt{\beta_t}\,\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \beta_t \mathbf{I}).$$

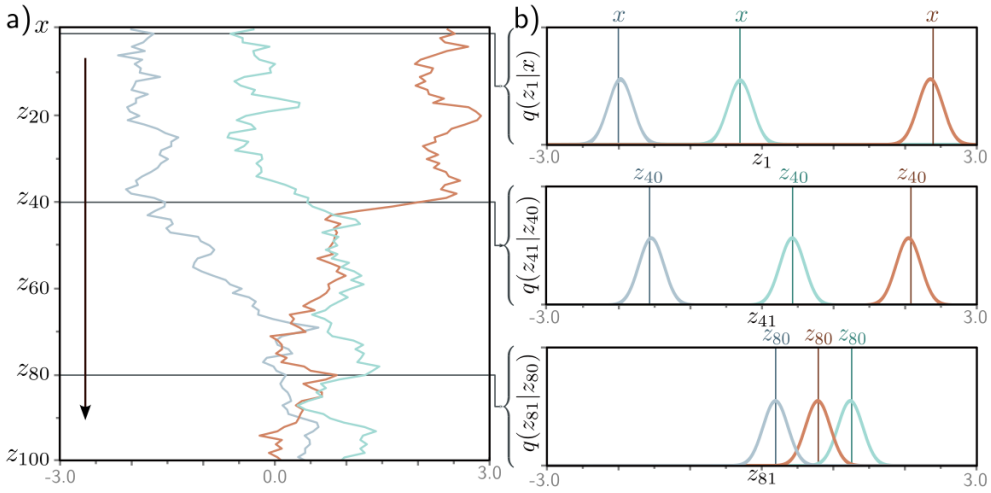
  This is a Gaussian distribution with zero mean and covariance $\beta_t \mathbf{I}$.

Summing the Two Terms



The sum of a deterministic term $\mathbf{m} = \sqrt{1-\beta_t}\,\mathbf{z}_{t-1}$ and a Gaussian term $\mathcal{N}(\mathbf{0}, \beta_t \mathbf{I})$ is also Gaussian. Specifically:

$$\mathbf{z}_t = \sqrt{1-\beta_t}\,\mathbf{z}_{t-1} + \sqrt{\beta_t}\,\boldsymbol{\epsilon}_t \sim \mathcal{N}\left(\mathbf{z}_t; \sqrt{1-\beta_t}\,\mathbf{z}_{t-1}, \beta_t \mathbf{I}\right).$$



$$q(\mathbf{z}_1 \mid \mathbf{x}) = \text{Norm}_{\mathbf{z}_1}\left[\sqrt{1-\beta_1}\,\mathbf{x}, \beta_1 \mathbf{I}\right],$$
$$q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) = \text{Norm}_{\mathbf{z}_t}\left[\sqrt{1-\beta_t}\,\mathbf{z}_{t-1}, \beta_t \mathbf{I}\right], \quad \forall t \in \{2, \ldots, T\}.$$

This process forms a *Markov chain*, where the distribution of $\mathbf{z}_t$ depends solely on the immediately preceding variable $\mathbf{z}_{t-1}$. After a sufficient number of steps T, all information about the original data $\mathbf{x}$ is lost, and the final latent variable $\mathbf{z}_T$ converges to a standard normal distribution, i.e., $q(\mathbf{z}_T \mid \mathbf{x}) = q(\mathbf{z}_T)$.

The joint distribution of the latent variables $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T$, conditioned on the input $\mathbf{x}$, can be factorized as:

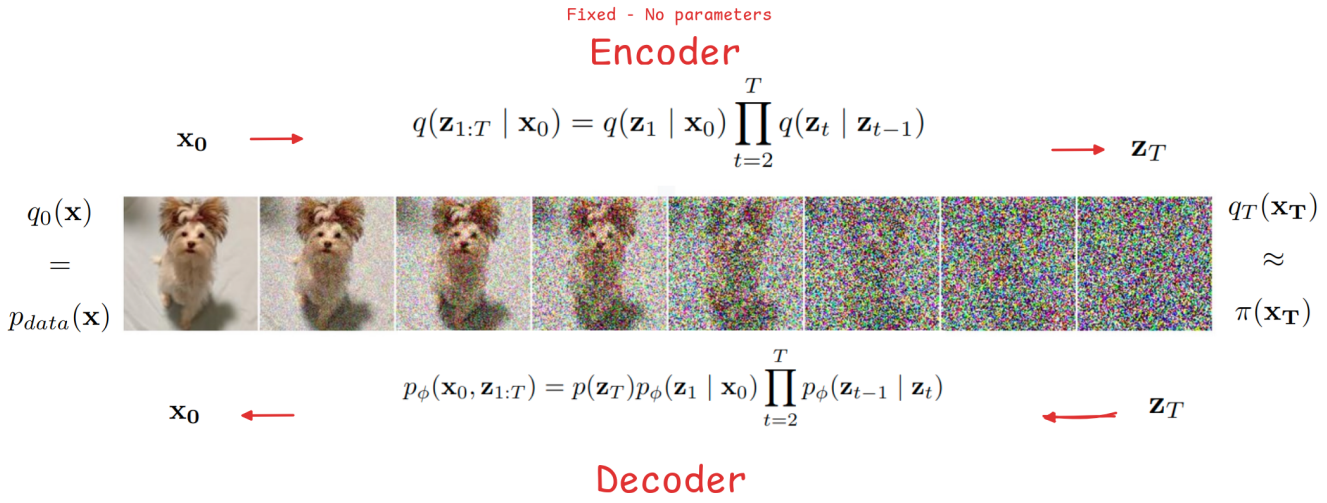
$$q(\mathbf{z}_{1:T} \mid \mathbf{x}) = q(\mathbf{z}_1 \mid \mathbf{x}) \prod_{t=2}^{T} q(\mathbf{z}_t \mid \mathbf{z}_{t-1}).$$

This is similar to encoder in VAE with the difference that in VAE we pass $\mathbf{x}_0$ to a NNs and it provides us with μ and σ^2 that we sample from but here we follow the iterative process of adding noise and we dont learn a NNs. You can consider every $\mathbf{z}_t$ as some latent vectors which represent different version of $\mathbf{x}_0$ with different noies level where mappings are not invertible.

Where:

$$\mathbf{x}_0 \in \Re^d$$
$$\mathbf{z}_t \in \Re^d \forall t$$

3.1 Diffusion kernel $q(\mathbf{z}_t \mid \mathbf{x})$



To train the decoder to reverse this process, we need to generate multiple samples $\mathbf{z}_t$ at time t for the same data point $\mathbf{x}$. However, sequentially generating these samples using this equation is computationally expensive for large t. Fortunately, there exists a closed-form expression for $q(\mathbf{z}_t \mid \mathbf{x})$, referred to as the *diffusion kernel*, which allows us to directly sample $\mathbf{z}_t$ from $\mathbf{x}$ without computing the intermediate variables $\mathbf{z}_1, \ldots, \mathbf{z}_{t-1}$.

To derive $q(\mathbf{z}_t \mid \mathbf{x})$, consider the first two steps of the forward process:

$$\mathbf{z}_1 = \sqrt{1-\beta_1}\,\mathbf{x} + \sqrt{\beta_1}\,\boldsymbol{\epsilon}_1, \quad \mathbf{z}_2 = \sqrt{1-\beta_2}\,\mathbf{z}_1 + \sqrt{\beta_2}\,\boldsymbol{\epsilon}_2.$$

Substituting the first equation into the second, we get:

$$\mathbf{z}_2 = \sqrt{1-\beta_2}\left(\sqrt{1-\beta_1}\,\mathbf{x} + \sqrt{\beta_1}\,\boldsymbol{\epsilon}_1\right) + \sqrt{\beta_2}\,\boldsymbol{\epsilon}_2.$$

Simplifying:

$$\mathbf{z}_2 = \sqrt{(1-\beta_2)(1-\beta_1)}\,\mathbf{x} + \sqrt{1-\beta_2}\,\sqrt{\beta_1}\,\boldsymbol{\epsilon}_1 + \sqrt{\beta_2}\,\boldsymbol{\epsilon}_2.$$

The last two terms are independent Gaussian samples with mean zero and variances $1-\beta_2$ (scaled by $(1-\beta_1)$) and β_2, respectively. Since the variance of the sum of independent Gaussian variables is the sum of their individual variances, we can write:

$$\mathbf{z}_2 = \sqrt{(1-\beta_2)(1-\beta_1)}\,\mathbf{x} + \sqrt{1-(1-\beta_2)(1-\beta_1)}\,\boldsymbol{\epsilon},$$

By repeating this substitution process iteratively for $\mathbf{z}_3, \mathbf{z}_4, \ldots, \mathbf{z}_t$, we generalize the forward process to time t. The result is:

$$\mathbf{z}_t = \sqrt{\alpha_t}\,\mathbf{x} + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon},$$

where

$$\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$$

.

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

.

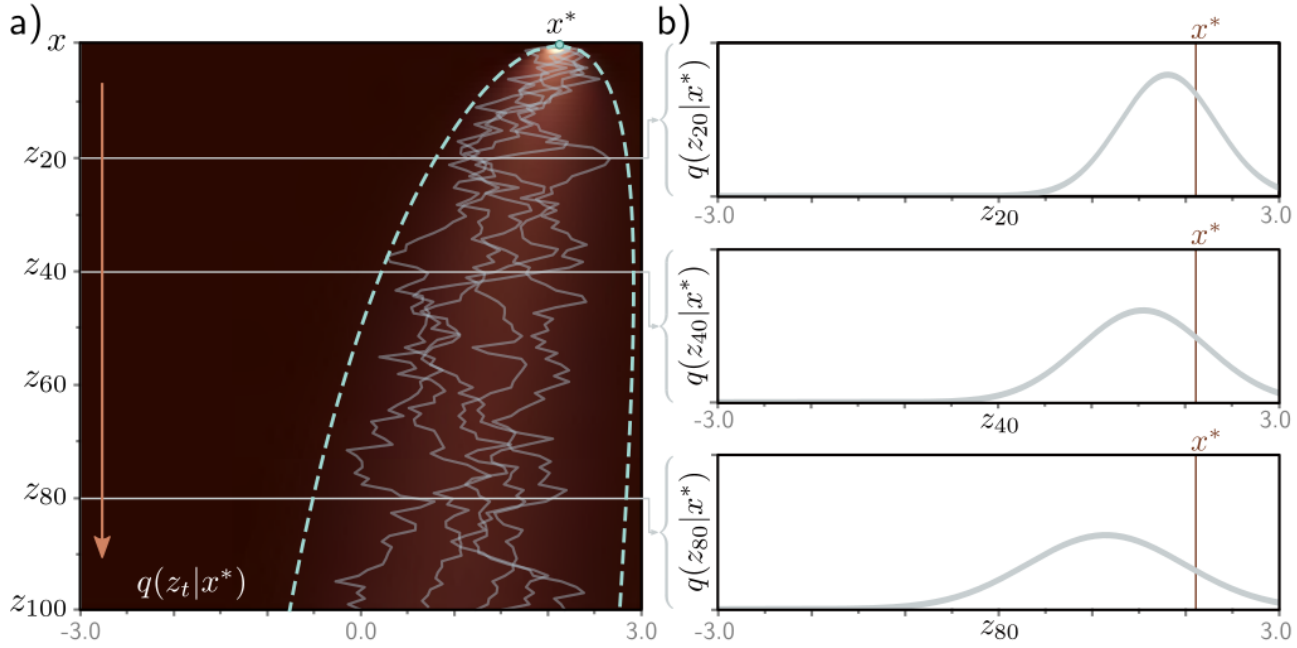
This allows us to express the conditional distribution $q(\mathbf{z}_t \mid \mathbf{x})$ in closed form as:

$$q(\mathbf{z}_t \mid \mathbf{x}) = \mathrm{Norm}_{\mathbf{z}_t}\left[\sqrt{\alpha_t}\,\mathbf{x}, (1-\alpha_t)\mathbf{I}\right].$$

In this expression:

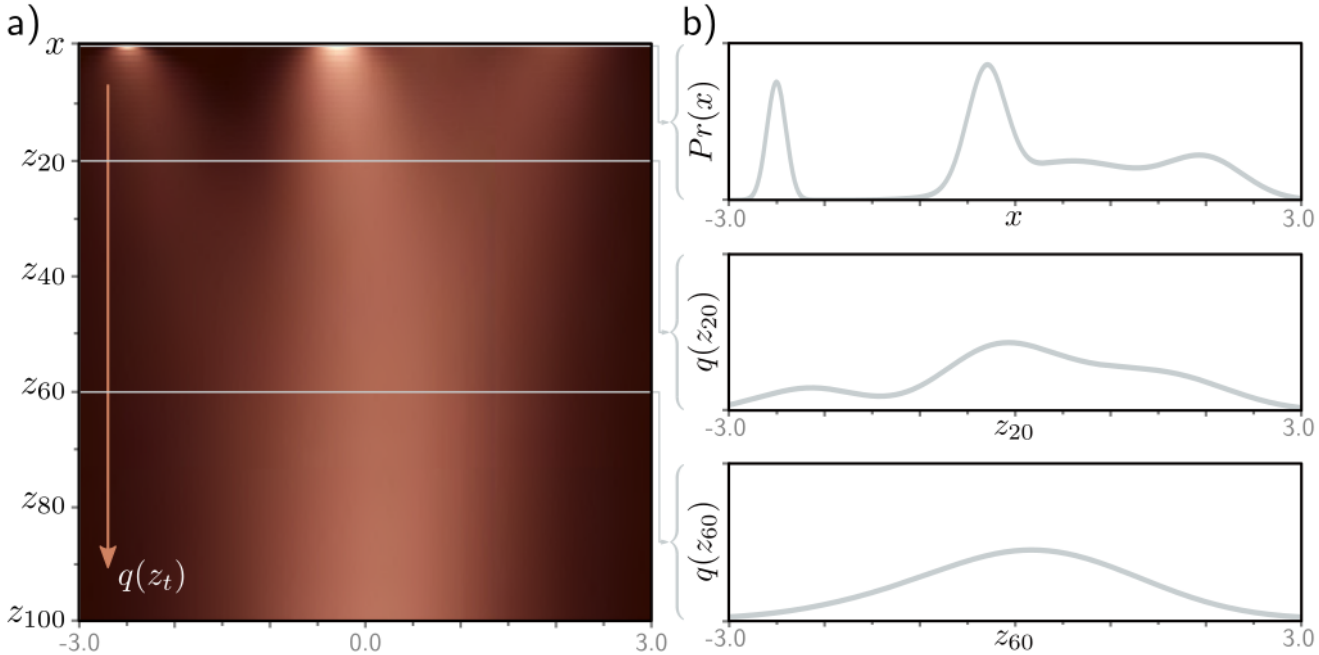
- The mean $\sqrt{\alpha_t}\,\mathbf{x}$ reflects the scaled input data point.
- The variance $(1-\alpha_t)\mathbf{I}$ grows with t, progressively blending the data with Gaussian noise.

This closed-form solution allows us to directly sample $\mathbf{z}_t$ at any time t given $\mathbf{x}$, without explicitly computing the intermediate variables $\mathbf{z}_1, \ldots, \mathbf{z}_{t-1}$.



Diffusion kernel. a) The point $x^* = 2.0$ is propagated through the latent variables (five paths shown in gray). The diffusion kernel $q(\mathbf{z}_t \mid x^*)$ is the probability distribution over variable $\mathbf{z}_t$ given that we started from x^*. It can be computed in closed form and is a normal distribution whose mean moves toward zero and whose variance increases as t increases. The heatmap illustrates $q(\mathbf{z}_t \mid x^*)$ for each variable, with cyan lines indicating ± 2 standard deviations from the mean. b) The diffusion kernel $q(\mathbf{z}_t \mid x^*)$ is explicitly shown for $t = 20, 40, 80$. In practice, the diffusion kernel allows us to sample a latent variable $\mathbf{z}_t$ corresponding to a given x^* without computing the intermediate variables $\mathbf{z}_1, \ldots, \mathbf{z}_{t-1}$. As t becomes very large, the diffusion kernel converges to a standard normal distribution.

3.2 Marginal distributions $q(\mathbf{z}_t)$



The marginal distribution $q(\mathbf{z}_t)$ is the probability of observing a value of $\mathbf{z}_t$ given the distribution of possible starting points $\mathbf{x}$ and the possible diffusion paths for each starting point. It can be computed by considering the joint

distribution $q(\mathbf{x}, \mathbf{z}_{1,\ldots,t})$ and marginalizing over all the variables except $\mathbf{z}_t$:

$$q(\mathbf{z}_t) = \iint q(\mathbf{z}_{1,\ldots,t}, \mathbf{x})\, d\mathbf{z}_{1,\ldots,t-1}\, d\mathbf{x}.$$

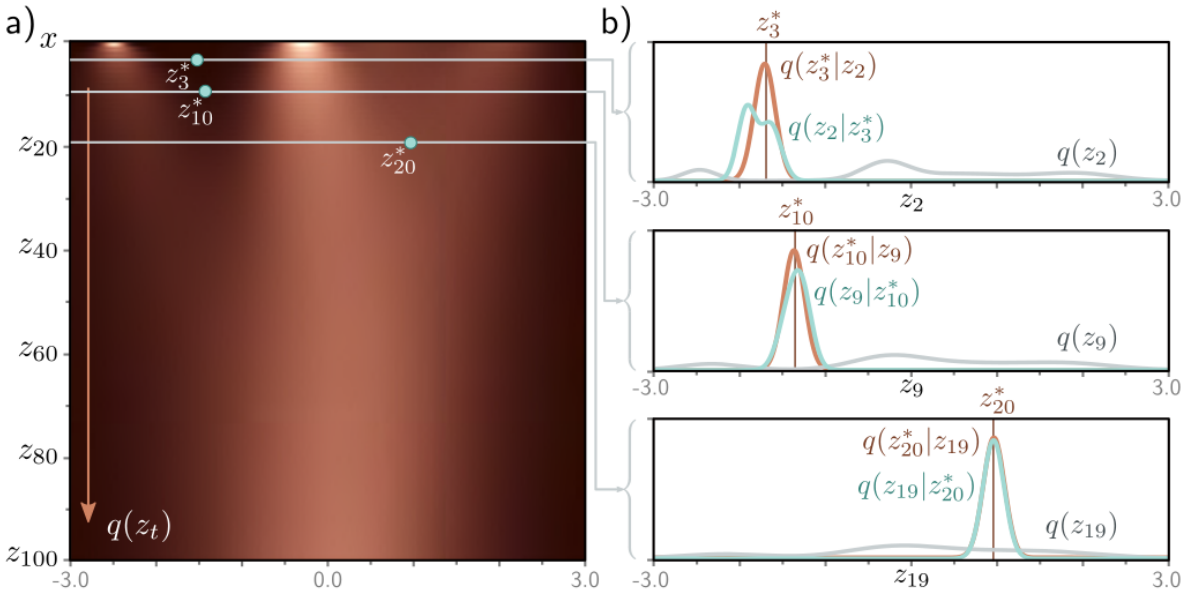
$$q(\mathbf{z}_t) = \iint q(\mathbf{z}_{1,\ldots,t} \mid \mathbf{x}) P_r(\mathbf{x})\, d\mathbf{z}_{1,\ldots,t-1}\, d\mathbf{x}.$$

Given the diffusion kernel $q(\mathbf{z}_t \mid \mathbf{x})$, which bypasses the intermediate variables, we can simplify $q(\mathbf{z}_t)$ as:

$$q(\mathbf{z}_t) = \int q(\mathbf{z}_t \mid \mathbf{x}) P_r(\mathbf{x})\, d\mathbf{x}.$$

Hence, if we repeatedly sample from the data distribution $P_r(\mathbf{x})$ and apply the diffusion kernel $q(\mathbf{z}_t \mid \mathbf{x})$ to each sample, the result is the marginal distribution $q(\mathbf{z}_t)$. However, the marginal distribution cannot be computed in closed form because the original data distribution $P_r(\mathbf{x})$ is unknown.

3.3 Conditional distribution $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$



The conditional probability $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$ defines the mixing process described earlier. To reverse this process, we use Bayes' rule:

$$q(\mathbf{z}_{t-1} \mid \mathbf{z}_t) = \frac{q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) q(\mathbf{z}_{t-1})}{q(\mathbf{z}_t)}.$$

This expression is generally intractable because the marginal distribution $q(\mathbf{z}_{t-1})$ cannot be computed directly. However, for small values of t, $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$ can be approximated numerically. In many practical scenarios, the reverse distribution is well-approximated, which is critical for building a decoder to approximate the reverse diffusion process.

The probability $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$ is computed using Bayes' rule and is proportional to $q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) q(\mathbf{z}_{t-1})$. In general, this distribution is not normally distributed (as shown in the top graph); however, in many cases, a normal distribution provides a good approximation (as illustrated in the bottom two graphs).

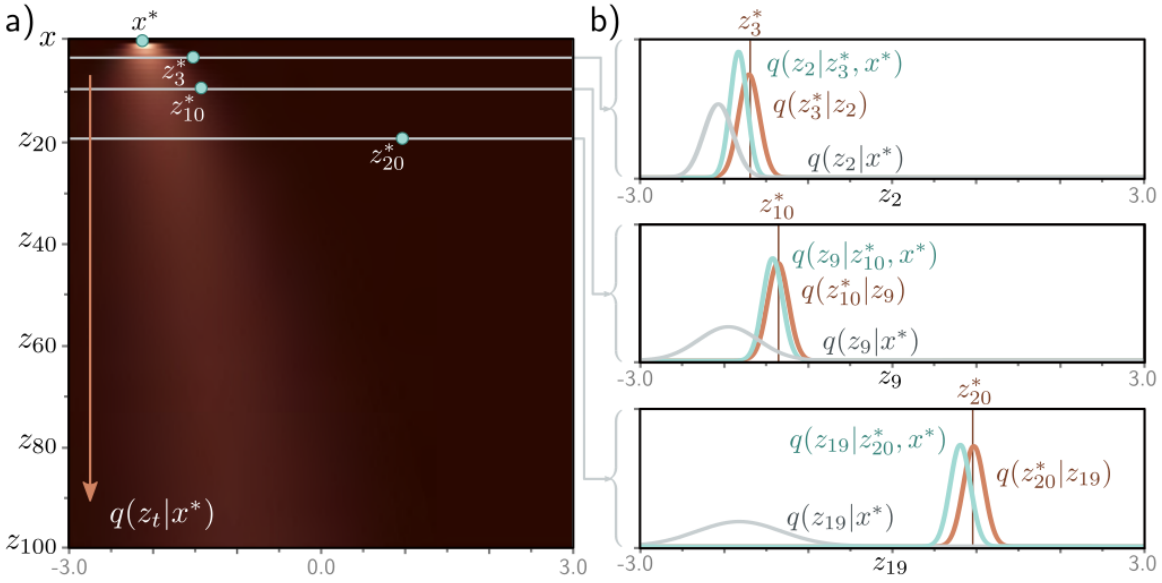
3.4 Conditional diffusion distribution $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{x})$

There is one final distribution related to the encoder to consider. As noted earlier, the conditional distribution $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$ cannot be directly computed because the marginal distribution $q(\mathbf{z}_{t-1})$ is unknown. However, if the starting variable $\mathbf{x}$ is known, then the distribution $q(\mathbf{z}_{t-1} \mid \mathbf{x})$ at the previous time step can be determined. This distribution corresponds to the diffusion kernel and is normally distributed.

Given this, we can compute the conditional diffusion distribution $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{x})$ in closed form. This distribution is used to train the decoder and represents the distribution over $\mathbf{z}_{t-1}$, given the current latent variable $\mathbf{z}_t$ and training data example $\mathbf{x}$. The the conditional diffusion distribution given as :

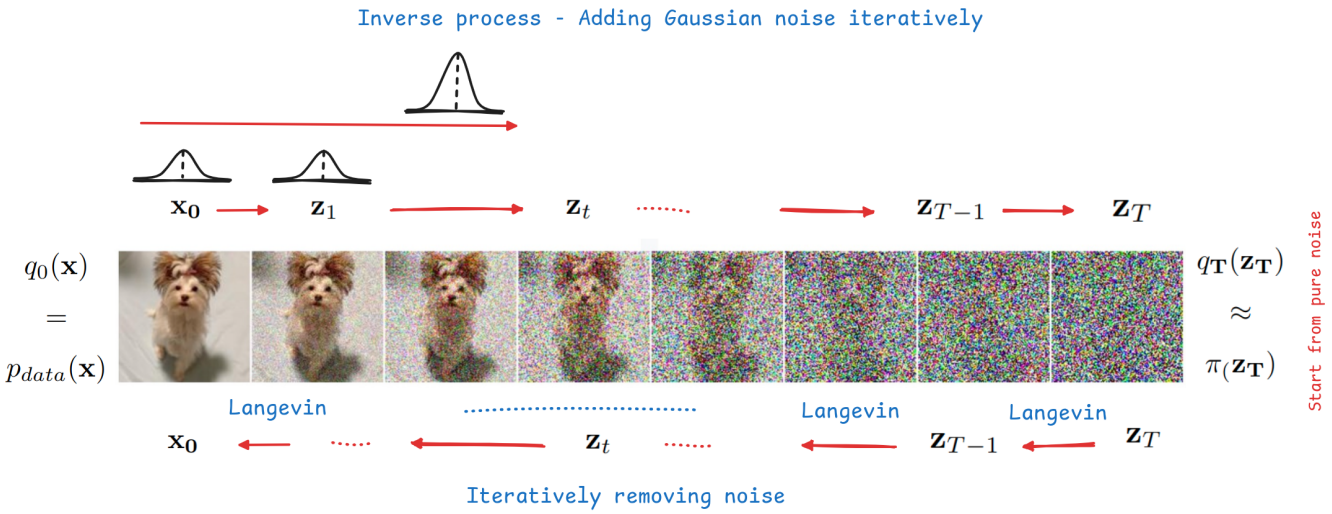
$$q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{x}) = \text{Norm}_{\mathbf{z}_{t-1}}\left[\frac{(1-\alpha_{t-1})}{1-\alpha_t}\sqrt{1-\beta_t}\,\mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}, \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}\mathbf{I}\right].$$

This distribution is already normalized as a properly scaled Gaussian probability distribution, and the constants of proportionality cancel out during the derivation.

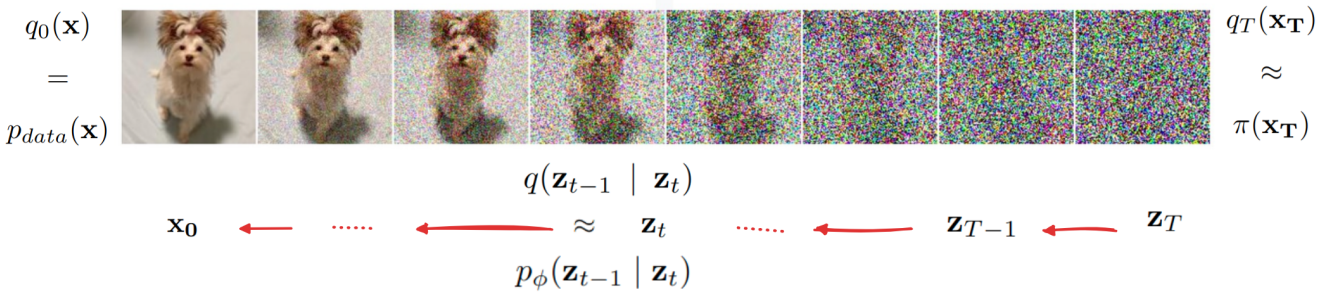


The conditional distribution $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{x})$ is illustrated for $x^* = -2.1$, where three points $\mathbf{z}_t^*$ are highlighted. The probability $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t^*, x^*)$ is computed using Bayes' rule and is proportional to $q(\mathbf{z}_t^* \mid \mathbf{z}_{t-1})q(\mathbf{z}_{t-1} \mid x^*)$. This distribution is normally distributed and can be expressed in closed form. The first likelihood term $q(\mathbf{z}_t^* \mid \mathbf{z}_{t-1})$ is a Gaussian distribution in $\mathbf{z}_{t-1}$ with a mean that is slightly farther from zero than $\mathbf{z}_t^*$ (shown as brown curves). The second term $q(\mathbf{z}_{t-1} \mid x^*)$ represents the diffusion kernel and is illustrated as gray curves.

4 Backward Diffusion Process - Decoder



We iteratively removing noise using Langevin chain and convert $\mathbf{x}_t$ into a sample from the data distribution and use the new particle to start a new Langevin chain where we follow the gradients corresponding to a data distribution with a little bit less noise and continue this process unil we generate a clean sample.



We can reverse this process (denoising) by:

- Sample $\mathbf{x}_T$ from $\pi(\mathbf{x}_T)$, i.e. from pure noise.
- Iteratively sample from the true denoising distribution $q(\mathbf{x}_{T-1}|\mathbf{x}_t)$

When we learn a diffusion model, we learn the *reverse process*. In other words, we model a series of probabilistic mappings that go backwards, starting from the latent variable $\mathbf{z}_T$ to $\mathbf{z}_{T-1}$, from $\mathbf{z}_{T-1}$ to $\mathbf{z}_{T-2}$, and so on until we reach the data $\mathbf{x}$. The true reverse distributions $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$ of the diffusion process are complex multi-modal distributions, which depend on the data distribution $P_r(\mathbf{x})$. **To approximate these, we assume Gaussian distributions for the reverse process as follows:**

Prior Distribution for $\mathbf{z}_T$

$$P_r(\mathbf{z}_T) = \mathcal{N}(\mathbf{z}_T; \mathbf{0}, \mathbf{I}) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}\mathbf{z}_T^\top \mathbf{z}_T\right).$$

where d is the dimension of $\mathbf{z}_T$, $\mathbf{0}$ is the mean vector, and $\mathbf{I}$ is the identity covariance matrix.

Conditional Distribution for $\mathbf{z}_{t-1} \mid \mathbf{z}_t, \phi_t$:

$$p_r(\mathbf{z}_{t-1} \mid \mathbf{z}_t) = \mathcal{N}\left(\mathbf{z}_{t-1}; \mathbf{f}_t(\mathbf{z}_t, \phi_t), \sigma_t^2 \mathbf{I}\right).$$

The expanded form is:

$$p_r(\mathbf{z}_{t-1} \mid \mathbf{z}_t) = \frac{1}{(2\pi\sigma_t^2)^{d/2}} \exp\left(-\frac{1}{2\sigma_t^2} \left\|\mathbf{z}_{t-1} - \mathbf{f}_t(\mathbf{z}_t, \phi_t)\right\|^2\right),$$

- $\mathbf{f}_t(\mathbf{z}_t, \boldsymbol{\phi}_t)$ is a neural network that computes the mean of the normal distribution for the estimated mapping from $\mathbf{z}_t$ to the preceding latent variable $\mathbf{z}_{t-1}$,
- σ_t^2 are predetermined terms that control the variance of the Gaussian distributions.

Conditional Distribution for $\mathbf{x} \mid \mathbf{z}_1, \boldsymbol{\phi}_1$:

$$P_r(\mathbf{x} \mid \mathbf{z}_1, \boldsymbol{\phi}_1) = \frac{1}{(2\pi\sigma_1^2)^{d/2}} \exp\left(-\frac{1}{2\sigma_1^2} \left\|\mathbf{x} - \mathbf{f}_1(\mathbf{z}_1, \phi_1)\right\|^2\right),$$

Joint distribution:

$$p_r(\mathbf{x}_0, \mathbf{z}_{1:T} | \phi_{1\ldots T}) = p(\mathbf{z}_T) p_r(\mathbf{x} \mid \mathbf{z}_1, \boldsymbol{\phi}_1) \prod_{t=2}^{T} p_r(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \boldsymbol{\phi}_t)$$

where:

- $\mathbf{z}_{1:T}$ represents the sequence of latent variables,
- $p(\mathbf{z}_T)$ is the prior distribution at the final time step T,
- $p_{\boldsymbol{\phi}_t}(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$ denotes the conditional reverse process distribution parameterized by ϕ,

If the diffusion process hyperparameters β_t are small (close to zero) and the number of time steps T is large, this Gaussian approximation is reasonable.

4.1 Generate Samples:

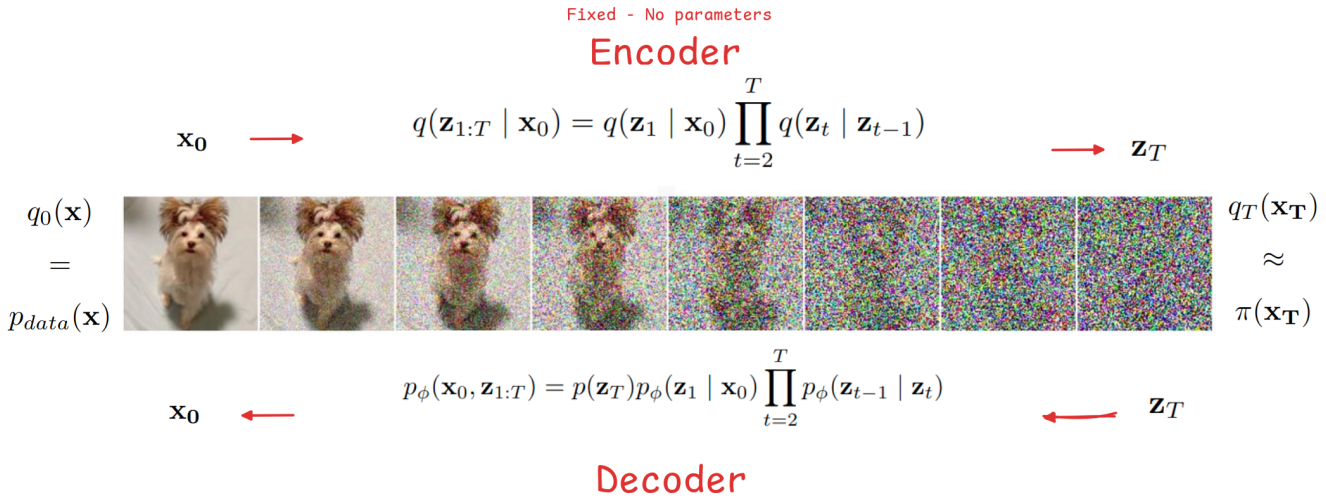
To generate new samples from the data distribution $P_r(\mathbf{x})$, we use *ancestral sampling*. The process proceeds as follows:

1. Draw $\mathbf{z}_T \sim P_r(\mathbf{z}_T) = \mathcal{N}(\mathbf{z}_T; \mathbf{0}, \mathbf{I}) = \pi$,
2. Sample $\mathbf{z}_{T-1} \sim P_r(\mathbf{z}_{T-1} \mid \mathbf{z}_T, \boldsymbol{\phi}_T)$,
3. Sample $\mathbf{z}_{T-2} \sim P_r(\mathbf{z}_{T-2} \mid \mathbf{z}_{T-1}, \boldsymbol{\phi}_{T-1})$,
4. Continue this process iteratively until we reach $\mathbf{z}_1$,
5. Finally, generate the data $\mathbf{x}$ by sampling:

$$\mathbf{x} \sim P_r(\mathbf{x} \mid \mathbf{z}_1, \boldsymbol{\phi}_1) = \mathcal{N}\left(\mathbf{x}; \mathbf{f}_1(\mathbf{z}_1, \boldsymbol{\phi}_1), \sigma_1^2 \mathbf{I}\right).$$

In this way, the reverse process generates new examples that approximate the true data distribution $P_r(\mathbf{x})$.

5 Training



The joint distribution of the observed variable $\mathbf{x}$ and the latent variables $\{\mathbf{z}_t\}$ is given by:

$$p_r(\mathbf{x}_0, \mathbf{z}_{1:T}|\phi_{1\ldots T}) = p(\mathbf{z}_T) p_{\phi_1}(\mathbf{z}_1 \mid \mathbf{x}_0) \prod_{t=2}^{T} p_r(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \phi_t)$$

The likelihood of the observed data $p_r(\mathbf{x})$ is obtained by marginalizing over the latent variables:

$$p_r(\mathbf{x}) = \int p_r(\mathbf{x}, \mathbf{z}_{1\ldots T}|\phi_{1\ldots T}) d\mathbf{z}_{1\ldots T}.$$

To train the model, we maximize the log-likelihood of the training data $\{\mathbf{x}_i\}$ with respect to the parameters $\boldsymbol{\phi}$:

$$\hat{\boldsymbol{\phi}}_{1\ldots T} = \arg\max_{\boldsymbol{\phi}_{1\ldots T}} \left[\sum_{i=1}^{I} \log p_r(\mathbf{x}_i|\phi_{1\ldots T}) \right].$$

Since the marginalization in the likelihood $p_r(\mathbf{x}_i)$ is intractable, we use Jensen's inequality to define a lower bound on the likelihood. This allows us to optimize the parameters $\boldsymbol{\phi}_{1\ldots T}$ with respect to the bound, similar to the approach used for variational autoencoders (VAEs).

This gives us the evidence lower bound (ELBO):

$$\text{ELBO}[\boldsymbol{\phi}_{1\ldots T}] = \int q(\mathbf{z}_{1\ldots T} \mid \mathbf{x}_0) \log \frac{p_r(\mathbf{x}_0, \mathbf{z}_{1:T}|\phi_{1\ldots T})}{q(\mathbf{z}_{1:T} \mid \mathbf{x}_0)} d\mathbf{z}_{1\ldots T}.$$

Alternatively, in terms of the expectation, the ELBO can be expressed as:

$$\text{ELBO}[\boldsymbol{\phi}_{1\ldots T}] = \mathbb{E}_{q(\mathbf{x}_0) q(\mathbf{z}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_r(\mathbf{x}_0, \mathbf{z}_{1:T}|\phi_{1\ldots T})}{q(\mathbf{z}_{1:T} \mid \mathbf{x}_0)} \right].$$

$$\mathbb{E}_{q(\mathbf{x}_0)} \left[\log p_\theta(\mathbf{x}_0) \right] \geq \mathbb{E}_{q(\mathbf{x}_0) q(\mathbf{z}_{1:T}|\mathbf{x}_0)} \left[\log \frac{p_r(\mathbf{x}_0, \mathbf{z}_{1:T}|\phi_{1\ldots T})}{q(\mathbf{z}_{1:T} \mid \mathbf{x}_0)} \right].$$

which can be shown as negative ELBO:

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_r(\mathbf{x}_0)\right] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{z}_{1:T}|\mathbf{x}_0)}\left[-\log \frac{p_r(\mathbf{x}_0, \mathbf{z}_{1:T}|\phi_{1\ldots T})}{q(\mathbf{z}_{1:T} \mid \mathbf{x}_0)}\right].$$

Here:

- $q(\mathbf{z}_{1\ldots T} \mid \mathbf{x}_0)$ is the approximate posterior distribution,
- $p_r(\mathbf{x}_0, \mathbf{z}_{1\ldots T}|\phi_{1\ldots T})$ is the joint likelihood under the model.

The ELBO provides a lower bound on the log marginal likelihood $\log P_r(\mathbf{x} \mid \boldsymbol{\phi}_{1\ldots T})$, making it a practical surrogate objective for optimization.

5.1 Loss Function

To fit the model, we maximize the ELBO with respect to the parameters $\boldsymbol{\phi}_{1\ldots T}$. We recast this as a minimization by multiplying with minus one and approximating the expectations with samples to give the loss function:

$$L[\boldsymbol{\phi}_{1\ldots T}] = \sum_{i=1}^{I} \underbrace{-\log\left[\text{Norm}_{\boldsymbol{x}_i}\left(f_1[\boldsymbol{z}_{i1}, \boldsymbol{\phi}_1], \sigma_1^2 \boldsymbol{I}\right)\right]}_{\text{reconstruction term}} + \sum_{t=2}^{T} \frac{1}{2\sigma_t^2} \left\| \underbrace{\frac{1-\alpha_{t-1}}{1-\alpha_t}\sqrt{1-\beta_t}\boldsymbol{z}_{it} + \frac{\sqrt{\alpha_t\beta_t}}{1-\alpha_t}\boldsymbol{x}_i}_{\text{target, mean of } q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})} - \underbrace{\boldsymbol{f}_t[\boldsymbol{z}_{it}, \boldsymbol{\phi}_t]}_{\text{predicted } \boldsymbol{z}_{t-1}} \right\|^2$$

where $\boldsymbol{x}_i$ is the i^{th} data point, and $\boldsymbol{z}_{it}$ is the associated latent variable at diffusion step t.

The Key Concept: Gaussian Distribution and Its Mean

For a Gaussian distribution:

$$p(z) = \mathcal{N}(z \mid \mu, \sigma^2),$$

the mean μ represents the center of the distribution.

The mean is also the value that minimizes the expected squared error (L_2-norm loss):

$$\mathbb{E}_{p(z)}\left[(z-\hat{z})^2\right] \quad \text{is minimized when} \quad \hat{z} = \mu.$$

Breakdown:

- **Target (mean):**

  $$\frac{1-\alpha_{t-1}}{1-\alpha_t}\sqrt{1-\beta_t}\,\boldsymbol{z}_{it} + \frac{\sqrt{\alpha_t\beta_t}}{1-\alpha_t}\boldsymbol{x}_i$$

  This is the *mean* of the posterior $q(\boldsymbol{z}_{t-1} \mid \boldsymbol{z}_t, \boldsymbol{x})$, representing the optimal denoised latent variable $\boldsymbol{z}_{t-1}$.

- **Prediction:**

  $$\boldsymbol{f}_t[\boldsymbol{z}_{it}, \boldsymbol{\phi}_t]$$

  We estimate/predict this using NNs. The model's predicted $\boldsymbol{z}_{t-1}$, based on the input $\boldsymbol{z}_{it}$ and parameters $\boldsymbol{\phi}_t$.

- **Loss (Error):**

  $$\|\text{Target} - \text{Prediction}\|^2$$

  Minimizing this error drives the model to approximate the posterior mean, ensuring accurate denoising and reconstruction during the reverse diffusion process.

Why is this important? The posterior mean provides the *optimal denoised estimate* of $\boldsymbol{z}_{t-1}$. Predicting it step-by-step enables the model to reconstruct the original data $\boldsymbol{x}_i$ effectively during the reverse diffusion process.

6 Diffusion Models as Score-based Models

The diffusion process progressively adds Gaussian noise to the original data. At time step t, the noisy latent variable $\mathbf{z}_t$ is expressed as:

$$\mathbf{z}_t = \sqrt{\alpha_t}\,\mathbf{x} + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon},$$

where α_t is a weighting coefficient, $\mathbf{x}$ is the original data, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is Gaussian noise. However, during inference (sampling), we don't know the actual noise ϵ for a given input.

The original data $\mathbf{x}$ can be recovered by expressing it as the diffused latent variable $\mathbf{z}_t$ minus the noise:

$$\mathbf{x} = \frac{1}{\sqrt{\alpha_t}} \cdot \mathbf{z}_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}} \cdot \boldsymbol{\epsilon}.$$

The backward (denoising) process is modeled as a Gaussian distribution parameterized by:

$$p_r(\mathbf{z}_{t-1} \mid \mathbf{z}_t) = \mathcal{N}\left(\mathbf{z}_{t-1}; \mathbf{f}_t(\mathbf{z}_t, \phi_t), \sigma_t^2 \mathbf{I}\right),$$

Here, $\mathbf{f}_t$ is the predicted mean of $\mathbf{z}_{t-1}$ given $\mathbf{z}_t$. In this formulation, we train the neural network to directly predict the mean of $\mathbf{z}_{t-1}$.

$$\mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t] = \frac{1}{\sqrt{\alpha_t}}\mathbf{z}_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}}\boldsymbol{\epsilon}$$

This tells us that if we had access to the noise $\boldsymbol{\epsilon}$ that generated $\mathbf{z}_t$, we could directly compute the mean $\mathbf{f}_t$.

Decoder Parameterization:

1. **Original Role of f:** In the initial loss function, $\mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$ is used to predict $\mathbf{z}_{t-1}$, which represents the denoised latent variable at the previous time step. This corresponds to the *reconstruction term* and aligns with minimizing the difference between the predicted $\mathbf{z}_{t-1}$ and the target mean of the posterior $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{x})$.

   Mathematically, this looks like:

   $$\mathbf{f}_t[\mathbf{z}_{it}, \boldsymbol{\phi}_t] \approx \text{mean of } q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{x}).$$

2. **Replacing with Noise $\boldsymbol{\epsilon}$:** The diffusion process introduces Gaussian noise step-by-step to the data $\mathbf{x}$, such that the noised latent variable $\mathbf{z}_t$ at step t becomes a mixture of $\mathbf{x}$ and the cumulative Gaussian noise $\boldsymbol{\epsilon}$.

   Instead of predicting $\mathbf{z}_{t-1}$, it is more efficient to predict the noise $\boldsymbol{\epsilon}$ that was added to the data. The noise model is:

   $$\mathbf{z}_t = \sqrt{\alpha_t}\,\mathbf{x} + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon}.$$

   A network $\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$ is trained to predict $\boldsymbol{\epsilon}$. Once $\boldsymbol{\epsilon}$ is known, the original data $\mathbf{x}$ or the mean $\mathbf{z}_{t-1}$ can be recovered using the above equation.

3. Using the earlier form of $\mathbf{f}_t$, we can now substitute $\hat{\boldsymbol{\epsilon}} = \mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$ into the equation:

   $$\mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t] = \frac{1}{\sqrt{\alpha_t}}\mathbf{z}_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}}\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$$

   $$=$$

   $$\mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t] = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t].$$

If we parametrized these NNs that gives us the mean of these Gaussian using this form in terms of the $\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$. Then it takes z_t as an input and then predicts the noise which was added to it ($\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$) and subtract it from the z_t which is equivalent to denoising score matching loss:

Why is it better to predict ϵ instead of $\mathbf{f}_t$? If we train the model $\mathbf{g}_t$ to predict $\boldsymbol{\epsilon}$ directly, we get several benefits:

- **Simplicity:** Instead of having the network output the mean $\mathbf{f}_t$, it predicts the noise $\boldsymbol{\epsilon}$ that was added to $\mathbf{x}$ to obtain $\mathbf{z}_t$.
- **Consistency:** Since the noise $\boldsymbol{\epsilon}$ is drawn from a simple distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, it's easier to model and predict than the complex mean of the conditional distribution $p_r(\mathbf{z}_{t-1} \mid \mathbf{z}_t)$.
- **Simpler loss function:** Instead of directly matching the mean of $q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{x})$, we minimize the distance between $\boldsymbol{\epsilon}$ and its predicted version $\hat{\boldsymbol{\epsilon}}$. This leads to the loss function:

  $$L = \mathbb{E}_{\mathbf{x}_0, t, \boldsymbol{\epsilon}} \left[\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}\|^2 \right]$$

  This is a simple mean-squared error (MSE) loss between the true noise $\boldsymbol{\epsilon}$ and the predicted noise $\hat{\boldsymbol{\epsilon}}$.

5. Loss function (for training $\mathbf{g}_t$)

Since we know $\mathbf{z}_t$ is generated as:

$$\mathbf{z}_t = \sqrt{\alpha_t}\mathbf{x} + \sqrt{1 - \alpha_t}\boldsymbol{\epsilon}$$

Given $\mathbf{x}$, t, and $\boldsymbol{\epsilon}$, we can generate training samples for $\mathbf{z}_t$. The loss is then the mean-squared error (MSE) between the true noise $\boldsymbol{\epsilon}$ and the predicted noise $\hat{\boldsymbol{\epsilon}} = \mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$. The loss function is:

$$L = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), t \sim \mathcal{U}\{1,T\}, \boldsymbol{\epsilon} \sim \mathcal{N}(0,\mathbf{I})} \left[\|\boldsymbol{\epsilon} - \mathbf{g}_t(\mathbf{z}_t, \phi_t)\|^2 \right]$$

$$=$$

$$L = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), t \sim \mathcal{U}\{1,T\}, \epsilon \sim \mathcal{N}(0,\mathbf{I})} \left[\lambda_t \|\epsilon - \mathbf{g}_t(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon)\|^2 \right].$$

This loss function can be seen as a denoising score-matching loss, since $\mathbf{g}_t$ tries to predict the noise that was added to $\mathbf{x}$ to generate $\mathbf{z}_t$.

Why Do We Need λ_t ?

The choice of λ_t depends on how we think about the variance of the noise at each step. The noise at time t has a variance determined by $\bar{\alpha}_t$ and $1 - \bar{\alpha}_t$. If we don't properly account for this, certain timesteps will contribute more to the total loss than others. To make the loss consistent across timesteps, we introduce a time-dependent factor λ_t.

Some possible choices of λ_t include:

- **Simple constant:** $\lambda_t = 1$ for all t (not ideal since it doesn't balance contributions from each step).
- **Variance scaling:** $\lambda_t = \frac{1}{1-\bar{\alpha}_t}$. This reweights the loss to balance for the changing magnitude of the signal at each t.

These reweightings ensure that the model pays more attention to "harder" noise samples (when the signal is weak) than to "easy" samples (when the signal dominates).

Summary:

$$L = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), t \sim \mathcal{U}\{1,T\}, \epsilon \sim \mathcal{N}(0,\mathbf{I})} \underbrace{\left[\underbrace{\lambda_t \| \epsilon - \mathbf{g}_t(\underbrace{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon}_{\text{sample}})}_{\text{actual noise - predicted noise}} \|^2 \right]}_{\text{scaled error}}$$

Average error

Minimizing the negative ELBO or maximizing the lower bound on the average log-likelihood is the same as trying to estimate the scores of these noise-perturbed data distributions.

Training and Inference

Algorithm 18.1: Diffusion model training

Input: Training data $\mathbf{x}$
Output: Model parameters $\boldsymbol{\phi}_t$
repeat
  for $i \in \mathcal{B}$ **do** // For every training example index in batch
    $t \sim \text{Uniform}[1, \ldots T]$ // Sample random timestep
    $\boldsymbol{\epsilon} \sim \text{Norm}[\mathbf{0}, \mathbf{I}]$ // Sample noise
    $\ell_i = \left\| \mathbf{g}_t\left[\sqrt{\alpha_t}\mathbf{x}_i + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, \boldsymbol{\phi}_t\right] - \boldsymbol{\epsilon} \right\|^2$ // Compute individual loss
  Accumulate losses for batch and take gradient step
until converged

Algorithm 18.2: Sampling

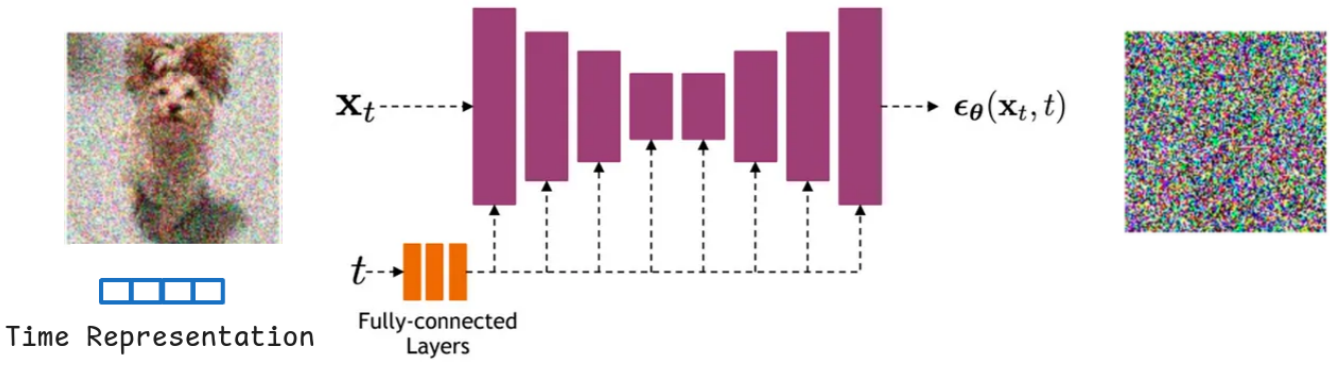
Input: Model, $\mathbf{g}_t[\bullet, \boldsymbol{\phi}_t]$
Output: Sample, $\mathbf{x}$
$\mathbf{z}_T \sim \text{Norm}_{\mathbf{z}}[\mathbf{0}, \mathbf{I}]$ // Sample last latent variable
for $t = T \ldots 2$ **do**
  $\hat{\mathbf{z}}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$ // Predict previous latent variable
  $\boldsymbol{\epsilon} \sim \text{Norm}_{\boldsymbol{\epsilon}}[\mathbf{0}, \mathbf{I}]$ // Draw new noise vector
  $\mathbf{z}_{t-1} = \hat{\mathbf{z}}_{t-1} + \sigma_t\boldsymbol{\epsilon}$ // Add noise to previous latent variable
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}\sqrt{1-\beta_1}}\mathbf{g}_1[\mathbf{z}_1, \boldsymbol{\phi}_1]$ // Generate sample from $\mathbf{z}_1$ without noise

Here the training loss is basically the Denoising score matching and the sampling is basically the Annealed Langevin Dynamics.

7 Architectures for the denoiser

7.1 UNET:

we need to construct models that can take a noisy image and predict the noise that was added at each step. The obvious architectural choice for this image-to-image mapping is the U- Net. However, there may be a very large number of diffusion steps, and training and storing multiple U-Nets is inefficient. The solution is to train a single U-Net that also takes a predetermined vector representing the time step as input:



1. Forward Diffusion Process The forward process progressively adds Gaussian noise to an image $\mathbf{x}_0$ over discrete timesteps $t \in \{1, 2, \ldots, T\}$. At timestep t, the noisy image $\mathbf{x}_t$ is generated as:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}),$$

where $\bar{\alpha}_t$ is a cumulative product of noise schedule coefficients. As t increases, the clean image $\mathbf{x}_0$ becomes increasingly corrupted by noise, with $\mathbf{x}_T$ representing pure Gaussian noise.

2. UNet Architecture The noisy input $\mathbf{x}_t$ at timestep t is passed to a UNet architecture, which is tasked with predicting the noise $\boldsymbol{\epsilon}$ that was added to $\mathbf{x}_0$. The UNet consists of:

- **Contracting Path (Encoder):** Reduces the spatial resolution while extracting high-level features.
- **Bottleneck:** A compressed representation of the noisy image.
- **Expanding Path (Decoder):** Upsamples the features back to the original resolution.
- **Skip Connections:** Connections between encoder and decoder layers to preserve fine-grained spatial details. Concatenation between output of encoder and decoder at each timestamp t.

The UNet outputs the predicted noise $\boldsymbol{\epsilon}_{\phi_t}(\mathbf{x}_t)$, where ϕ_t are the model parameters at timestamp t.

3. Time Representation t The timestep t represents the current point in the forward diffusion process and determines the level of noise in $\mathbf{x}_t$. The network must understand the timestep t to correctly predict the noise. Since neural networks cannot directly interpret the scalar t, the timestep is transformed into a suitable **time representation**:

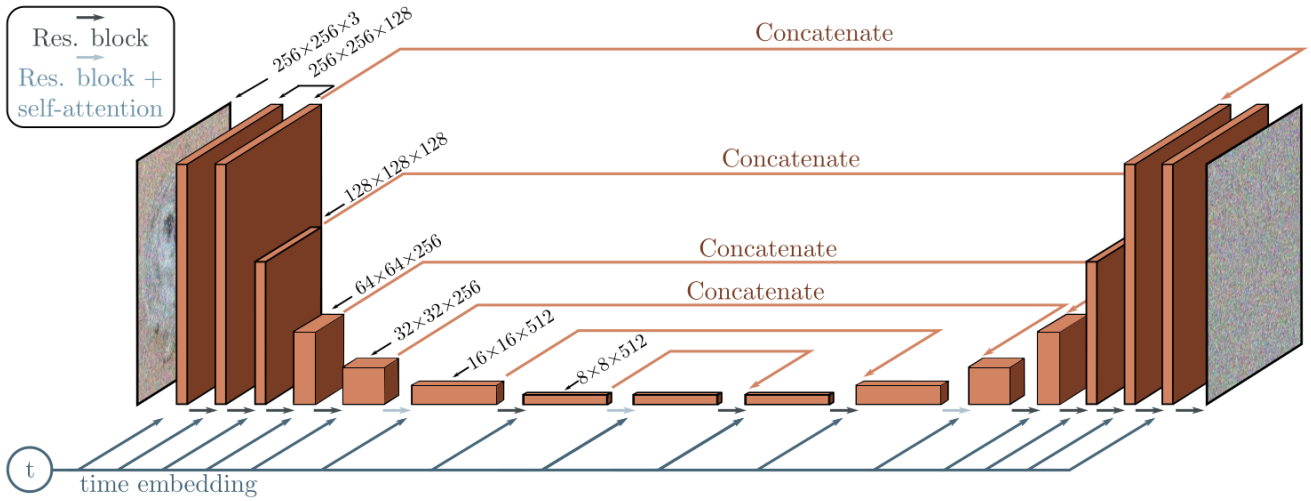
- **Sinusoidal Embeddings:** A common method where t is encoded using sinusoidal functions such as sine and cosine:
$$\text{Time\_embedding}(t) = \left[\sin\left(\frac{t}{10000^{2i/d}}\right), \cos\left(\frac{t}{10000^{2i/d}}\right)\right], \quad i = 0, \ldots, d/2,$$
where d is the dimensionality of the embedding.
- **Fully-Connected Layers (As shown in the image):** The scalar timestep t is passed through a series of fully-connected layers to produce a high-dimensional vector. This vector serves as a *time embedding* that conditions the model on the timestep.

The time embedding is injected into the UNet at various stages (dotted arrows in the image) to provide the network with information about the level of noise in the input $\mathbf{x}_t$. This allows the UNet to adapt its predictions based on the timestep t.

4. Denoising Prediction The UNet takes the noisy input $\mathbf{x}_t$ and the time embedding t as inputs. It outputs the predicted noise:

$$\boldsymbol{\epsilon}_{\phi_t}(\mathbf{x}_t).$$

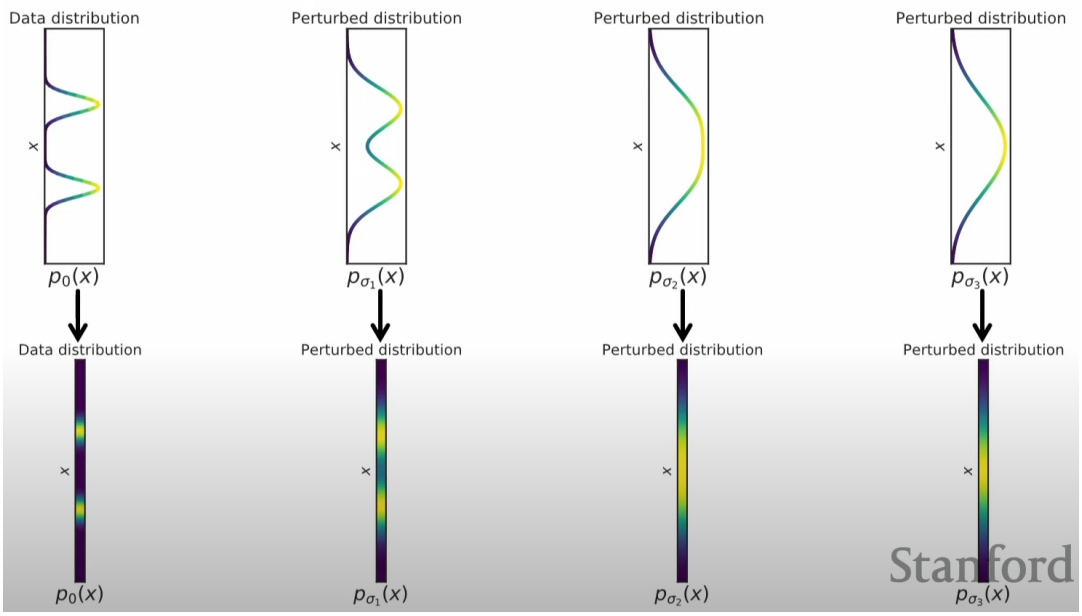
This predicted noise is used to denoise $\mathbf{x}_t$ during the backward process, which iteratively removes noise to recover the clean image $\mathbf{x}_0$.



U-Net as used in diffusion models for images. The network aims to predict the noise that was added to the image. It consists of an encoder which reduces the scale and increases the number of channels and a decoder which increases the scale and reduces the number of channels. The encoder representations are concatenated to their partner in the decoder. Connections between adjacent representations consist of residual blocks, and periodic global self-attention in which every spatial position interacts with every other spatial position. A single network is used for all time steps, by passing a sinusoidal time embedding through a shallow neural network and adding the result to the channels at every spatial position at every stage of the U-Net.

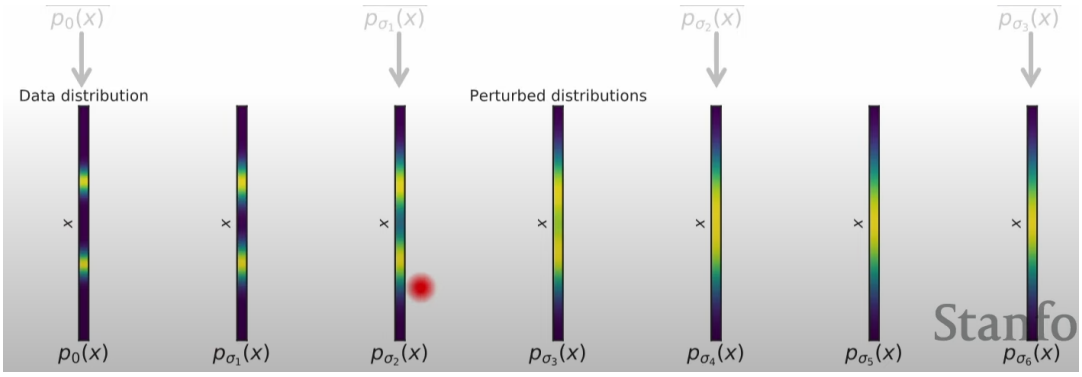
7.2 Infinite Noise Levels Score-based Modeling

We have seen in score-models that

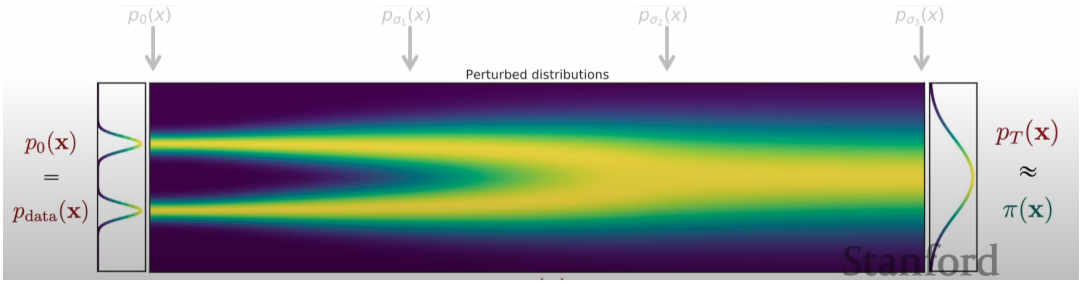


<<< → Noise level → <<<

What if we use multiple noise levels that are in between 0 and σ_1 or between σ_1 and σ_2 and ... ?



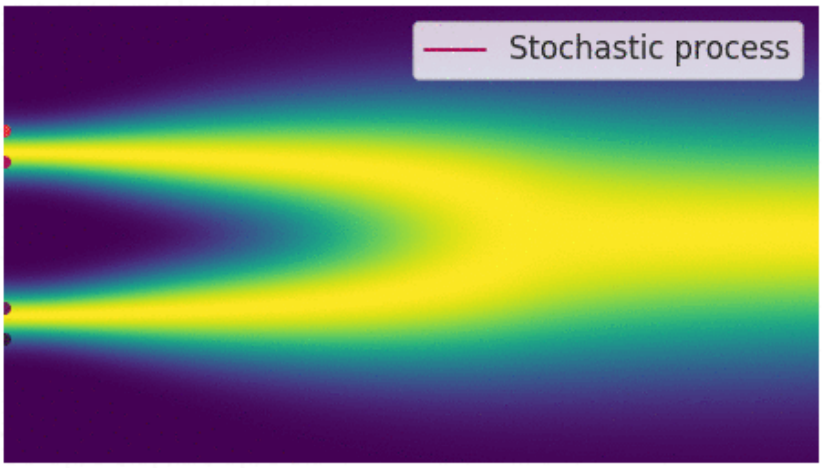
We can formulate diffusion models via infinite noise levels score-based models by treating the forward diffusion process as a continuous-time stochastic process or basically what happens if we consider an infinite number of noise levels?



$p_t(\mathbf{x})$, $t \in [0, T]$

7.2.1 Perturbing data with stochastic processes

As previously discussed, incorporating multiple noise scales is crucial for the success of score-based generative models. By extending the number of noise scales to infinity we achieve several benefits, including higher quality samples, exact log-likelihood computation, and controllable generation for solving inverse problems.



Link to Animation

When the number of noise scales approaches infinity, we essentially perturb the data distribution with continuously growing levels of noise. In this case, the noise perturbation procedure is a continuous-time stochastic process, as demonstrated above.

7.2.2 Stochastic Differential Equations (SDEs)

A stochastic differential equation (SDE) provides a concise way to represent a stochastic process. The general form of an SDE is:

$$d\mathbf{x}_t = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w},$$

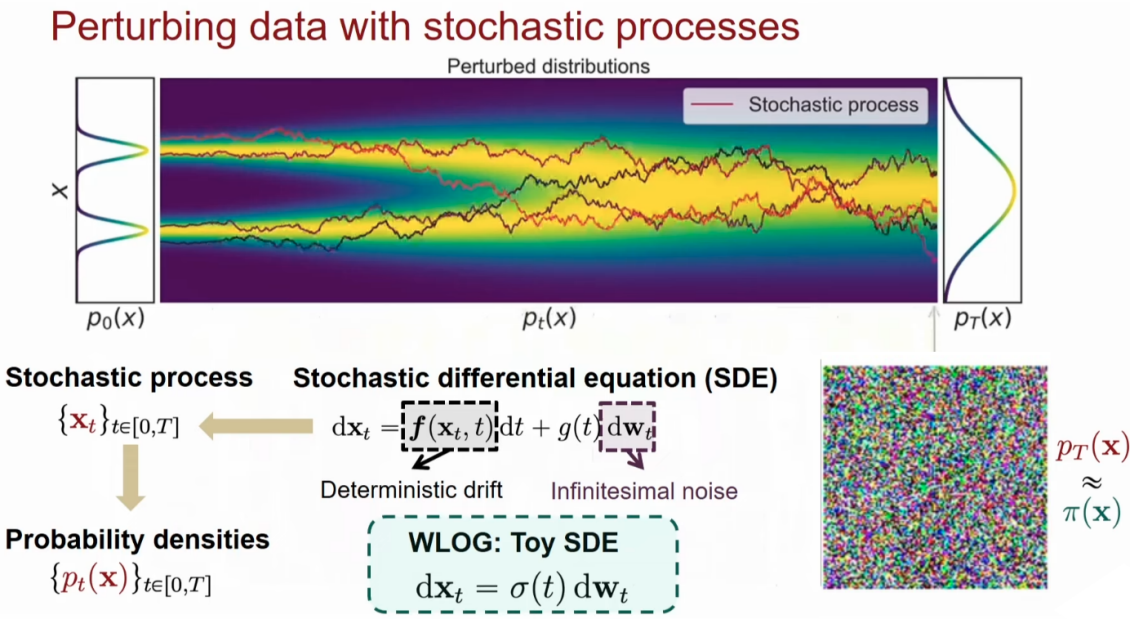
where:

- $d\mathbf{x}_t$ denotes infinitesimal changes in x during time interval dt
- $\mathbf{f}(\mathbf{x}, t)$ is the drift coefficient that describes deterministic dynamics,
- $g(t)$ is the diffusion coefficient that determines the amount of noise,
- $d\mathbf{w}$ represents the standard Brownian motion.
- $d\mathbf{w}$ denotes infinitesimal white noise.

The solution to an SDE generates a collection of random variables $\{\mathbf{x}(t)\}_{t\in[0,T]}$, describing the evolution of the data over time.

7.2.3 Encoder:

Instead of describing the relationship among z_t and z_{t-1} via Encoders, we can describe the relationship via SDE. Which is basically a way to describe how the values of these random variables $\mathbf{x}_t$ or $\mathbf{z}_t$ that are indexed by a continuous time variable t are related to each other.



From left to right, noise is progressively added, starting from pure data and eventually reaching pure noise. You can imagine what happens when you perturb the data with increasing levels of stochastic noise:

- Initially, the data points are distributed according to $p_0(\mathbf{x})$, which represents the original data density.
- As noise is continuously added, the distribution transitions to $p_T(\mathbf{x})$, which corresponds to pure noise.

This process can be represented as a collection of random variables $\{\mathbf{x}_t\}$, where $t \in [0, T]$. These random variables describe how the data evolves as noise is added over time, with t serving as the "time" axis. The stochastic

process $\{\mathbf{x}_t\}_{t\in[0,T]}$ is characterized by its associated probability densities $\{p_t(\mathbf{x})\}_{t\in[0,T]}$, which describe the data distributions at each time step.

The dynamics of this process are governed by the following stochastic differential equation (SDE):

$$d\mathbf{x}_t = \underbrace{\mathbf{f}(\mathbf{x}_t, t)\, dt}_{\text{deterministic drift}} + \underbrace{\mathbf{g}(t)\, d\mathbf{w}_t}_{\text{stochastic noise}} ,$$

where:

- $\mathbf{f}(\mathbf{x}_t, t)$ represents the deterministic drift component. $f(x,t) = -1/2(\beta(t)x)$
- $\beta(t)$ determines the decaying rate of a signal and the magnitude of the noise.
  e.g. Linear: $\beta(t) = \beta_{\min} + (\beta_{\max} - \beta_{\min}).t$
- $\mathbf{g}(t)\, d\mathbf{w}_t$ represents the stochastic noise, with $d\mathbf{w}_t$ being an infinitesimal Wiener process increment.
- $g(t) = \sqrt{\beta(t)}$ is the diffusion coefficient

For simplicity, we often use a toy SDE where the drift term is omitted, and only noise is considered:

$$d\mathbf{x}_t = \sigma_t d\mathbf{w}_t,$$

where σ_t is a time-dependent noise coefficient. This equation models a random walk where noise $\sigma_t d\mathbf{w}_t$ is added at each time step. After sufficiently many steps, the data transitions from its original distribution $p_0(\mathbf{x})$ to a noise distribution $p_T(\mathbf{x})$, which approximates pure noise.

In essence, the SDE describes the evolution of particles moving from left to right following deterministic dynamics while adding a small amount of noise at each time step. The key insight is that if the goal is to transition from data $p_0(\mathbf{x})$ to pure noise $p_T(\mathbf{x})$, this process can be effectively modeled using a stochastic differential equation.

7.2.4 Types of SDEs and Noise Evolution

Stochastic differential equations (SDEs) are used to describe the evolution of random variables over time with a combination of deterministic and stochastic components. Below, we outline various forms of SDEs and their applications in modeling noise evolution:

1. **General SDE / Variance Preserving SDE (VP-SDE):**

$$d\mathbf{x}_t = \underbrace{\mathbf{f}(\mathbf{x}_t, t)\, dt}_{\text{deterministic drift}} + \underbrace{\mathbf{g}(t)\, d\mathbf{w}_t}_{\text{stochastic noise}} ,$$

   where:

   - $\mathbf{f}(\mathbf{x}_t, t)$ represents the deterministic drift term.
   - $\mathbf{g}(t)\, d\mathbf{w}_t$ represents the stochastic noise term, with $d\mathbf{w}_t$ being an infinitesimal increment of a Wiener process.

   This is the most general form of an SDE and accounts for both deterministic evolution and stochastic noise.

2. **Toy SDE / Variance Exploding SDE (VE-SDE):**

$$d\mathbf{x}_t = \sigma_t d\mathbf{w}_t,$$

   where:

   - The deterministic drift term $\mathbf{f}(\mathbf{x}_t, t)$ is omitted, leaving only the stochastic noise component.
   - σ_t is a time-dependent noise scale that determines the variance of the noise.

   This simplified SDE is often used to study noise evolution without the complexity of deterministic contributions, but the variance increases over time.

3. **Discrete Noise Scales:**

   Discrete noise scales $\sigma_1 < \sigma_2 < \cdots < \sigma_L$ provide a finite-step approximation of the continuous-time SDE. Instead of adding infinitesimally small noise increments, noise is introduced at discrete steps, approximating the continuous process. This method is commonly used in numerical solutions, such as Euler-Maruyama discretization, for practical computation.

   While discrete noise scales are not true SDEs, they approximate the behavior of continuous SDEs when the time steps are sufficiently small. This connection bridges practical computational methods with the theoretical framework of continuous SDEs.

4. **Hand-Designed SDE:** Extending the idea of discrete noise scales, continuous SDEs replace finite steps with a smooth function σ_t that evolves over time. For example:

$$d\mathbf{x} = e^t d\mathbf{w},$$

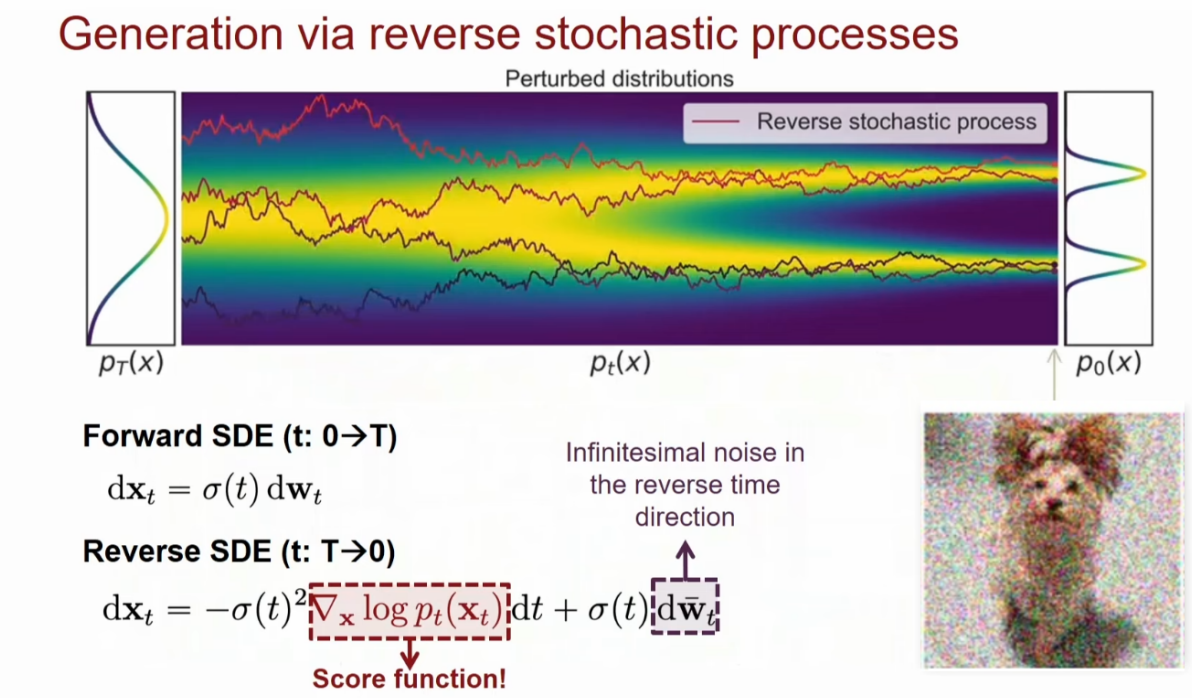
   where:

   - The noise scale is explicitly defined as $\sigma_t = e^t$, meaning the noise variance grows exponentially with time.
   - This is a specific example of a toy SDE where σ_t is designed for a particular application or modeling need.

   Hand-designed SDEs are tailored for specific scenarios by choosing σ_t based on the desired noise properties.

7.2.5 Decoder (Reverse Stochastic Processes):

VE-SDE:



What happens if we reverse the direction of time?

We start with a stochastic process that evolves over time from left to right, transitioning from the data distribution $p_0(\mathbf{x})$ (pure data) to $p_T(\mathbf{x})$ (pure noise). Now, we want to reverse this process and move from $p_T(\mathbf{x})$ back to $p_0(\mathbf{x})$, effectively generating samples. This reversal involves inverting the time direction of the stochastic process.

It turns out that there exists a simple stochastic differential equation (SDE) that describes this reverse-time process. The reverse SDE requires access to the **score functions** of the noise-perturbed data densities $p_t(\mathbf{x})$, where $t \in [0, T]$. If we can approximate these score functions, we can describe the reverse-time stochastic process as:

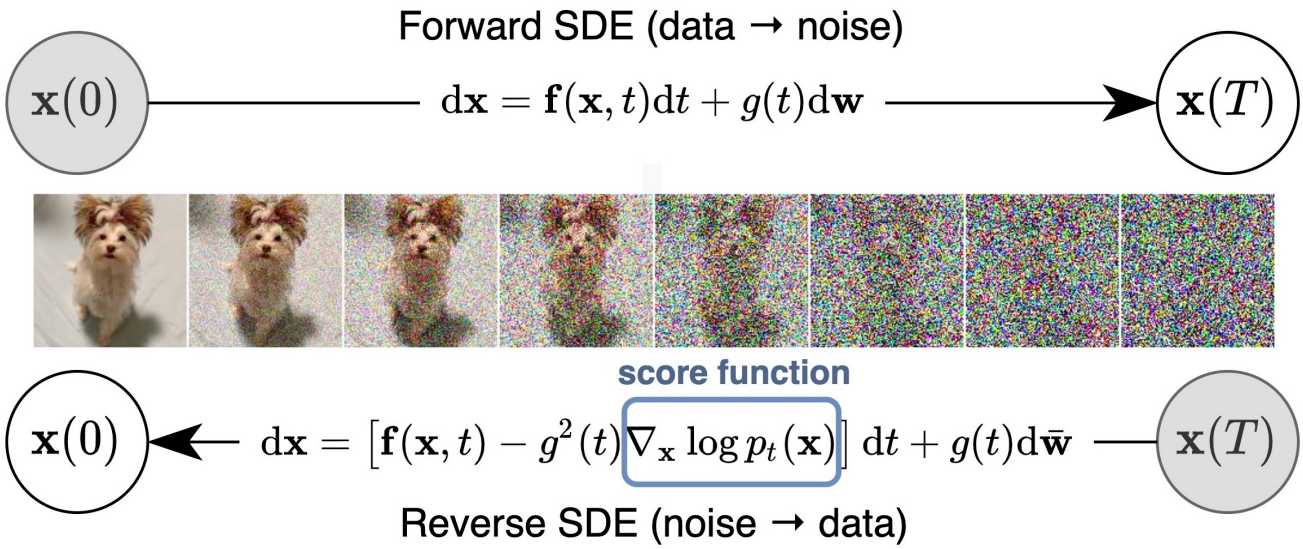
$$d\mathbf{x}_t = -\sigma_t^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) dt + \sigma_t d\bar{\mathbf{w}}_t,$$

$$\sigma(t) = \sigma_{\min}(\frac{\sigma_{\max}}{\sigma_{\min}})^t$$

where:

- σ_t is the noise scaling coefficient,
- $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)$ is the score function,
- $d\bar{\mathbf{w}}_t$ represents infinitesimal noise added in the reverse-time direction.

VP-SDE:



The forward SDE describes how data evolves from $p_0(\mathbf{x})$ to $p_T(\mathbf{x})$ (pure noise):

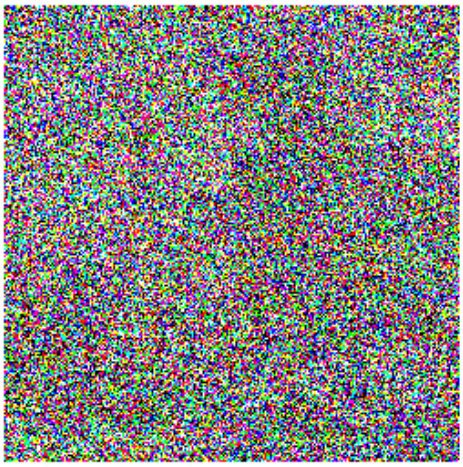
$$d\mathbf{x}_t = f(x, t)dt + g(t)dw$$

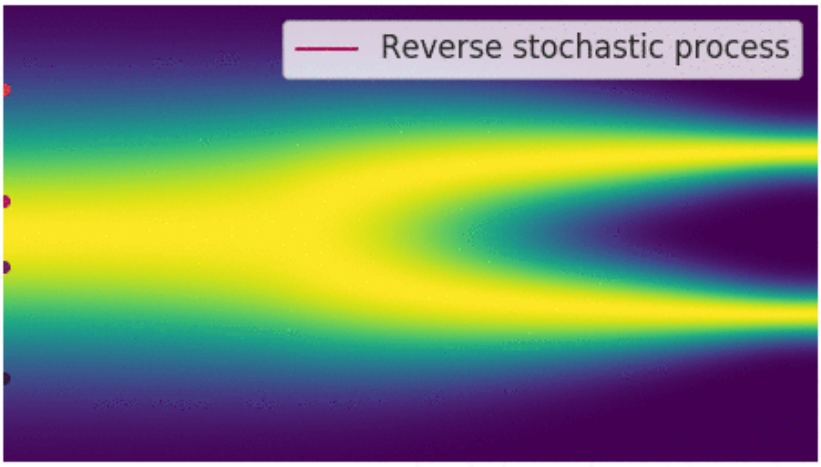
The reverse SDE allows us to generate data by reversing the perturbation process:

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] dt + g(t)\, d\bar{\mathbf{w}}$$

7.2.6 Role of the Score Function

The score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ plays a critical role in the reverse SDE. It represents the gradient of the log-probability density function of the noise-perturbed data at time t. If we can approximate these score functions using a score-based model, we can construct a generative model that transitions from noise $p_T(\mathbf{x})$ to data $p_0(\mathbf{x})$.





Link to Animation

8 Score-Based Generative Modeling via SDEs

8.1 Time-Dependent Score-Based Model

The goal is to approximate the score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, which represents the gradient of the log-probability density of the noise-perturbed data at time t. We achieve this using a neural network $\mathbf{s}_\theta(\mathbf{x}, t)$ (parameterized by θ):

$$\mathbf{s}_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x}),$$

where:

- $\mathbf{s}_\theta(\mathbf{x}, t)$: A Neural network approximating the score function.
- t: Time variable representing the noise scale; $t \in [0, T]$, where T corresponds to infinite noise.
- $p_t(\mathbf{x})$: Probability density of noise-perturbed data at time t.

8.2 Training Objective

The score-based model is trained by minimizing the weighted denoising score matching loss:

$$\mathbb{E}_{t\sim\mathcal{U}(0,T)} \left[\lambda(t) \mathbb{E}_{p_t(\mathbf{x})} \left[\|\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x}, t)\|_2^2 \right] \right],$$

where:

- $\lambda(t)$: A weighting function to balance the loss at different noise scales.
- $t \sim \mathcal{U}(0, T)$: Time sampled uniformly between 0 and T.
- $p_t(\mathbf{x})$: Noise-perturbed data distribution at time t.

8.3 Reverse-Time SDE

Given the learned score function $\mathbf{s}_\theta(\mathbf{x}, t)$, we can define the reverse-time stochastic differential equation (SDE) to transition from pure noise $p_T(\mathbf{x})$ to data $p_0(\mathbf{x})$:

$$d\mathbf{x} = -\sigma_t^2 \mathbf{s}_\theta(\mathbf{x}, t)dt + \sigma_t d\bar{\mathbf{w}},$$

where:

- σ_t: Noise magnitude at time t.
- $\mathbf{s}_\theta(\mathbf{x}, t)$: Neural network approximating the score function.
- $d\bar{\mathbf{w}}$: Infinitesimal Brownian motion noise in the reverse-time direction.

8.4 Euler-Maruyama Approximation

To numerically solve the forward and reverse-time SDEs, we use the Euler-Maruyama method, analogous to the Euler method for ordinary differential equations (ODEs), which discretizes time. The update rules at each time step Δt for both VE-SDE and VP-SDE are given below.

VE-SDE (Variance Exploding SDE):

Forward SDE:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \sigma_t \sqrt{\Delta t}\, \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}),$$

Reverse SDE:

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t - \sigma_t^2\, \mathbf{s}_\theta(\mathbf{x}_t, t)\, \Delta t + \sigma_t \sqrt{|\Delta t|}\, \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}),$$

VP-SDE (Variance Preserving SDE):

Forward SDE:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + f(\mathbf{x}_t, t)\Delta t + g(t)\sqrt{\Delta t}\, \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}),$$

Reverse SDE:

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t + \left[f(\mathbf{x}_t, t) - g^2(t)\, \mathbf{s}_\theta(\mathbf{x}_t, t)\right] \Delta t + g(t)\sqrt{|\Delta t|}\, \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}),$$

where:

- Δt: Step size in time.
- $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$: Standard Gaussian noise.
- σ_t: Noise scale for VE-SDE at time t.
- $f(\mathbf{x}, t)$: Drift function in VP-SDE.
- $g(t)$: Diffusion coefficient in VP-SDE.
- $\mathbf{s}_\theta(\mathbf{x}, t)$: Score function estimated by the neural network.

This is basically similar to Langevin dynamics where we follow the gradient and add a little bit of noise at each time step. Which is discretization of the reverse SDE.

8.5 Summary of Terms

- $\mathbf{s}_\theta(\mathbf{x}, t)$: Time-dependent score function approximator (a neural network).
- $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$: True score function of the noise-perturbed data density.
- $t \in [0, T]$: Time variable where $t = 0$ corresponds to pure data and $t = T$ corresponds to pure noise.
- σ_t: Noise scale that increases over time.
- $d\mathbf{w}$: Forward-time Brownian motion increment.
- $d\bar{\mathbf{w}}$: Reverse-time Brownian motion increment.

9 Reverse Process: Sampling from the Reverse SDE

9.1 Where Does Δt Come From?

The time step Δt arises from discretizing the continuous reverse-time SDE into discrete steps using numerical methods, such as the Euler-Maruyama method. The reverse SDE:

$$d\mathbf{x} = -\sigma_t^2 \mathbf{s}_\theta(\mathbf{x}, t)dt + \sigma_t d\bar{\mathbf{w}}$$

is continuous over time $t \in [0, T]$. To solve it numerically, we divide the time interval into N equal steps:

$$t_0 = 0,\ t_1 = \Delta t,\ t_2 = 2\Delta t,\ \ldots,\ t_N = T,$$

where:

$$\Delta t = \frac{T}{N}.$$

How to Choose Δt:

- **Accuracy vs. Computation:**
  - Smaller Δt: Reduces discretization error but increases computational cost.
  - Larger Δt: Reduces computation but introduces discretization error.
- **Empirical Choice:** A common choice is $N = 1000$, giving:

$$\Delta t = \frac{T}{1000}.$$

- **Noise Schedule Sensitivity:** If σ_t (noise scale) changes rapidly, Δt should be smaller for stability.

9.2 Sampling Step-by-Step

To solve the reverse SDE numerically, we use the Euler-Maruyama method with a discretized time step Δt. Below, we outline the iterative sampling process for both VP-SDE and VE-SDE over three steps.

VP-SDE Sampling

The reverse SDE is given by:

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t + \left[f(\mathbf{x}_t, t) - g^2(t)\,\mathbf{s}_\theta(\mathbf{x}_t, t)\right]\Delta t + g(t)\sqrt{|\Delta t|}\,\mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}).$$

Here is an illustration over three iterations:

- **Initialization (at $t = T$):**

$$\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I}).$$

- **Iteration 1 ($t = T \to t - \Delta t$):**

$$\mathbf{x}_{T-\Delta t} = \mathbf{x}_T + \left[f(\mathbf{x}_T, T) - g^2(T)\,\mathbf{s}_\theta(\mathbf{x}_T, T)\right]\Delta t + g(T)\sqrt{\Delta t}\,\mathbf{z}_1.$$

- **Iteration 2 ($t - \Delta t \to t - 2\Delta t$):**

$$\mathbf{x}_{T-2\Delta t} = \mathbf{x}_{T-\Delta t} + \left[f(\mathbf{x}_{T-\Delta t}, T-\Delta t) - g^2(T-\Delta t)\,\mathbf{s}_\theta(\mathbf{x}_{T-\Delta t}, T-\Delta t)\right]\Delta t + g(T-\Delta t)\sqrt{\Delta t}\,\mathbf{z}_2.$$

- **Iteration 3 ($t - 2\Delta t \to t - 3\Delta t$):**

$$\mathbf{x}_{T-3\Delta t} = \mathbf{x}_{T-2\Delta t} + \left[f(\mathbf{x}_{T-2\Delta t}, T-2\Delta t) - g^2(T-2\Delta t)\,\mathbf{s}_\theta(\mathbf{x}_{T-2\Delta t}, T-2\Delta t)\right]\Delta t + g(T-2\Delta t)\sqrt{\Delta t}\,\mathbf{z}_3.$$

VE-SDE Sampling

The reverse SDE is given by:

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t - \sigma_t^2 \, \mathbf{s}_\theta(\mathbf{x}_t, t)\Delta t + \sigma_t \sqrt{\Delta t}\, \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}).$$

Here is an illustration over three iterations:

- **Initialization (at $t = T$):**
$$\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I}).$$
- **Iteration 1 ($t = T \to t - \Delta t$):**
$$\mathbf{x}_{T-\Delta t} = \mathbf{x}_T - \sigma_T^2 \, \mathbf{s}_\theta(\mathbf{x}_T, T)\Delta t + \sigma_T \sqrt{\Delta t}\, \mathbf{z}_1.$$
- **Iteration 2 ($t - \Delta t \to t - 2\Delta t$):**
$$\mathbf{x}_{T-2\Delta t} = \mathbf{x}_{T-\Delta t} - \sigma_{T-\Delta t}^2 \, \mathbf{s}_\theta(\mathbf{x}_{T-\Delta t}, T - \Delta t)\Delta t + \sigma_{T-\Delta t} \sqrt{\Delta t}\, \mathbf{z}_2.$$
- **Iteration 3 ($t - 2\Delta t \to t - 3\Delta t$):**
$$\mathbf{x}_{T-3\Delta t} = \mathbf{x}_{T-2\Delta t} - \sigma_{T-2\Delta t}^2 \, \mathbf{s}_\theta(\mathbf{x}_{T-2\Delta t}, T - 2\Delta t)\Delta t + \sigma_{T-2\Delta t} \sqrt{\Delta t}\, \mathbf{z}_3.$$

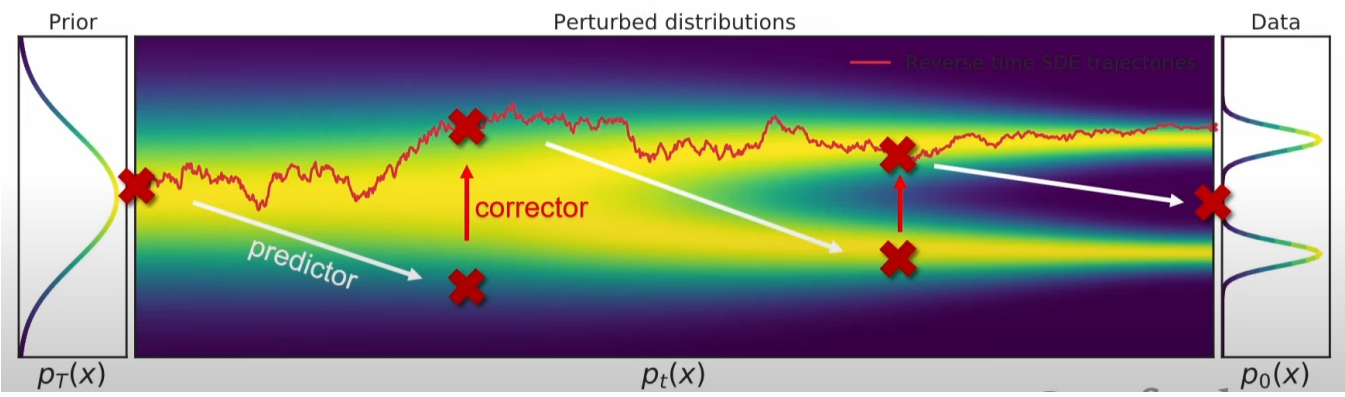
Summary

- The reverse process starts from pure noise $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$.
- At each step, the score function $\mathbf{s}_\theta(\mathbf{x}_t, t)$ pulls $\mathbf{x}_t$ towards the data manifold while noise $\mathbf{z}_k \sim \mathcal{N}(0, \mathbf{I})$ adds stochasticity.
- Smaller Δt improves accuracy but increases computational cost.
- As $t \to 0$, the noise decreases, and $\mathbf{x}_t$ converges to a sample from the data distribution $p_0(\mathbf{x})$.

10 Predictor-Corrector Sampling Methods

Predictor-Corrector sampling is a hybrid sampling technique used in generative models based on stochastic differential equations (SDEs). It combines two complementary components to improve sample quality:

1. **Predictor:** Advances the sample forward in time using a numerical solver for the SDE (e.g., Euler-Maruyama or other ODE solvers).
2. **Corrector:** Refines the predicted sample using score-based Markov Chain Monte Carlo (MCMC) techniques to better align with the data distribution.



10.1 Predictor Step (Numerical SDE Solver)

In the Predictor-Corrector approach, the predictor step numerically solves the reverse-time SDE during sampling. The reverse SDE:

$$\mathrm{d}\mathbf{x} = \left[f(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] \mathrm{d}t + g(t)\, \mathrm{d}\bar{\mathbf{w}},$$

is discretized using the **Euler-Maruyama method**:

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t + \left[f(\mathbf{x}_t, t) - g^2(t)\, \mathbf{s}_\theta(\mathbf{x}_t, t)\right] \Delta t + g(t)\sqrt{|\Delta t|}\, \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}),$$

where:

- $f(\mathbf{x}, t)$: Drift term (deterministic component).
- $g(t)$: Diffusion coefficient (scale of noise).
- $\mathbf{w}$: Standard Wiener process.
- $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$: Gaussian noise.
- Δt: Discrete time step.
- $\mathbf{s}_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ is the score function.

Here:

- $f(x_t, t)$ governs the deterministic drift.
- $g(t)\sqrt{\Delta t}\, z$ adds stochastic noise, where $z \sim \mathcal{N}(0, I)$ is standard Gaussian noise.

This step provides an initial "prediction" of the sample's trajectory by advancing the sample forward according to the SDE dynamics.

10.2 Corrector Step (Score-Based MCMC)

The corrector step refines the predicted sample to align it more closely with the target data distribution. This refinement relies on a **score function** $\nabla_x \log p_t(x)$, which is the gradient of the log-probability density of the data at time t.

If the score function is available, a common corrector technique is **Langevin Dynamics**, defined as:

$$x_{t+1} = x_t + \epsilon \nabla_x \log p_t(x_t) + \sqrt{2\epsilon}\, z, \quad z \sim \mathcal{N}(0, I),$$

where:

- $\nabla_x \log p_t(x_t)$ is the score function.
- ϵ is the step size for the MCMC update.
- $z \sim \mathcal{N}(0, I)$ adds stochastic noise for the MCMC update.

If the score function $\nabla_x \log p_t(x)$ is **not directly available**, models trained for **score estimation** (e.g., score-based generative models or denoising score-matching) are used to approximate it. These models provide an estimate of the score function based on the noisy sample x_t.

Without a reliable score estimate, the corrector step cannot refine the sample effectively, and the Predictor-Corrector approach may degrade to relying solely on the predictor step.

Additionally, using the discrete time means we might not have a Gaussian distribution during the denoising and technically our ELBO might not be tight. This would not be the case if we use infinite steps.

10.3 Combined Predictor-Corrector Sampling

The Predictor-Corrector method alternates between the predictor and corrector steps:

- The **Predictor step** advances the sample forward according to the SDE dynamics.
- The **Corrector step** refines the sample to align it more closely with the data distribution.

The combined approach balances deterministic updates (predictor) and probabilistic refinement (corrector), ensuring robust and high-quality sample generation.

10.4 Advantages of Predictor-Corrector Sampling

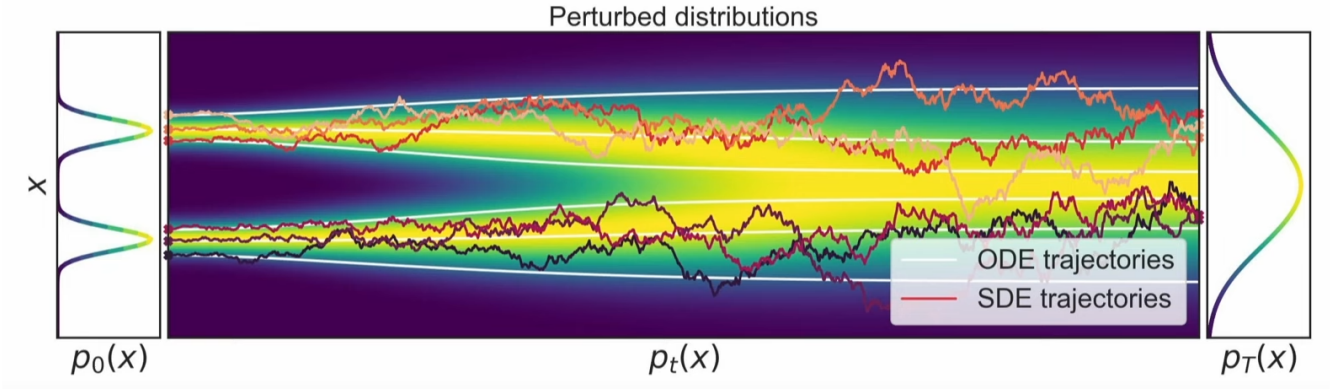
- **Improved Sample Quality:** The corrector step reduces errors introduced during the predictor step.
- **Combines Deterministic and Stochastic Updates:** The predictor provides a coarse update, while the corrector fine-tunes the sample to the target distribution.
- **Flexibility:** Different numerical solvers (e.g., Euler-Maruyama) and MCMC methods can be combined for various tasks.

10.5 Examples:

The sampling methods are also scalable for extremely high dimensional data. For example, it can successfully generate high fidelity images of resolution 1024×1024.

11 Probability Flow ODE

We can describe an stochastic process with exact same marginal densities but purely deterministic. Stochastic generative models, including high-quality score-based methods such as Langevin MCMC and SDEs, can generate samples by iteratively refining noise using the score function. However, an alternative and computationally efficient formulation exists in the form of **Probability Flow Ordinary Differential Equations (ODEs)**.



$$\underbrace{d\mathbf{x}_t = \sigma(t)d\mathbf{w}_t}_{\text{SDE}} \longrightarrow \underbrace{\frac{dx}{dt} = \frac{1}{2}g(t)^2\nabla_x \log p_t(x)}_{\text{ODE}}$$

ODE describes a **deterministic process** that data the data evolves smoothly over time based on the drift term

f(x,t).

- Offers an efficient alternative to Langevin MCMC and SDE-based sampling.
- Allows for exact likelihood computation and controllable sample generation.
- No noise is explicitly added during either the forward or reverse process.

For any SDE of the form:

$$dx = f(x,t)\,dt + \underbrace{g(t)\overbrace{dw}^{noise}}_{\text{stochastic term}},$$

the corresponding ODE becomes:

$$\frac{dx}{dt} = f(x,t) - \underbrace{\frac{1}{2}g(t)^2\nabla_x \log p_t(x),}_{\text{deterministic term}}.$$

where:

- $f(x,t)$: Drift term from the SDE.
- $p_t(x)$: Time-dependent probability density.
- $\nabla_x \log p_t(x) \approx s_\theta(x,t)$: Score function.(i.e., the gradient of the log-probability density.)
- $-\frac{1}{2}g(t)^2\nabla_x \log p_t(x)$ Captures how how the noise affects the probability density over time.

1. **Forward process:**
   - $$\frac{dx}{dt} = f(x,t) - \underbrace{\frac{1}{2}g(t)^2\nabla_x \log p_t(x),}_{\text{deterministic term}}.$$
   - Discretized Forward Process (Numerical Approximation):
   $$x_{t+\Delta t} = x_t + \Delta t \cdot \left(f(x_t,t) - \frac{1}{2}g(t)^2\nabla_x \log p_t(x_t)\right)$$
2. **Reverse process:**
   - $$\frac{dx}{dt} = -\left(f(x,t) - \frac{1}{2}g(t)^2\nabla_x \log p_t(x)\right)$$
   - Discretized Reverse Process (Numerical Approximation):
   $$x_{t-\Delta t} = x_t - \Delta t \cdot \left(f(x_t,t) - \frac{1}{2}g(t)^2\nabla_x \log p_t(x_t)\right)$$

11.1 Likelihood Computation

The probability flow ODE enables the computation of exact log-likelihoods under the generative process. This is based on the principle of mass conservation, expressed using the continuity equation. Below, we break down the key components.

1. Continuity Equation and Probability Current

The evolution of the probability density $p_t(x)$ is governed by the general continuity equation:

$$\frac{\partial p_t(x)}{\partial t} + \nabla_x \cdot \left(p_t(x) \cdot \frac{d\mathbf{x}}{dt} \right) = 0$$

1. $\frac{d\mathbf{x}}{dt}$ represents the **velocity field** — the direction and speed of movement at point x.
2. $p_t(x) \cdot \frac{d\mathbf{x}}{dt}$ is the **probability current**, indicating how much probability mass is flowing through space.

Velocity Field in General Case:

$$\frac{d\mathbf{x}}{dt} = f(x,t) - \frac{1}{2} g^2(t) \nabla_x \log p_t(x)$$

1. $f(x,t)$: Drift (deterministic part of the SDE)
2. $g(t)$: Diffusion scale
3. $\nabla_x \log p_t(x)$: Score function (direction of increasing density)

Intuition:

1. If $p_t(x) = 0.8$ and $\frac{d\mathbf{x}}{dt} = -1$, then a large amount of mass is flowing left.
2. If $p_t(x) = 0.01$, then very little mass flows, even if velocity is high.

2. Divergence of the Probability Current

Applying the divergence operator $\nabla_x \cdot (\cdot)$ to the probability current gives the net inflow or outflow of mass at a location:

- $\nabla_x \cdot (\cdot) > 0$: Mass is leaving → density decreases
- $\nabla_x \cdot (\cdot) < 0$: Mass is arriving → density increases
- $= 0$: Balanced flow → density remains constant

Analogy: Think of a small balloon at point x:

- Expanding means mass is flowing out (positive divergence)
- Shrinking means mass is flowing in (negative divergence)

3. Special Case — Deterministic Probability Flow ODE

When the dynamics are purely deterministic (e.g., for sampling with score-based models), we drop the stochastic term and define the flow by a drift field $f(x,t)$. In that case, the continuity equation simplifies to:

$$\frac{\partial p_t(x)}{\partial t} + \nabla_x \cdot (p_t(x) f(x,t)) = 0$$

Instantaneous Change in Log-Density:

$$\frac{d}{dt} \log p_t(x_t) = -\nabla_x \cdot f(x_t, t)$$

Integrated Log-Likelihood:

$$\log p(x_0) = \log p(x_T) - \int_0^T \nabla_x \cdot f(x_t, t)\, dt$$

1. $p(x_T)$ is the terminal distribution (typically standard Gaussian).
2. The integral captures how the volume contracts or expands along the flow.

4. Using a Learned Score Function $\mathbf{s}_\theta$

In score-based generative models (e.g., denoising diffusion), the drift is defined via a learned score function:

$$f(x, t) = -\frac{1}{2}\sigma_t^2\, \mathbf{s}_\theta(x, t)$$

The log-likelihood then becomes:

$$\log p_\theta(\mathbf{x}_0) = \underbrace{\log p(\mathbf{x}_T)}_{\text{Prior (e.g., } \mathcal{N}(0,I))} \underbrace{- \frac{1}{2}\int_0^T \sigma_t^2 \,\text{trace}\left(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x}_t, t)\right)\, dt}_{\text{Change of volume (divergence)}}$$

1. The trace term measures how the learned flow reshapes volume.
2. This formula enables exact log-likelihood computation via integration over continuous dynamics.

5. Gradient vs. Divergence (Clarification)

- **Gradient** $\nabla_x f(x)$: Applied to scalar-valued functions. Produces a vector pointing in the direction of steepest ascent:

$$\nabla_x f(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

- **Divergence** $\nabla_x \cdot \mathbf{v}(x)$: Applied to vector-valued functions. Produces a scalar measuring net inflow/outflow:

$$\nabla_x \cdot \mathbf{v}(x) = \sum_{i=1}^{n} \frac{\partial v_i(x)}{\partial x_i}$$

In our setting, divergence is applied to the probability current to monitor how density changes over time.

6. Summary Table

Term	Meaning
$\frac{d\mathbf{x}}{dt}$	Velocity field: direction and speed of particle motion
$p_t(x) \cdot \frac{d\mathbf{x}}{dt}$	Probability current: flow of mass at point x
$\nabla_x \cdot (\cdot)$	Divergence: net outflow/inflow of mass
Positive divergence	Mass is leaving the region; density decreases
Negative divergence	Mass is arriving; density increases

7. Modern Perspective

Recent generative modeling approaches (e.g., Neural ODEs, Continuous Normalizing Flows) treat the transformation as a continuous dynamic system. This allows:

- Exact likelihood evaluation
- Continuous sampling without discretization artifacts
- Learnable, invertible mappings using the trace-of-Jacobian formula

11.2 Practical Considerations

1. **Numerical Solvers**: The probability flow ODE can be solved using standard numerical ODE solvers, such as Euler's method, Runge-Kutta, or higher-order solvers.
2. **Stability**: Proper handling of the likelihood term is necessary to avoid instability, especially when the noise level σ is low.

11.3 Conclusion

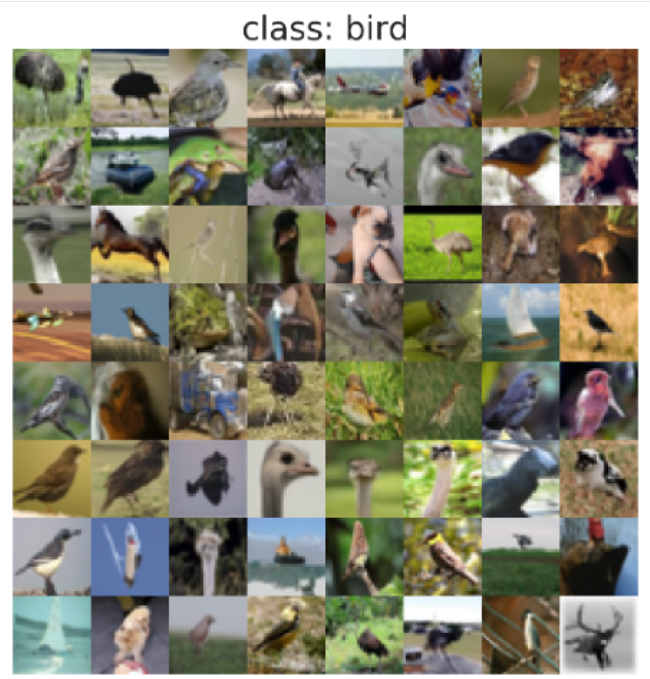
Probability flow ODEs provide a deterministic alternative to stochastic generative processes, enabling cleaner sample generation and exact likelihood computation. By incorporating likelihood gradients, they can also be adapted for solving inverse problems, making them a powerful tool for applications requiring controllable generation.

Table 1: Comparison between ODEs and SDEs in Generative Models

Aspect	**ODE**	**SDE**
Noise in Forward Process	Does not simulate the noisy forward process; instead, reparameterizes it as a deterministic reverse ODE matching the same marginals.	Noise is explicitly added at every step during the forward process (stochastic diffusion).
Noise in Backward Process	No noise added; purely deterministic backward integration using the learned score function.	Noise is added during reverse sampling, making it stochastic.
Number of Steps	Typically fewer steps are required since ODE solvers can use adaptive step sizes and avoid stochastic noise.	Requires more steps due to stochasticity, as noise introduces variability.
Sampling Difficulty	Simpler and deterministic; however, may struggle to fully capture multimodal or complex distributions.	More challenging due to randomness, but better captures distributional complexity.
Robustness	Susceptible to score estimation errors, since no stochasticity is present to smooth them out.	More robust due to stochasticity, which can average over errors in the score or drift.
Applications	Useful for exact likelihood computation and inverse problems (e.g., probability flow ODEs).	Preferred for generative models emphasizing stochastic behavior (e.g., DDPMs, SDE-based samplers).

11.4 Examples:

Class-conditional generation



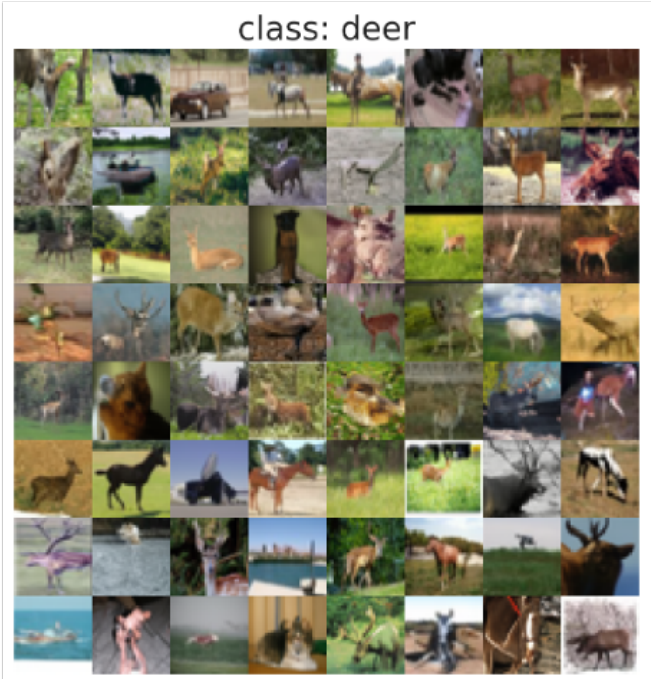


Image Painting

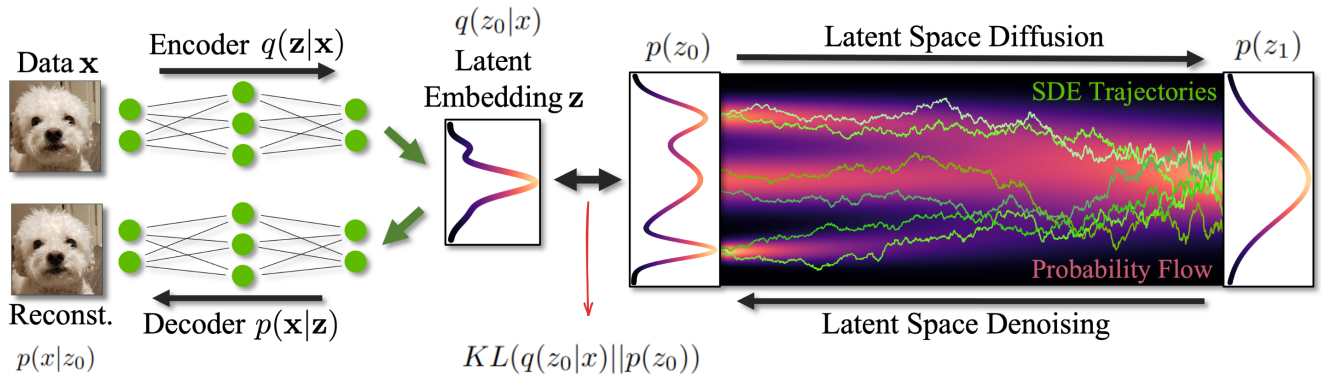
Image colorization

11.5 Techniques for Efficient ODE-Based Sampling:

- **Accelerated Sampling:** A method to speed up sampling by solving ODEs more efficiently.
- **Parallel ODE Solving:** Solving multiple ODEs simultaneously to enhance performance and scalability.
- **Distillation:** Reduces the complexity of sampling by approximating the ODE solutions.
- **Latent Diffusion Models:** Employ ODEs in a compressed latent space for efficient computation.
- **Stable Diffusion Models:** Utilize ODEs to ensure stability and high-quality sample generation in diffusion processes.

12 Latent Diffusion Models

The main difference in these models is that we are using latent diffusion model. Essentially what we are doing is to add an extra encoder and decoder layer at the beginning which are used to reduce the dimensionality of the data. Instead of having to do diffusion model over pixels, we train a diffusion model over the latent space of an VAE.

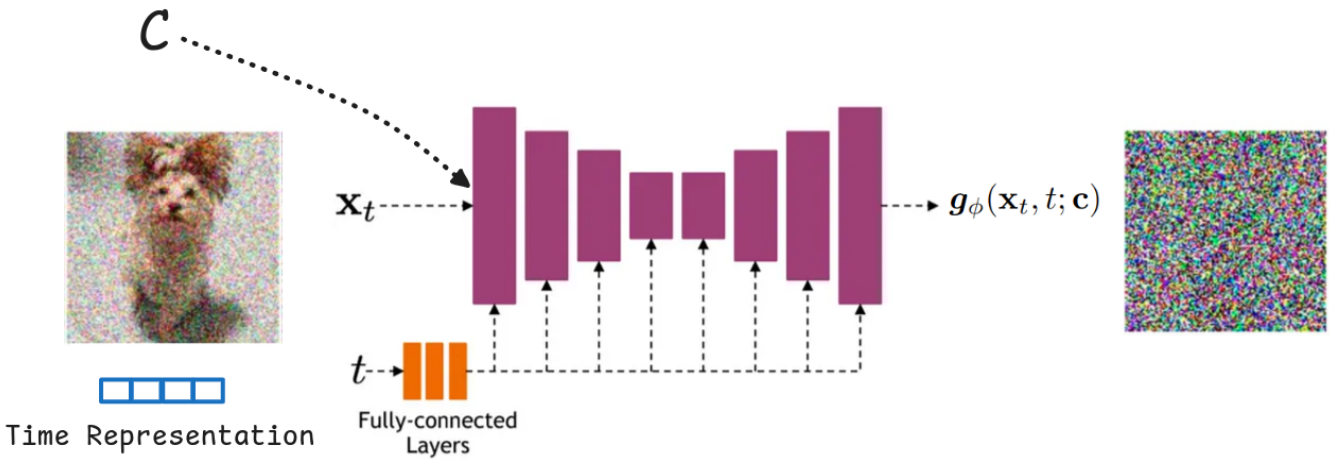


The main reason is that we get a lot faster diffusion training process over the smaller dimensional latent space. The other advantageous of this approach is that now we can use diffusion model for any data modality, including text. This requires that over VAE to do a good job.

13 Stable Diffusion

In this approach we pre-train the autoencoder (so its not trained during diffusion training) and we just focus on getting a good reconstruction via the diffusion model.First to work on large scale data and widely adopted by industry. Here for data generation the diffusion model acts as the prior distribution and hence the VAE doesn't need to have the distribution of the latent to be Gaussian.

14 Conditional Generation



If the data has associated labels c, these can be exploited to control the generation process. Leveraging such labels can improve generation results, as seen in GANs, and the same principle applies to diffusion models. Providing label information makes it easier to denoise an image, as the model has additional context about its content.

One approach to conditional synthesis in diffusion models is **classifier guidance**. This modifies the denoising update from $\mathbf{z}_t$ to $\mathbf{z}_{t-1}$ by incorporating class information c. The update can be expressed as:

$$\mathbf{z}_{t-1} = \hat{\mathbf{z}}_{t-1} + \sigma_t^2 \frac{\partial \log \Pr(c \mid \mathbf{z}_t)}{\partial \mathbf{z}_t} + \sigma_t \boldsymbol{\epsilon},$$

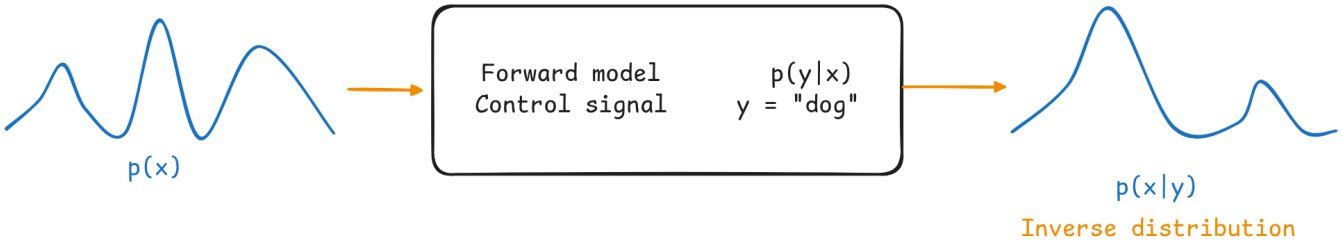
where:

- $\hat{\mathbf{z}}_{t-1}$ is the standard denoising step without class guidance.
- The term $\frac{\partial \log \Pr(c|\mathbf{z}_t)}{\partial \mathbf{z}_t}$ adjusts the update based on the gradient of the classifier with respect to the latent variable $\mathbf{z}_t$, encouraging the generation of samples more likely to belong to class c.
- $\sigma_t \boldsymbol{\epsilon}$ introduces stochasticity, where $\boldsymbol{\epsilon}$ is Gaussian noise.

We can train the score model conditioned on class label via:

$$\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p_{\text{data}}(\mathbf{x},\mathbf{c})} \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,\mathbf{I})} \mathbb{E}_{t\sim\mathcal{U}[0,T]} \left\| \underbrace{g_\phi(\mathbf{x}_t, t; \mathbf{c})}_{\text{Denoiser}} - \boldsymbol{\epsilon} \right\|_2^2$$

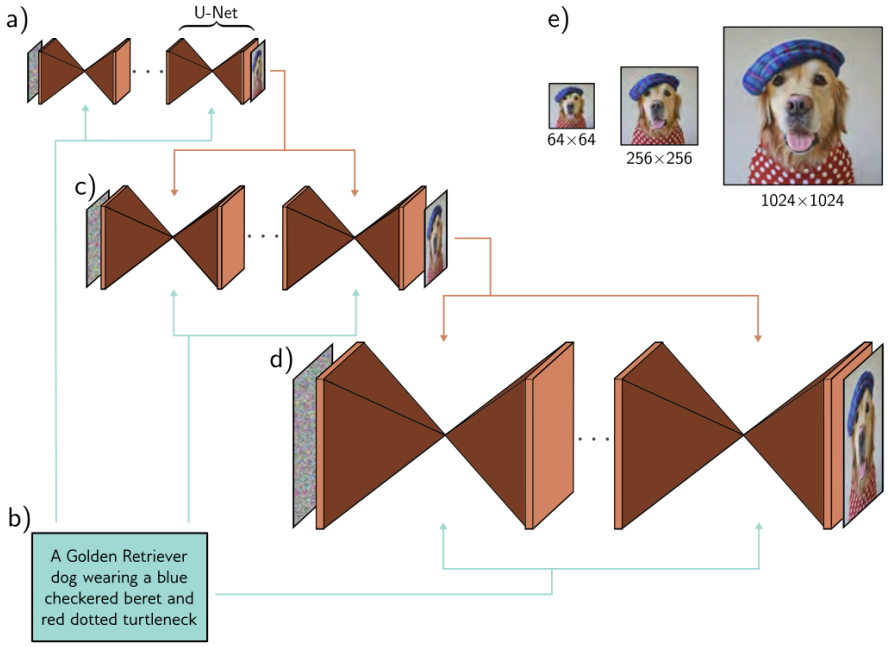
Classifier guidance can generate either conditional or unconditional samples, or any weighted combination of the two. A key advantage is that over-weighting the conditioning information often produces high-quality, albeit slightly stereotypical, examples. This behavior is analogous to truncation in GANs, where sampling from a narrower latent space improves sample quality at the expense of diversity.



To simplify the process, **classifier-free guidance** avoids the need for a separate classifier $\Pr(c \mid \mathbf{z}_t)$. Instead, class information c is incorporated directly into the main model $g_t[z_t, \phi_t, c]$ (estimated via score function) by adding an embedding of c to the network's layers. The model is trained jointly on conditional and unconditional objectives by randomly dropping the class information during training. This enables the model to generate both conditional and unconditional outputs flexibly at test time.

14.1 Hints for improving image quality

- **Estimating Variances in the Reverse Process:** Improving generation quality involves estimating both the means and variances (σ_t^2) of the reverse process. This is particularly beneficial when sampling with fewer steps.
- **Modifying Noise Schedules:** Adjusting the noise schedule in the forward process (e.g., allowing β_t to vary at each step) enhances the model's performance by fine-tuning the diffusion dynamics.
- **Using a Cascade of Diffusion Models for High-Resolution Images:** High-resolution images are generated by first creating a low-resolution image and progressively refining it through subsequent diffusion steps. The model conditions on the resized lower-resolution image, appending it to U-Net layers along with other contextual information such as class labels or captions.



14.2 Control the generation process

$$\text{Bayes' rule:} \quad p(x \mid y) = \frac{p(x)p(y \mid x)}{\underbrace{p(y)}_{untractable}}$$

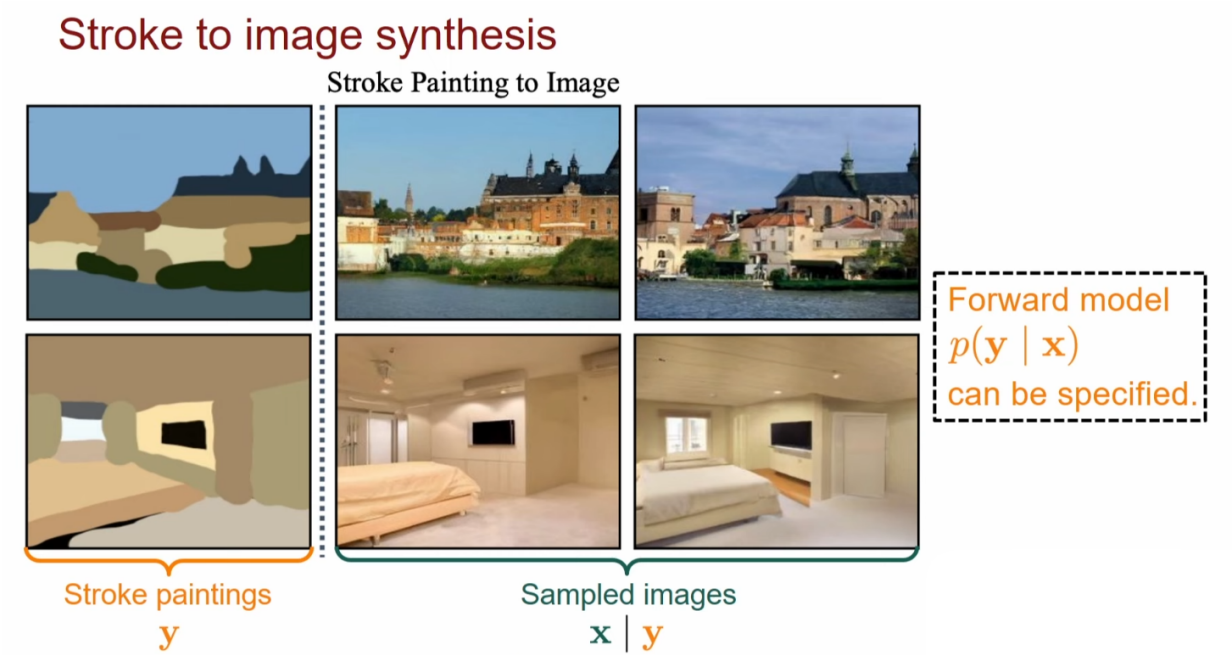
$$\text{Bayes' rule for score functions:} \quad \nabla_x \log p(x \mid y) = \nabla_x \log p(x) + \nabla_x \log p(y \mid x) - \underbrace{\nabla_x \log p(y)}_{=0}$$

$$\text{Unconditional score:} \quad \nabla_x \log p(x) \approx s_\theta(x)$$

$$\text{Bayes' rule for score functions:} \quad \nabla_x \log p(x \mid y) = s_\theta(x) + \nabla_x \log \underbrace{p(y \mid x)}_{\text{forward model}}$$

$p(y \mid x)$ is not learned explicitly in most cases. It is either:

- **Score-Based Approximation During Sampling:**
  The model assumes that $\nabla_x \log p(y \mid x)$ is available during sampling and adds this term explicitly to the unconditional score $\nabla_x \log p(x)$. The combined score becomes:
  $$\nabla_x \log p(x \mid y) \approx s_\theta(x) + \nabla_x \log p(y \mid x),$$
  where $s_\theta(x)$ approximates $\nabla_x \log p(x)$.
- **Approximation Using Domain-Specific Priors:**
  In cases where the forward process $p(y \mid x)$ is well understood (e.g., in inverse problems like MRI reconstruction), the conditional gradient $\nabla_x \log p(y \mid x)$ can be computed based on prior knowledge and incorporated directly during sampling:
  $$\nabla_x \log p(y \mid x) \propto \text{Domain-Specific Information (e.g., physics model).}$$
- **Simplified Score Matching (Classifier Free):**
  If $\nabla_x \log p(y \mid x)$ cannot be explicitly computed or approximated, the model may rely solely on the unconditional score:
  $$\nabla_x \log p(x \mid y) \approx s_\theta(x),$$
  effectively ignoring the contribution of $\nabla_x \log p(y \mid x)$.



Language-guided image generation

$\mathbf{y}$ $\mathbf{x} \mid \mathbf{y}$

(Prompt)
Treehouse in the style of Studio Ghibli animation

Forward model
$p(\mathbf{y} \mid \mathbf{x})$
is an image captioning neural network.

Controllable generation: Text-guided generation

An abstract painting of computer science:

A painting of the starry night by van Gogh

References

[1] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

[2] Stefano Ermon. *CS236*.

[3] Yang Song Generative Modeling by Estimating Gradients of the Data Distribution