Diffusion Models Beat GANs on Image Synthesis

High-Resolution Image Synthesis with Latent Diffusion Models

Robin Rombach[1] * Andreas Blattmann[1] * Dominik Lorenz[1] Patrick Esser[R] Björn Ommer[1]
[1]Ludwig Maximilian University of Munich & IWR, Heidelberg University, Germany [R]Runway ML

https://github.com/CompVis/latent-diffusion

GLIDE: Towards Photo

Text-Guided Diffusion Models

Alex Nichol * Prafulla Dhariwal * Aditya Ramesh * Pranav Shyam Pamela Mishkin Bob McGrew
Ilya Sutskever Mark Chen

t-Conditional
Image Generation with CLIP Latents

Aditya Ramesh*
OpenAI
aramesh@openai.com

Prafulla Dhariwal*
OpenAI
prafulla@openai.com

Alex Nichol*
OpenAI
alex@openai.com

Casey Chu*
OpenAI
casey@openai.com

Mark Chen
OpenAI
mark@openai.com

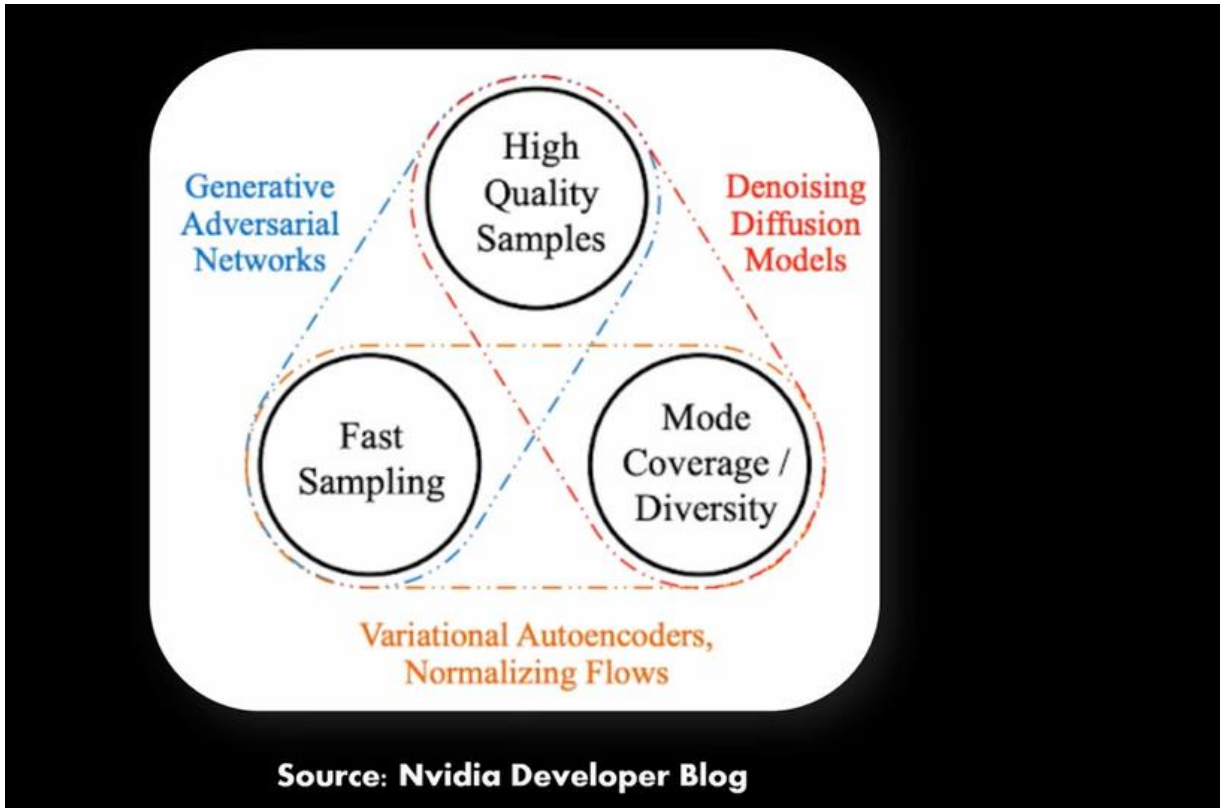
Abstract

Denoising Diffusion probabilistic model (DDPM)

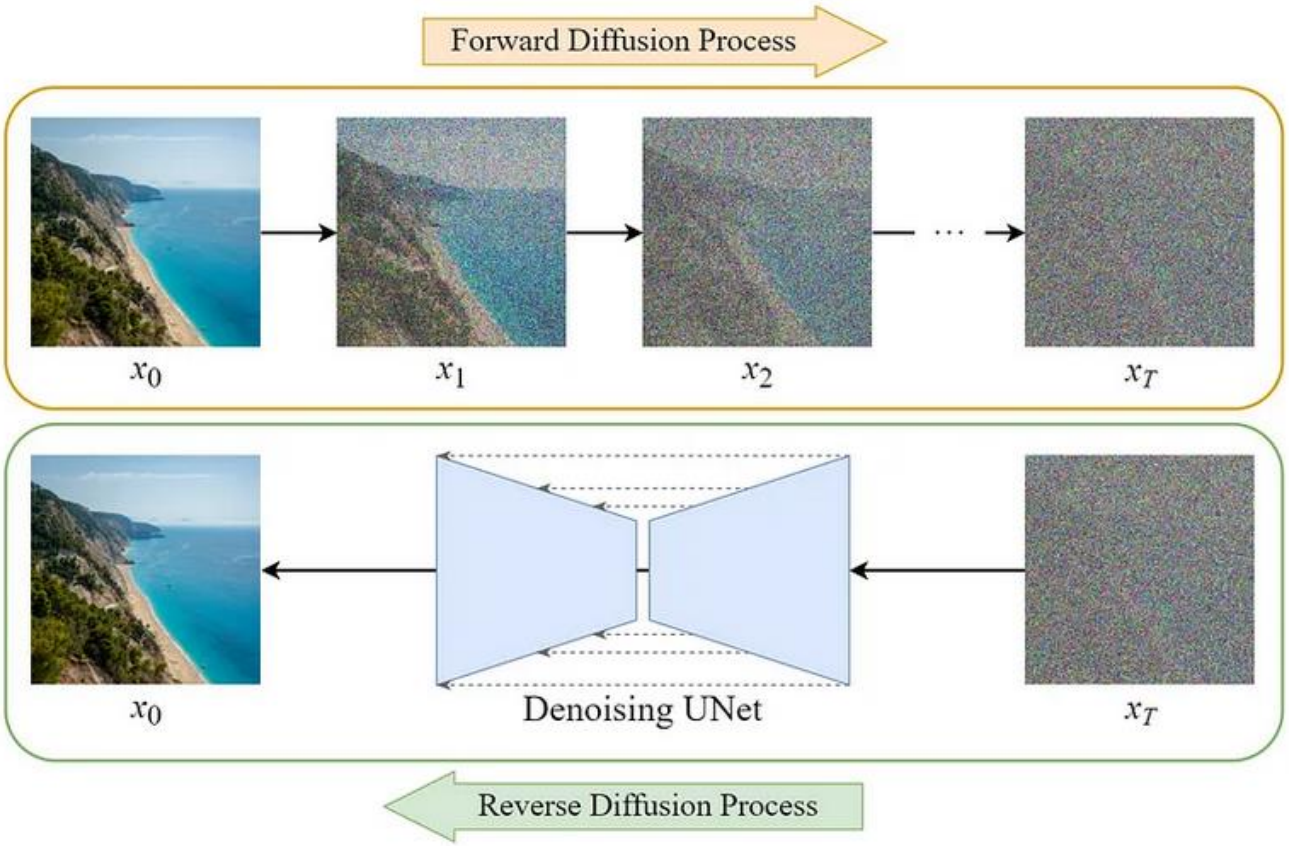
Resources:

1. https://medium.com/@steinsfu/diffusion-model-clearly-explained-cd331bd41166
2. https://theaisummer.com/diffusion-models/
3. https://stats.stackexchange.com/questions/600127/purpose-of-scaling-mean-by-sqrt1-beta-t-in-forward-diffusion-process
4. https://www.youtube.com/watch?v=zF5DU5bHTP0&list=PLd3hlSJsX_In7qup928HaHmilugBGctuF&index=4
5. https://www.youtube.com/watch?v=a4Yfz2FxXiY&t=40s
6. Code: https://github.com/dtransposed/code_videos/blob/main/Diffusion%20Model.ipynb
7. https://medium.com/@kemalpiro/step-by-step-visual-introduction-to-diffusion-models-235942d2f15c

Motivation


High Quality Samples
Generative Adversarial Networks
Denoising Diffusion Models
Fast Sampling
Mode Coverage / Diversity
Variational Autoencoders, Normalizing Flows
Source: Nvidia Developer Blog

Overview

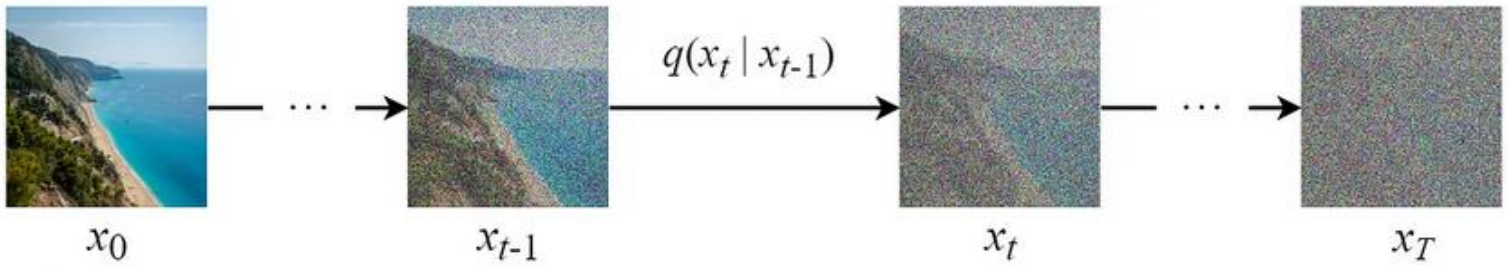


Overview of the Diffusion Model

The training of the Diffusion Model can be divided into two parts:

1. Forward Diffusion Process → add noise to the image.
2. Reverse Diffusion Process → remove noise from the image.

Forward Diffusion Process



Distribution of the noised images — Output — Mean μ_t — Variance Σ_t

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

Notations:

t : time step (from 0 to T)

x_0 : a data sampled from the real data distrition $q(x)$ (i.e. $x_0 \sim q(x)$)

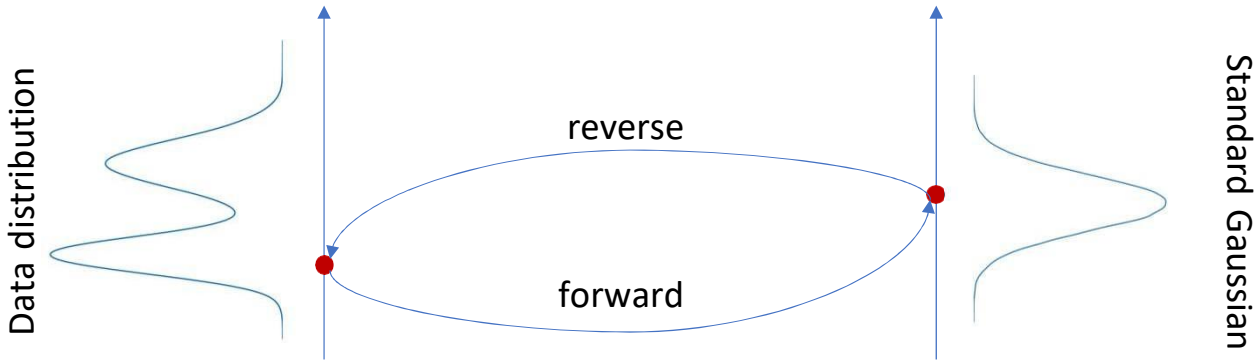
β_t : variance schedule ($0 \leq \beta_t \leq 1$, and β_0 = small number, β_T = large number)

I : identity matrix

Forward diffusion process

High-level overview

- Diffusion models are probabilistic models used for image generation
- They involve reversing the process of gradually degrading the data
- Consist of two processes:
  - **The forward process**: data is progressively destroyed by adding noise across multiple time steps
  - **The reverse process**: using a neural network, noise is sequentially removed to obtain the original data



High-level overview

- Three categories:
  - Denoising Diffusion Probabilistic Models (DDPM)
  - Noise Conditioned Score Networks (NCSN)
  - Stochastic Differential Equations (SDE)

Notations

$p(x_0)$ - data distribution

$\mathcal{N}(x; \mu, \sigma \cdot I)$ - Gaussian distribution

Random Variable (image)

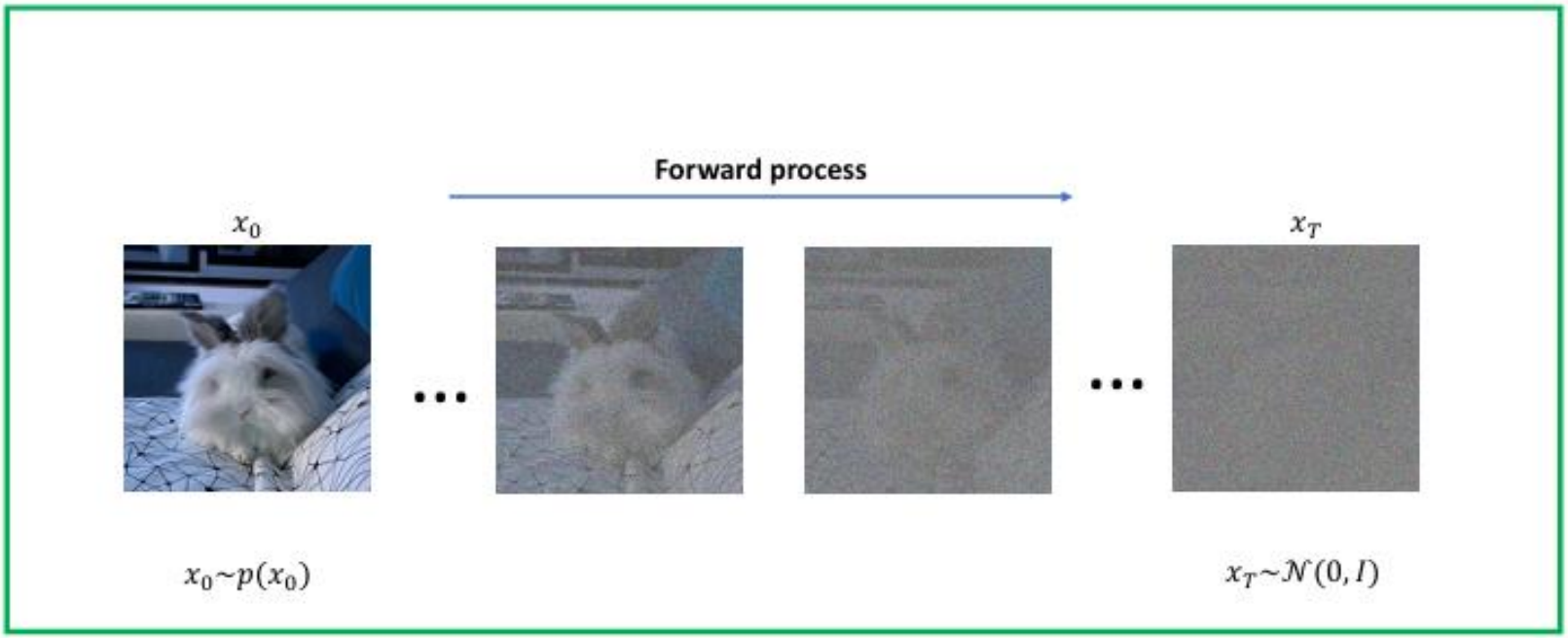
Mean Vector

Covariance matrix. I is the identity matrix

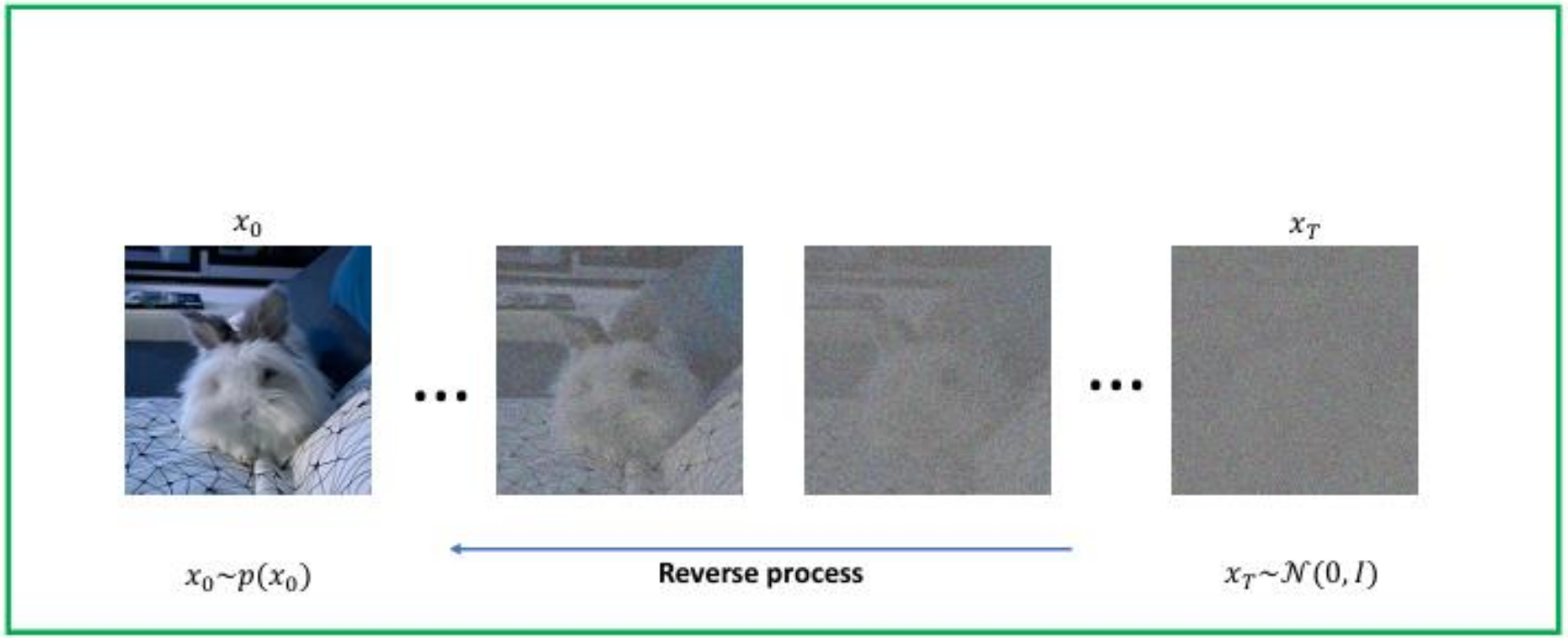
$$x = \mu + \sqrt{\sigma}\, z, \qquad z \sim \mathcal{N}(0, I)$$

Sample from this distribution (reparameterization trick)

Denoising Diffusion Probabilistic Models (DDPMs)



Denoising Diffusion Probabilistic Models (DDPMs)

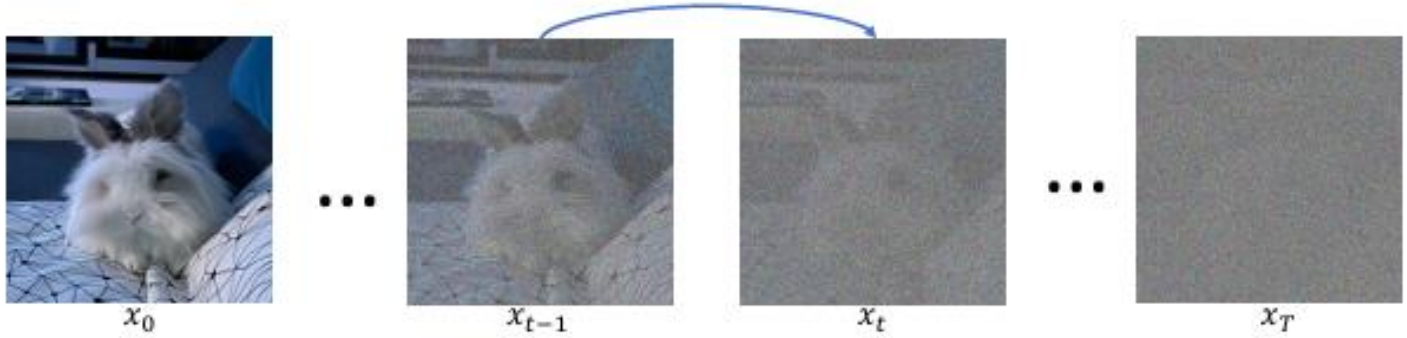


Denoising Diffusion Probabilistic Models (DDPMs)

Forward process (Iterative)

The image is replaced with noise

$$x_t \sim p(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t} \cdot x_{t-1}, \beta_t \mathrm{I}) \qquad \beta_t \ll 1 \, , t = \overline{1,T}$$



Denoising Diffusion Probabilistic Models (DDPMs)

Forward process. Ancestral sampling (One Shot)

$$x_t \sim p(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\hat{\beta}_t} \cdot x_0, (1 - \hat{\beta}_t)\mathrm{I})$$

Notations:

$$\hat{\beta}_t = \prod_{i=1}^{t} \alpha_i$$

$$\alpha_t = 1 - \beta_t$$

1. **Small steps:**
   - Decompose the image generation process(sampling) in many small steps “denoising” steps.
   - Small steps help to remove noise easily and accurately in backward process in order to produce high quality image.

2. **Context of Variance**:
   - In diffusion models, we often want to ensure that the variance of our data remains constant through each step of the process. This is important because if the variance increases with each step, the resulting vectors (or data points) can become excessively large or small, depending on the number of steps T taken. This could lead to instability in learning and generation.

- Assume we start with a random variable x_0 that follows a normal distribution: $x_0 \sim N(\mu, 1)$. This means that x_0 has a mean μ and a variance of 1 (normalized to unit variance).

3. **Initial Setup**:

Noise Addition Formula

3. **Step 1: Adding Noise**:
   - For the first step of the diffusion process (when $t = 1$), we scale the original data by a factor a and add Gaussian noise:

$$x_1 = ax_0 + \sqrt{\beta_1}\epsilon_1$$

   - Here, $\epsilon_1 \sim N(0, 1)$ is a random variable representing Gaussian noise with mean 0 and variance 1.

Calculating Variance

4. **Variance Calculation**:
   - To find the variance of x_1, we use the property that the variance of the sum of independent random variables is equal to the sum of their variances:

$$\mathrm{Var}(x_1) = \mathrm{Var}(ax_0 + \sqrt{\beta_1}\epsilon_1)$$

     This can be expanded as:

$$\mathrm{Var}(x_1) = \mathrm{Var}(ax_0) + \mathrm{Var}(\sqrt{\beta_1}\epsilon_1)$$

   - Since scaling a random variable by a constant scales its variance by the square of that constant:

$$\mathrm{Var}(x_1) = a^2\mathrm{Var}(x_0) + \beta_1\mathrm{Var}(\epsilon_1)$$

- Given that x_0 has unit variance (i.e., $\text{Var}(x_0) = 1$) and ϵ_1 also has unit variance (i.e., $\text{Var}(\epsilon_1) = 1$), we have:

$$\text{Var}(x_1) = a^2 + \beta_1$$

5. **Maintaining Unit Variance**:
   - To ensure that the variance remains equal to 1 at every step, we set up the equation:

$$a^2 + \beta_1 = 1$$

   - Solving for the scaling factor a gives us:

$$a = \sqrt{1 - \beta_1}$$

   - This means that to keep the variance unchanged after adding noise, we need to scale our original data by this factor.

Generalizing to Any Step t

6. **General Formula**:
   - The derived formula can be generalized for any step t in the diffusion process:

$$x_t = \sqrt{1 - \beta_t} x_{t-1} + \sqrt{\beta_t} \epsilon_t$$

   - Here, each term maintains the desired properties of variance and ensures that as we progress through steps, we do not unintentionally increase the overall variance.

7. **Probability Distribution**:
    - The final expression corresponds to a normal distribution where:

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

    - This indicates that given the previous state, the current state is normally distributed around a scaled version of it with added noise.

Conclusion

In summary, maintaining consistent variance throughout each step of the diffusion process is crucial for stability and effectiveness in training generative models. The parameter a adjusts how much of the original data is retained relative to how much noise is added, ensuring that despite multiple steps, the overall distribution characteristics remain stable and manageable.

https://www.youtube.com/watch?v=a4Yfz2FxXiY&t=1108s

- Farward pass/Noise Schedular: want to reach normalziation have zero mean and fix variance

Forward process

Can be pre-computed

$$q(x_t|x_{t-1}) = N\left(x_t; \sqrt{1-\beta}x_{t-1}, \beta_t I\right)$$

Diffusion models from scratch in PyTorch

Forward process →

Can be pre-computed

$$q(x_t|x_{t-1}) = N\left(x_t; \sqrt{1-\beta}x_{t-1}, \beta_t I\right)$$

→

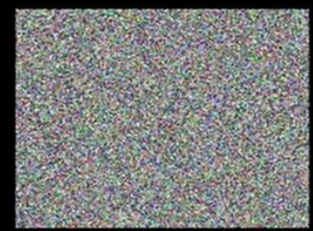
$$q(x_t|x_0) = N\left(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I\right)$$

T=200

Closed form forward process





...


β_0=0.0001 β_1=0.0002 β_2=0.0003 β_3=0.0004 β_{200}=0.02

α_0=0.9999 α_1=0.9998 α_2=0.9997 α_3=0.9996 α_4=0.98

$\bar{\alpha}_0$=0.9999 $\overline{\alpha_1}$=0.9997 $\overline{\alpha_2}$=0.9994 $\overline{\alpha_3}$=0.9990 $\overline{\alpha_4}$=0.1322

$$q(x_t|x_0) = N\left(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I\right)$$

https://medium.com/@steinsfu/diffusion-model-clearly-explained-cd331bd41166

Then we can expand it recursively to get the closed-form formula:

$$
\begin{aligned}
x_t &= \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \varepsilon_{t-1} \\
&= \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, \varepsilon_{t-1} \\
&= \sqrt{\alpha_t}\, \left(\sqrt{\alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_{t-1}}\, \varepsilon_{t-2} \right) + \sqrt{1-\alpha_t}\, \varepsilon_{t-1} \\
&= \sqrt{\alpha_t\, \alpha_{t-1}}\, x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\, \varepsilon_{t-2} + \sqrt{1-\alpha_t}\, \varepsilon_{t-1} \\
&= \sqrt{\alpha_t\, \alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_t\, \alpha_{t-1}}\, \bar{\varepsilon}_{t-2} \\
&\vdots \\
&= \sqrt{\alpha_t\, \alpha_{t-1}\, \dots\, \alpha_1}\, x_0 + \sqrt{1-\alpha_t\, \alpha_{t-1}\, \dots\, \alpha_1}\, \varepsilon \\
&= \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon
\end{aligned}
$$

How?

$$
\begin{aligned}
&\varepsilon_0, \dots, \varepsilon_{t-2}, \varepsilon_{t-1} \sim \mathcal{N}(0, I) \\
&\bar{\varepsilon}_0, \dots, \bar{\varepsilon}_{t-2}, \bar{\varepsilon}_{t-1} \sim \mathcal{N}(0, I) \\
&\varepsilon \sim \mathcal{N}(0, I) \\
&\alpha_t = 1 - \beta_t \\
&\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i
\end{aligned}
$$

Derivation of the closed-form formula

> *Note:*
> *all the ε are i.i.d. (independent and identically distributed) standard normal random variables.*
>
> *It is important to distinguish them using different symbols and subscripts because they are independent and their values could be different after sampling.*

$$
\begin{aligned}
x_t &= \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \varepsilon_{t-1} \\
&= \sqrt{\alpha_t}\, x_{t-1} + \sqrt{1-\alpha_t}\, \varepsilon_{t-1} \\
&= \sqrt{\alpha_t}\, \left(\sqrt{\alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_{t-1}}\, \varepsilon_{t-2} \right) + \sqrt{1-\alpha_t}\, \varepsilon_{t-1} \\
&= \sqrt{\alpha_t\, \alpha_{t-1}}\, x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\, \varepsilon_{t-2} + \sqrt{1-\alpha_t}\, \varepsilon_{t-1}
\end{aligned}
$$

$$
\varepsilon_0, \ldots, \varepsilon_{t-2}, \varepsilon_{t-1} \sim \mathcal{N}(0, I)
$$
$$
\bar{\varepsilon}_0, \ldots, \bar{\varepsilon}_{t-2}, \bar{\varepsilon}_{t-1} \sim \mathcal{N}(0, I)
$$
$$
\varepsilon \sim \mathcal{N}(0, I)
$$
$$
\alpha_t = 1 - \beta_t
$$
$$
\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i
$$

$X + Y$

Reparameterization Trick

$$
0 + \sqrt{\alpha_t(1-\alpha_{t-1})}\, \varepsilon_{t-2}
$$

Underlying Normal Distribution

$$
X \sim \mathcal{N}(0,\ \alpha_t(1-\alpha_{t-1})\, I)
$$

$$
0 + \sqrt{1-\alpha_t}\, \varepsilon_{t-1} \quad \dashrightarrow \quad Y \sim \mathcal{N}(0,\ (1-\alpha_t)\, I)
$$

Recall:

If $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$ $\quad Y \sim \mathcal{N}(\mu_Y, \sigma_Y^2)$

$Z = X + Y$

Then $Z \sim \mathcal{N}(\mu_X + \mu_Y,\ \sigma_X^2 + \sigma_Y^2)$

$$
Z \sim \mathcal{N}(0,\ \sigma_X^2 + \sigma_Y^2)
$$
$$
Z \sim \mathcal{N}(0,\ (1-\alpha_t\alpha_{t-1})\, I)
$$

$$
\begin{aligned}
\sigma_X^2 + \sigma_Y^2 &= \alpha_t(1-\alpha_{t-1}) + (1-\alpha_t) \\
&= \alpha_t - \alpha_t\alpha_{t-1} + 1 - \alpha_t \\
&= 1 - \alpha_t\alpha_{t-1}
\end{aligned}
$$

Reparameterization Trick

$$
0 + \sqrt{1-\alpha_t\alpha_{t-1}}\, \bar{\varepsilon}_{t-2}
$$

$$
= \sqrt{\alpha_t\, \alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_t\, \alpha_{t-1}}\, \bar{\varepsilon}_{t-2}
$$

$$
\vdots
$$

$$
= \sqrt{\alpha_t\, \alpha_{t-1}\, \ldots\, \alpha_1}\, x_0 + \sqrt{1-\alpha_t\, \alpha_{t-1}\, \ldots\, \alpha_1}\, \varepsilon
$$

$$
= \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon
$$

$\bar{\epsilon}_{t-2}$ represents the cumulative effect of noise added at the earlier step $t-2$, scaled according to the parameters of the diffusion process.

Reverse Process

1. **Overview**:
   - The reverse process begins with x_T (pure Gaussian noise) and attempts to generate x_0 (the original data/image) by reversing the forward diffusion process.
   - At each timestep t, the model predicts x_{t-1} from x_t.

3. **Goal**:
   - The neural network is trained to approximate the true reverse process, which is the posterior distribution:
   $$q(x_{t-1}|x_t, x_0).$$
   - The objective is to make the predicted mean (μ_θ) close to the actual mean of the posterior.

2. **Reverse Distribution:**

   - The reverse process $p_\theta(x_{t-1}|x_t)$ is modeled as a Gaussian distribution:

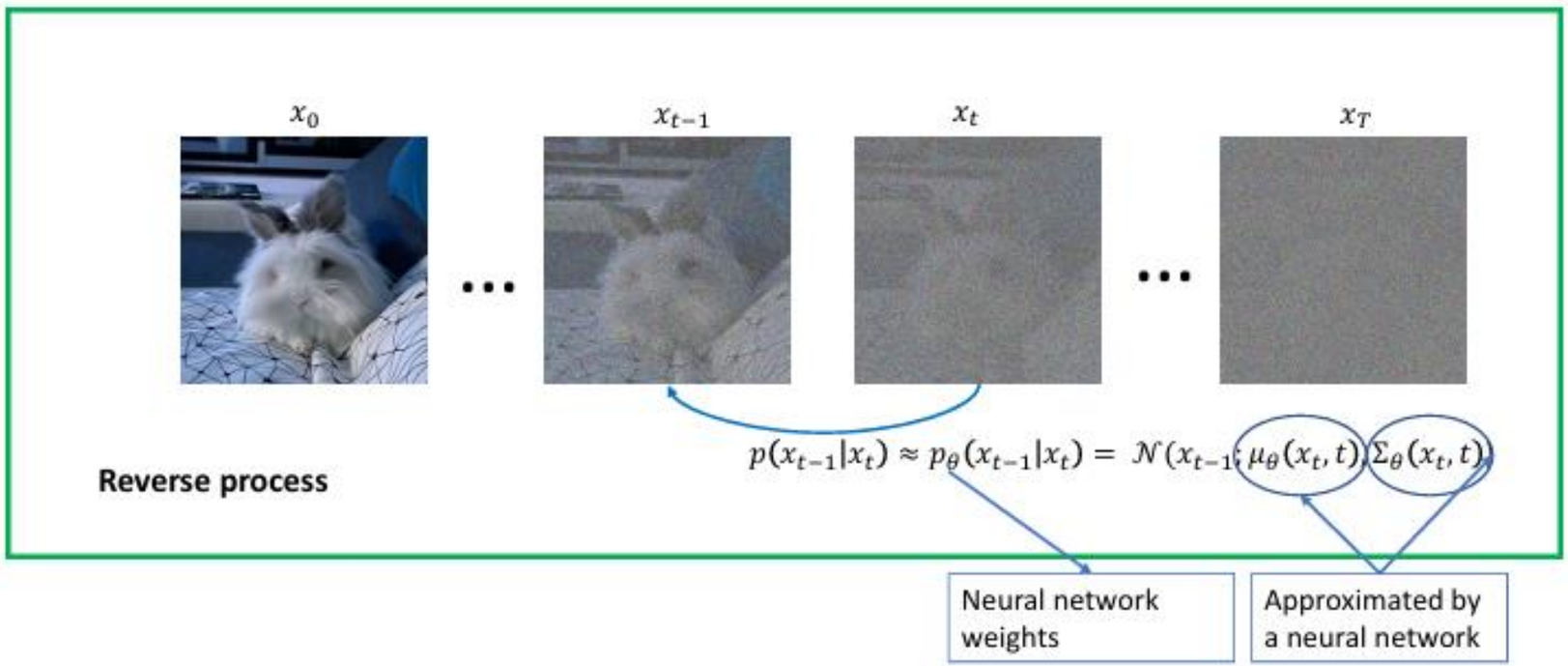
$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 \mathbf{I}).$$

   - The mean (μ_θ) is learned via a neural network, while the variance (σ_t^2) is often fixed or simplified to make the model easier to train.

Remember that:

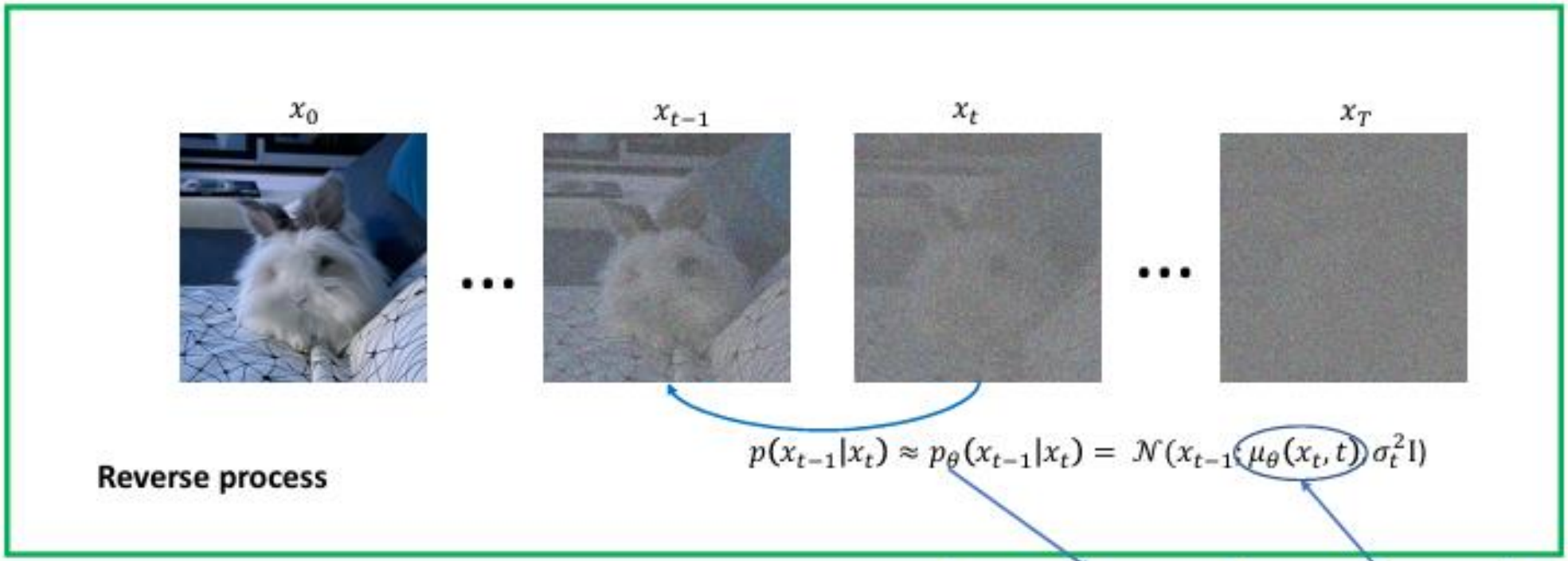
DDPMs. Training objective

Remember that:



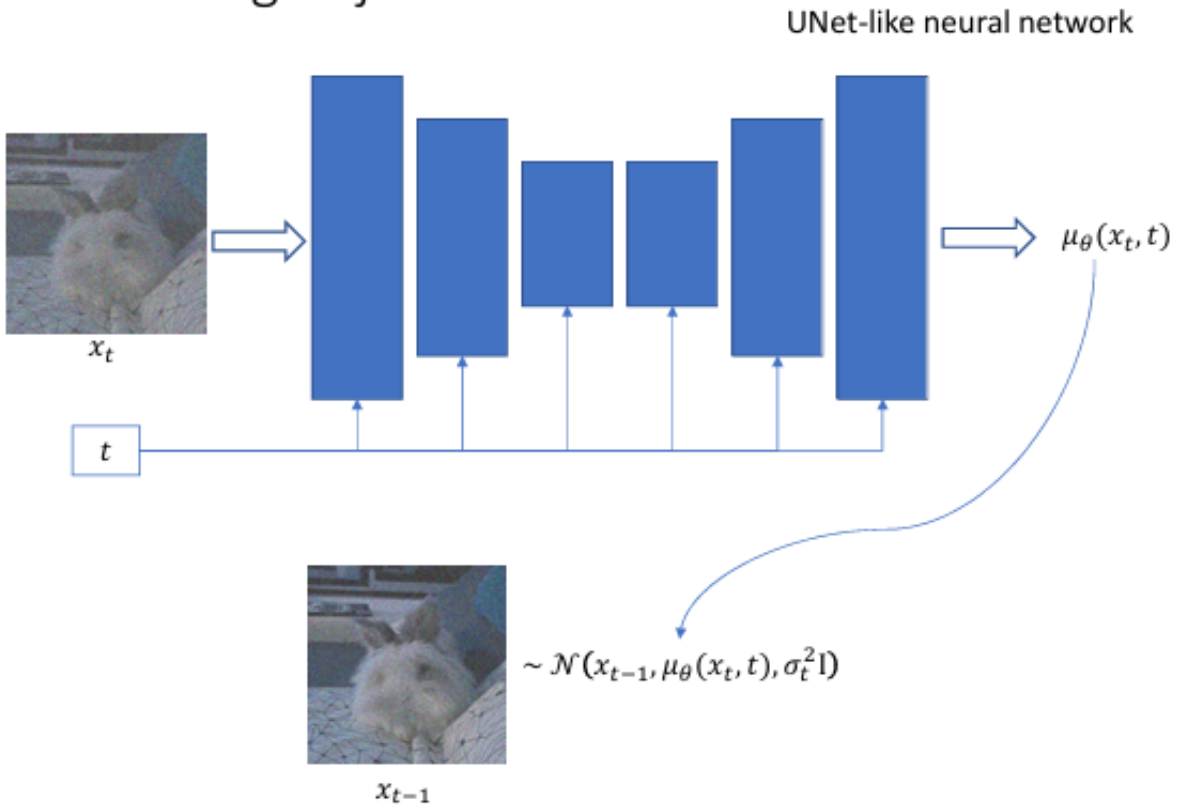
DDPMs. Training objective

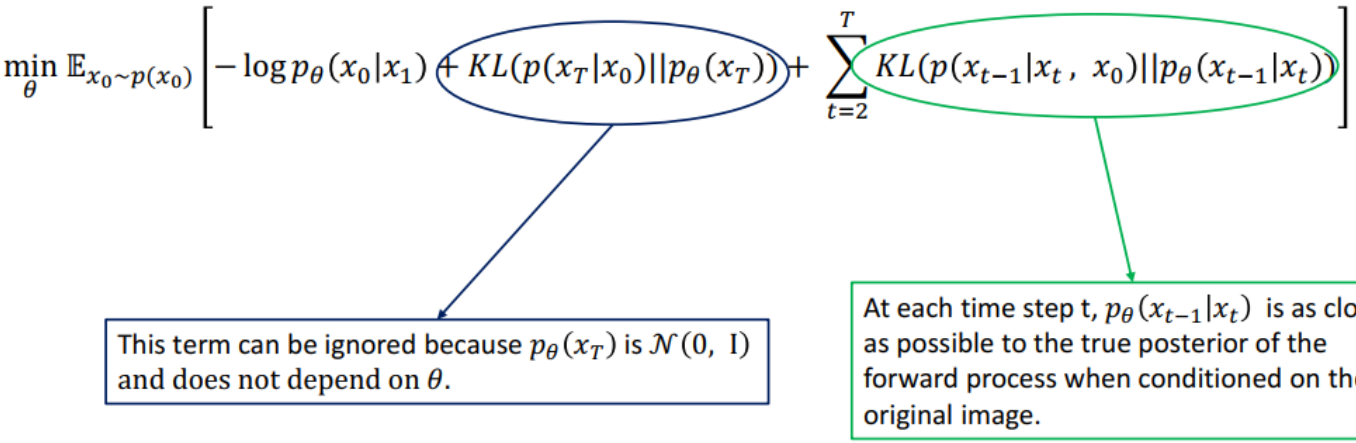
Simplification:



Fix the variance instead of learning, and predict/learn the mean

DDPMs. Training objective





Training Objective

The training objective is derived from the Evidence Lower Bound (ELBO) and consists of three main terms:

Term 1: −log pθ(x0|x1):

This measures how well the final denoised sample x1 can reconstruct the original data x0.

In many implementations, this term is often simplified or ignored because it focuses on the final step of the reverse proces

Term 2: KL(pθ(xT|x0)‖ pθ(xT))

This measures the divergence between the prior pθ(xT) and the distribution p(xT|x0)/ q(xT|x0) at the final timestep T.

Since pθ(xT) is often chosen as a standard Gaussian (N(0,I) and doesn't depend on the parameters θ, this term is typically ignored during training.

Term 3: ∑_(t=2)^(T) ⟦KL(p(xt-1|xt,x0)‖pθ(xt-1|xt))⟧

This measures the divergence between the true posterior p(xt−1|xt,x0)/ q(xt−1|xt ,x0) (from the forward process) and the learned reverse process pθ(xt−1|xt).

- **It ensures that the neural network learns a reverse process that closely mimics the true posterior.**
- **This is the primary term used for training and simplifies to a noise-prediction task.**

DDPMs. Training Objective. Simplifications

$$\min_{\theta} \mathbb{E}_{x_0 \sim p(x_0)} \left[-\log p_\theta(x_0|x_1) + \sum_{t=2}^{T} KL(p(x_{t-1}|x_t,\ x_0)||p_\theta(x_{t-1}|x_t)) \right]$$

- The KL divergences of 2 gaussians is L2 distance between their means
- The first term measures the reconstruction error and can be addressed with an independent decoder.
- DDPMs paper introduced two simplifications that led to a much simple objective that is based on the noise in the image.

DDPMs. Training Objective. Simplifications

$$\min_{\theta} \mathbb{E}_{x_0 \sim p(x_0)} \left[-\log p_\theta(x_0|x_1) + \sum_{t=2}^{T} KL(p(x_{t-1}|x_t,\ x_0)||p_\theta(x_{t-1}|x_t)) \right]$$

Tractable posterior:

$$p(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1};\ \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I\right)$$

$$\left.\begin{aligned} \tilde{\mu}(x_t, x_0) &= \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \hat{\beta}_t}} z_t \right), z_t \sim \mathcal{N}(0, \mathrm{I}) \\ \mu_\theta(x_t, x_0) &= \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \hat{\beta}_t}} z_\theta(x_t, t) \right) \end{aligned}\right\} \Rightarrow$$

$$\Rightarrow KL(p(x_{t-1}|x_t,\ x_0)\big|\big|p_\theta(x_{t-1}|x_t)\big) = \mathbb{E}_{z_t \sim \mathcal{N}(0,\mathrm{I})} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \hat{\beta}_t)} \|z_t - z_\theta(x_t, t)\|_2^2 \right]$$

Notations:

$$\hat{\beta}_t = \prod_{i=1}^{t} \alpha_i$$

$$\alpha_t = 1 - \beta_t$$

$$\tilde{\beta}_t = \frac{1 - \hat{\beta}_{t-1}}{1 - \hat{\beta}_t} \cdot \beta_t$$

DDPMs. Training Objective. Simplifications

$$\min_{\theta} \mathbb{E}_{x_0 \sim p(x_0)} \left[-\log p_\theta(x_0|x_1) + \sum_{t=2}^{T} KL(p(x_{t-1}|x_t,\ x_0)||p_\theta(x_{t-1}|x_t)) \right]$$

Tractable posterior:

$$p(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1};\ \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I)$$

$$\tilde{\mu}(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\hat{\beta}_t}} z_t \right), z_t \sim \mathcal{N}(0, \mathrm{I})$$

$$\mu_\theta(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\hat{\beta}_t}} z_\theta(x_t, t) \right)$$

$\Rightarrow$

Notations:

$$\hat{\beta}_t = \prod_{i=1}^{t} \alpha_i$$

$$\alpha_t = 1 - \beta_t$$

$$\tilde{\beta}_t = \frac{1-\hat{\beta}_{t-1}}{1-\hat{\beta}_t} \cdot \beta_t$$

Ignored

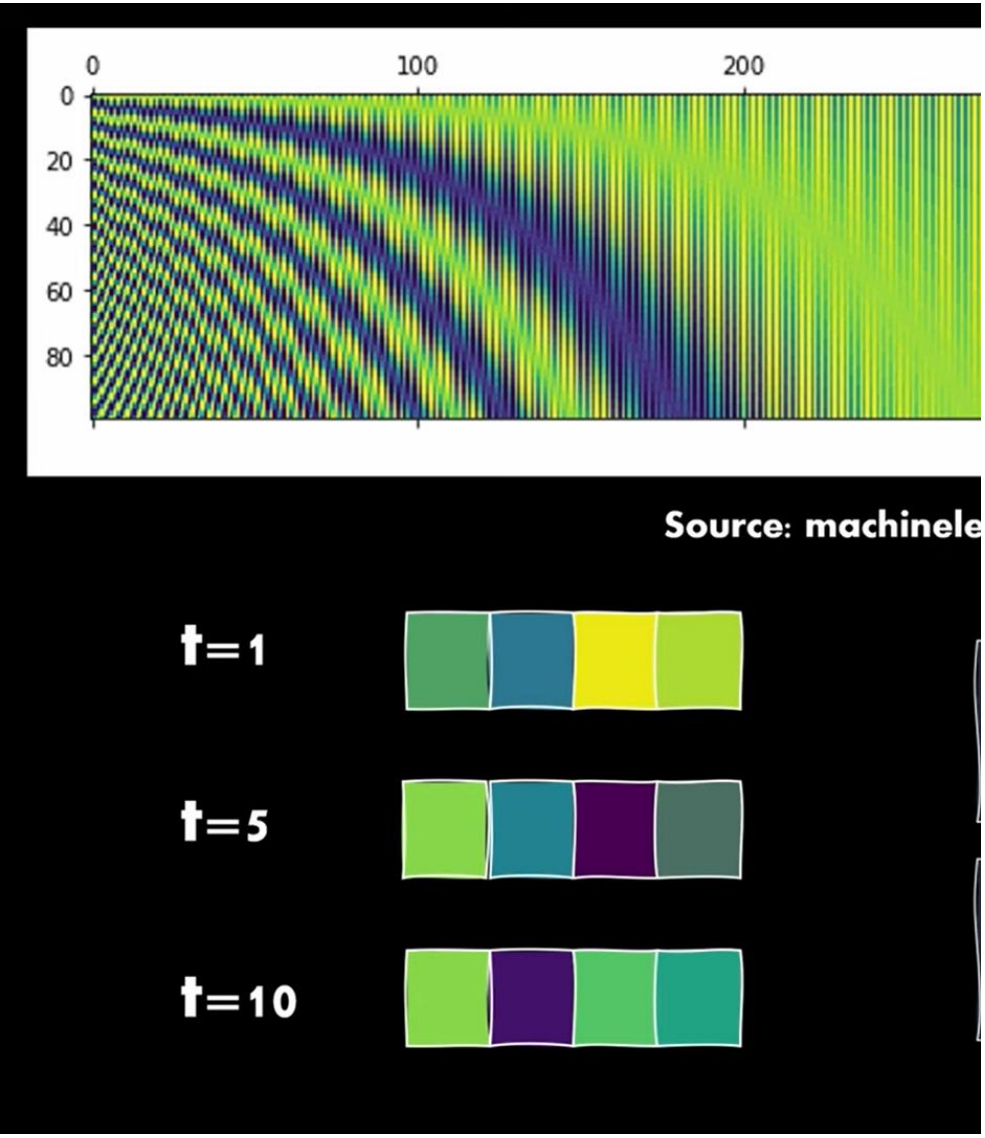
$$\Rightarrow KL(p(x_{t-1}|x_t,\ x_0)||p_\theta(x_{t-1}|x_t)) = \mathbb{E}_{z_t \sim \mathcal{N}(0,\mathrm{I})} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\hat{\beta}_t)} \|z_t - z_\theta(x_t, t)\|_2^2 \right]$$

DDPMs. Training Algorithm

$$\min_{\theta} \underbrace{\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{x_0 \sim p(x_0), z_t \sim \mathcal{N}(0,\mathrm{I})} \|z_t - z_\theta(x_t, t)\|_2^2}_{\mathcal{L}_{simple}}$$

Algorithm 1 Training

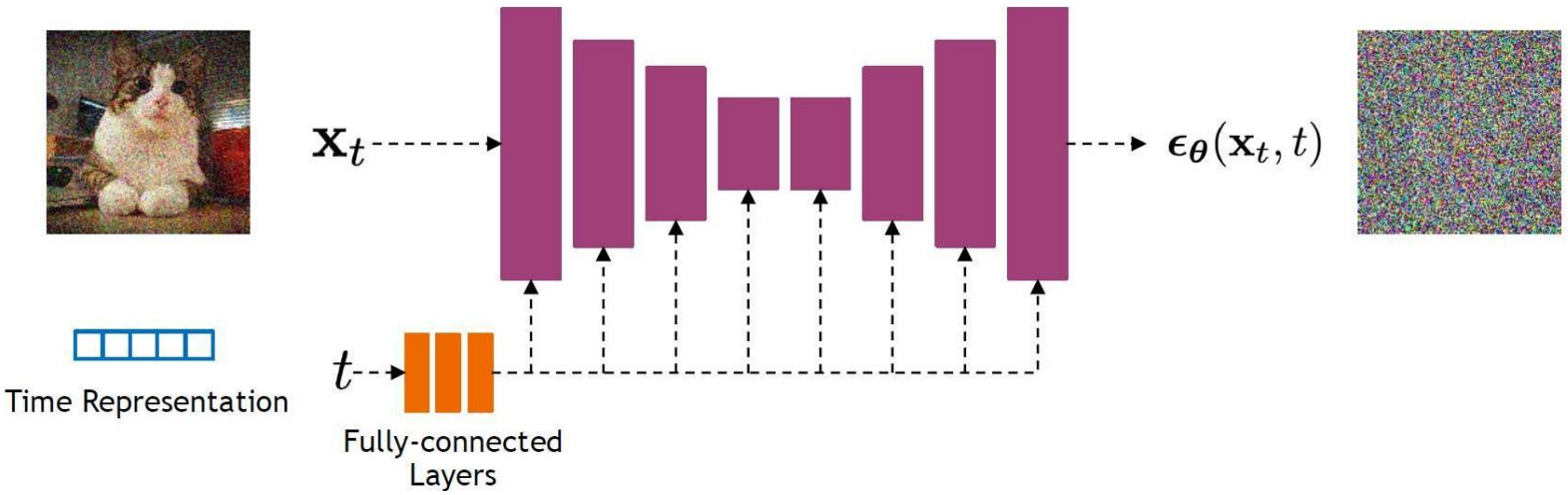
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0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


0
100
200
0
20
40
60
80
Source: machinele
t=1
t=5
t=10

Implementation Considerations

Network Architectures

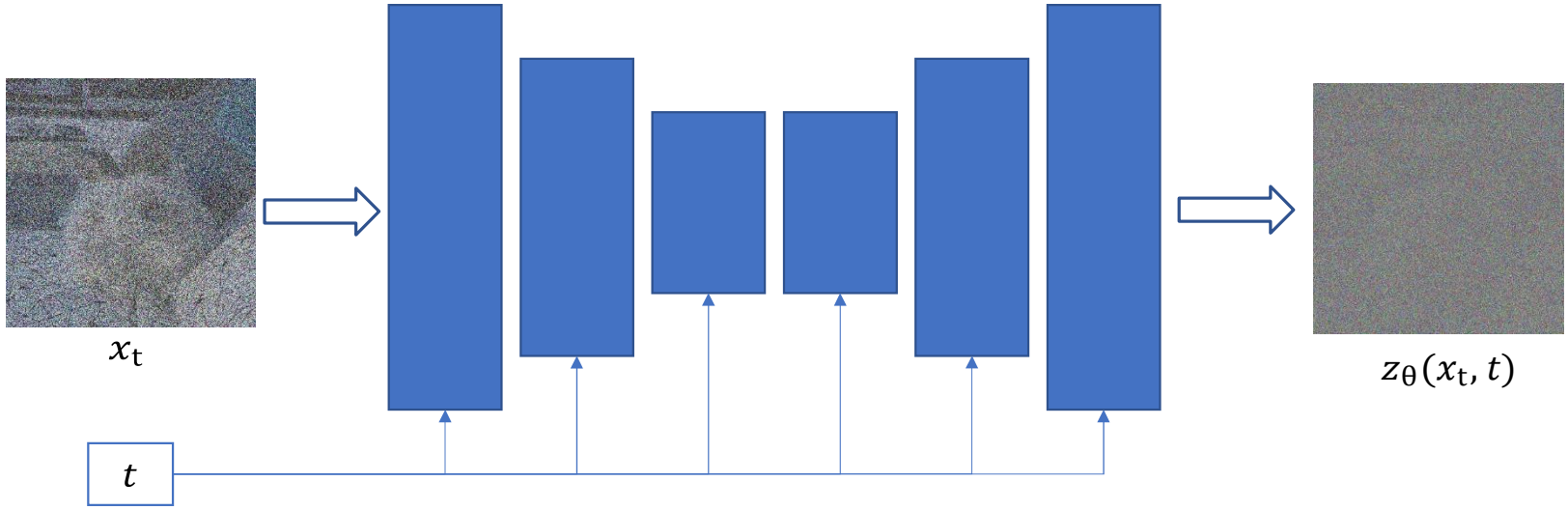
Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent $\epsilon_\theta(\mathbf{x}_t, t)$



Time representation: sinusoidal positional embeddings or random Fourier features.

Time features are fed to the residual blocks using either simple spatial addition or using adaptive group normalization layers. (see Dharivwal and Nichol NeurIPS 2021)

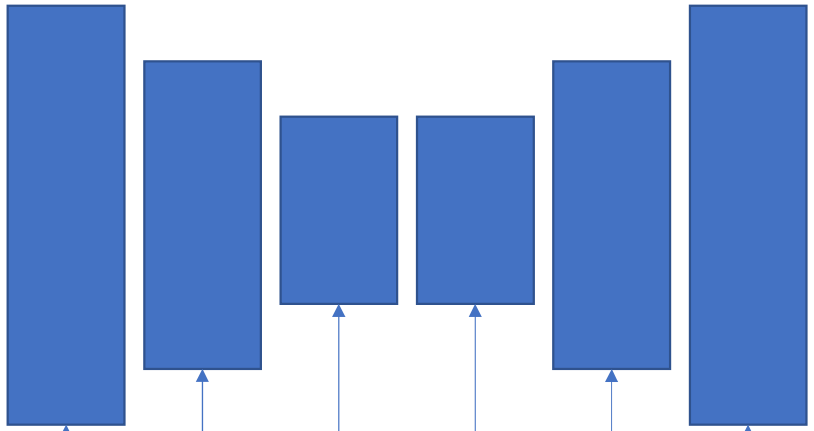
DDPMs. Sampling

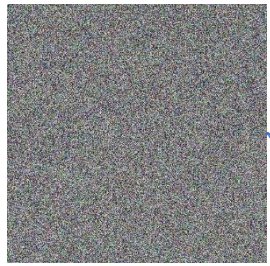


- Pass the current noisy image along with *t* to the neural network
- With the resultant z_θ compute the mean of the gaussian distribution

DDPMs. Sampling

x_t





$z_\theta(x_t, t)$

t

Sample the image x_{t-1} for the next iteration

x_{t-1}

$$\mu_\theta(x_t, t)$$

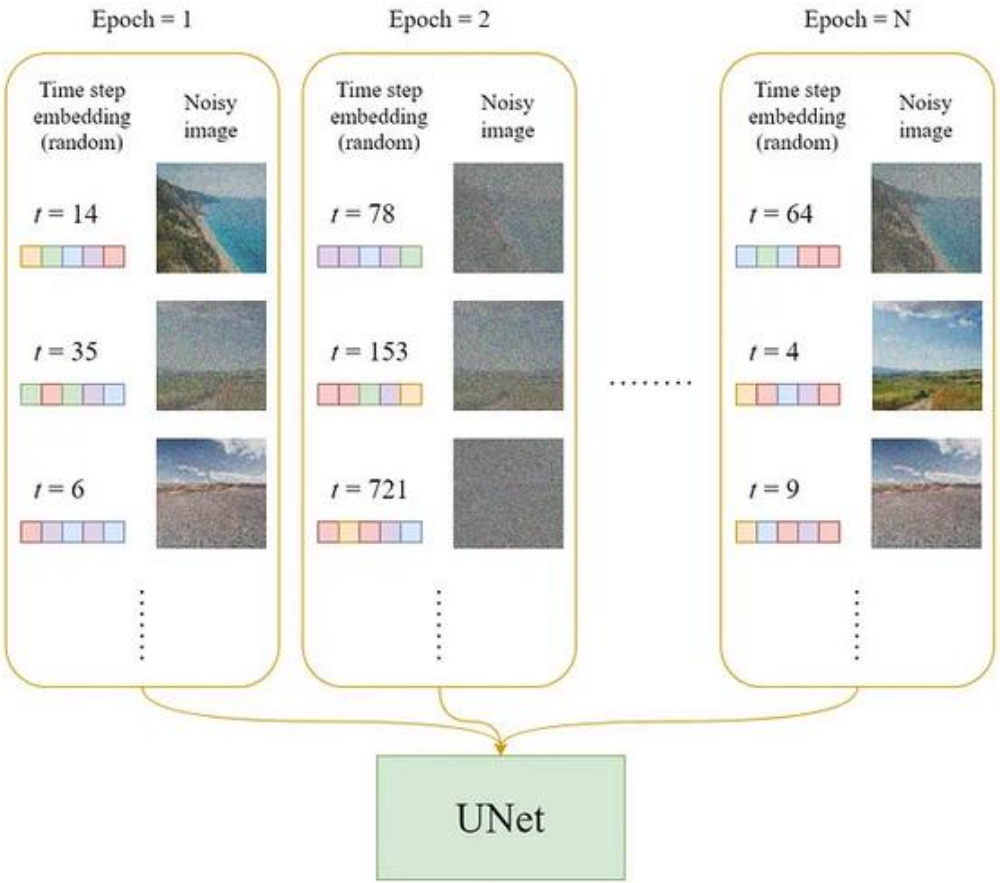
$$\sim \mathcal{N}\left(x_{t-1}, \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\hat{\beta}_t}} z_\theta(x_t, t)\right), \sigma_t^2 \mathrm{I}\right)$$

The U-Net Model

Dataset

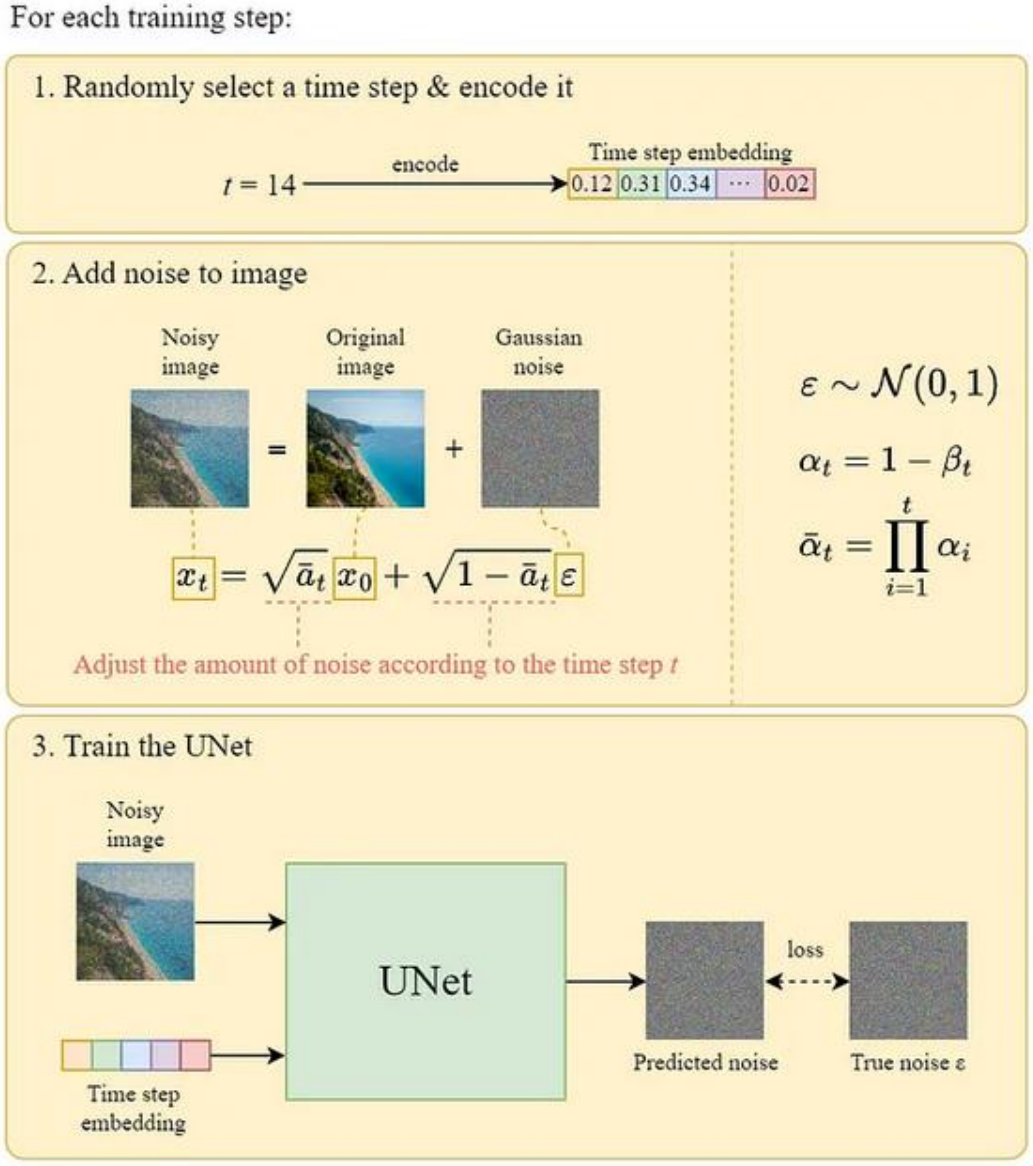
In each epoch:

1. A random time step t will be selected for each training sample (image).
2. Apply the Gaussian noise (corresponding to t) to each image.
3. Convert the time steps to embeddings (vectors).



Dataset for training

The official training algorithm is as above, and the following diagram is an illustration of how a training step works:

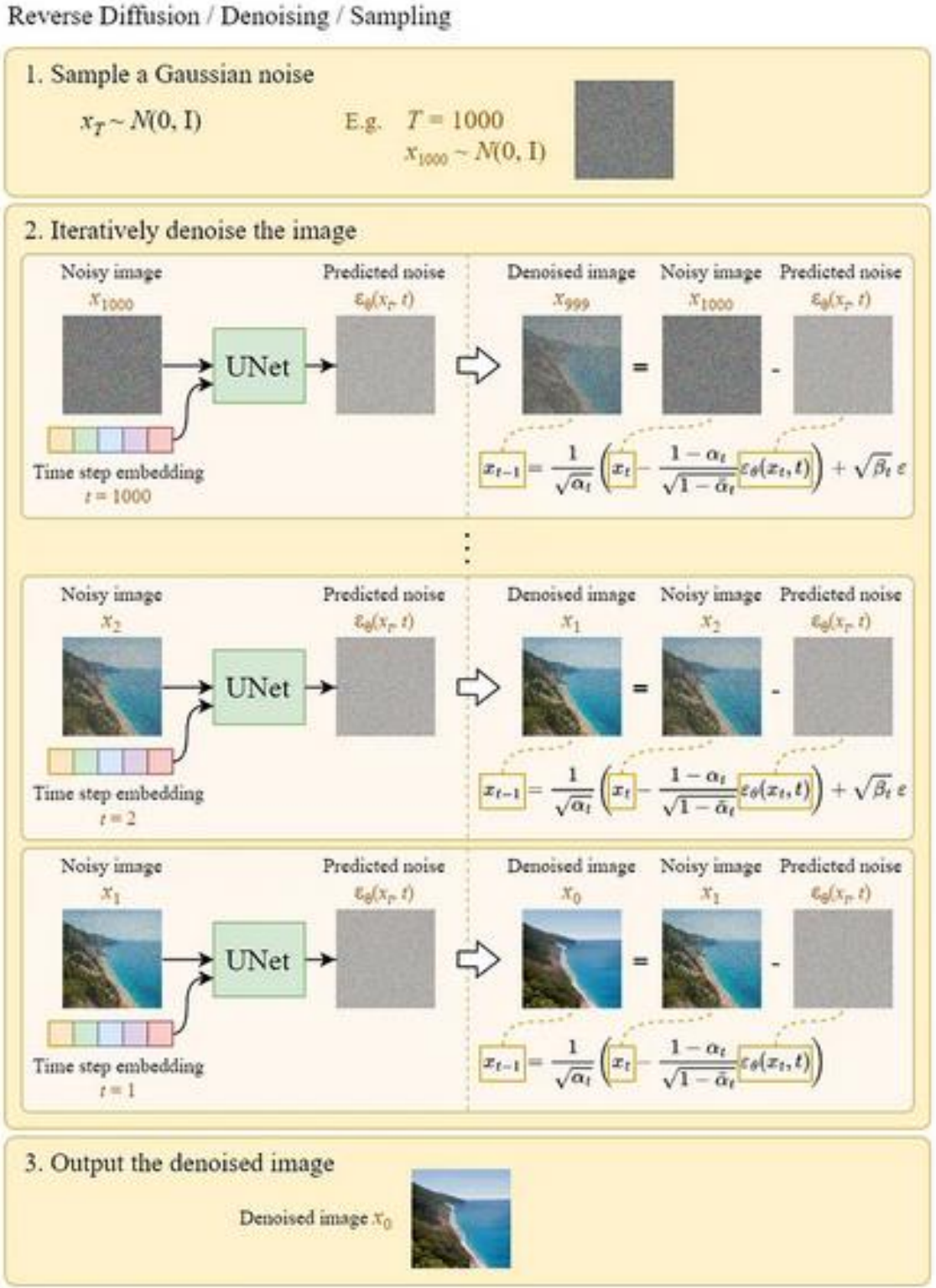


Training step illustration

Algorithm 1 Training

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

Training algorithm [2]



Sampling illustration

Reverse Diffusion

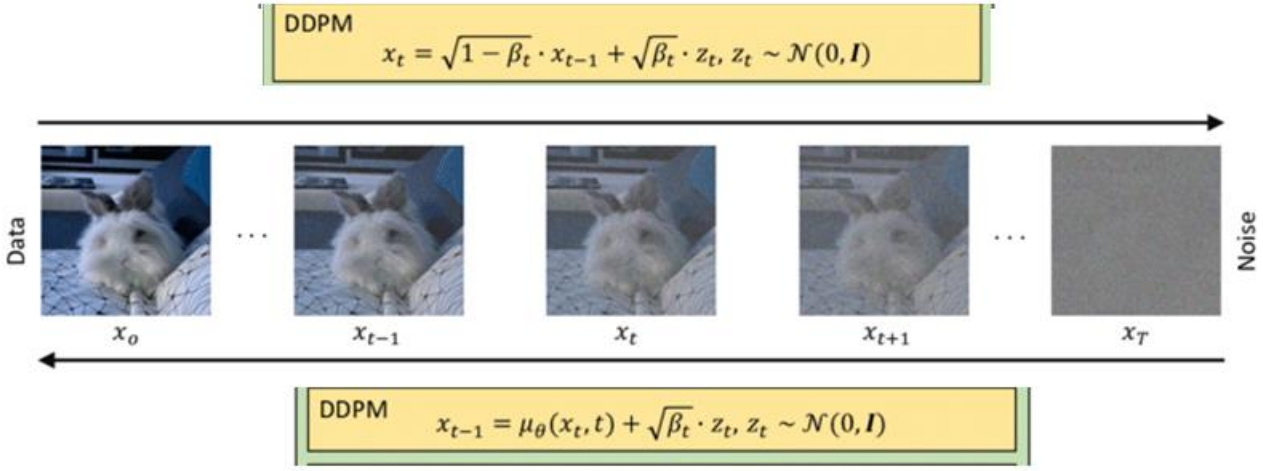
Algorithm 2 Sampling

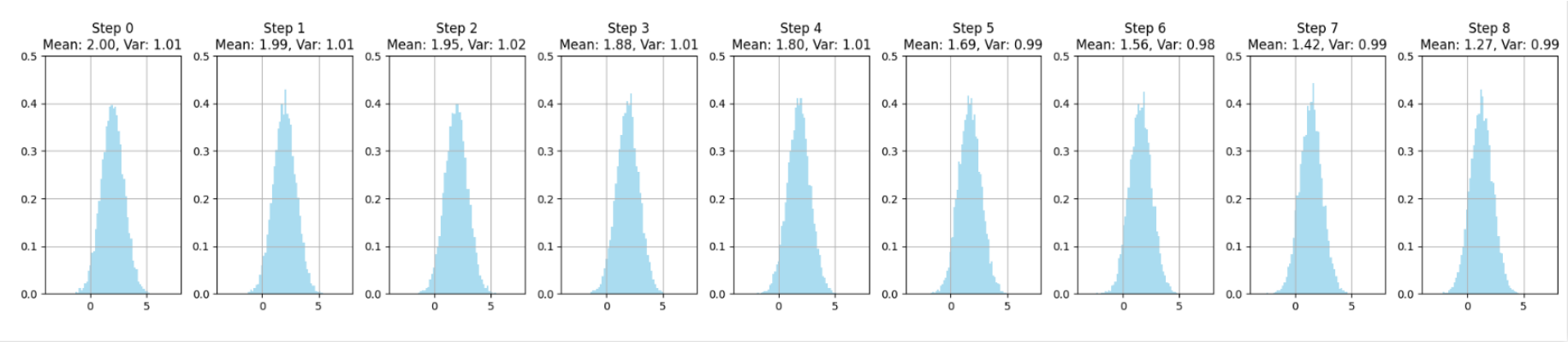
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,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

Sampling algorithm [2]

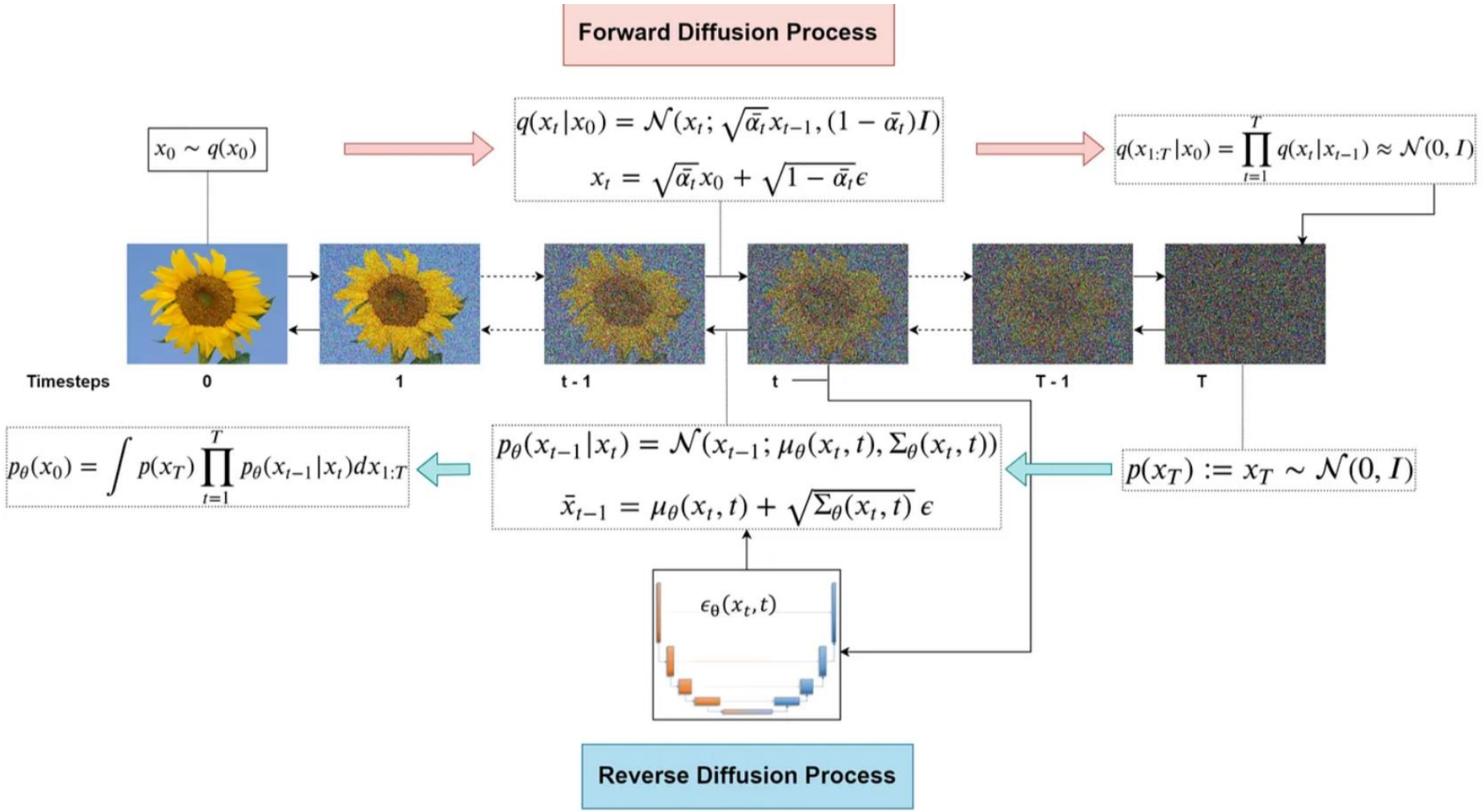
We can generate images from noises using the above algorithm. The following diagram is an illustration of it:

8. https://medium.com/@steinsfu/diffusion-model-clearly-explained-cd331bd41166
9. https://theaisummer.com/diffusion-models/
10. https://stats.stackexchange.com/questions/600127/purpose-of-scaling-mean-by-sqrt1-beta-t-in-forward-diffusion-process
11. https://www.youtube.com/watch?v=zF5DU5bHTP0&list=PLd3hlSJsX_In7qup928HaHmilugBGctuF&index=4
12. https://www.youtube.com/watch?v=a4Yfz2FxXiY&t=40s
13. Code: https://github.com/dtransposed/code_videos/blob/main/Diffusion%20Model.ipynb
14. https://medium.com/@kemalpiro/step-by-step-visual-introduction-to-diffusion-models-235942d2f15c





This highlights how the forward process maintains a fixed variance while progressively reducing the contribution of the original data's structure by modifying the mean. Let me know if you'd like further clarifications or examples!


Forward Diffusion Process
$x_0 \sim q(x_0)$
$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_{t-1}, (1-\bar{\alpha}_t)I)$
$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$
$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \approx \mathcal{N}(0, I)$
Timesteps
0
1
t - 1
t
T - 1
T
$p_\theta(x_0) = \int p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) dx_{1:T}$
$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$
$\bar{x}_{t-1} = \mu_\theta(x_t, t) + \sqrt{\Sigma_\theta(x_t, t)}\,\epsilon$
$p(x_T) := x_T \sim \mathcal{N}(0, I)$
$\epsilon_\theta(x_t, t)$
Reverse Diffusion Process