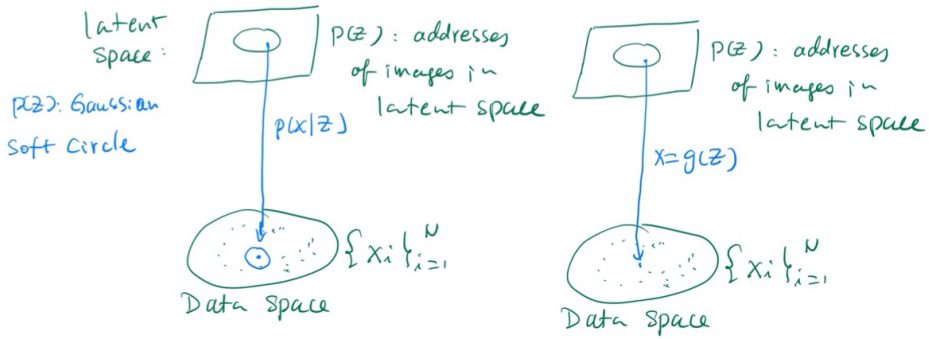
2024-11-0

Recap Basic Generative AI Methods for Images

- *Varitional autoencoder* for data generation and representation learning
  $\{\mathbf{x}_i\}_{i=1}^N, p(\mathbf{z}) \rightarrow p(\mathbf{x}|\mathbf{z})$ $q(\mathbf{z}|\mathbf{x})$ used in inference
- *Generative adversarial networks* for data generation
  $\{\mathbf{x}_i\}_{i=1}^N, p(\mathbf{z}) \rightarrow \mathbf{x} = g(\mathbf{z})$

* Diffusion model



VAE GAN

Model used to generate new data:

VAE: $z \sim p(z)$ $x \sim p(x|z)$

GAN: $z \sim p(z)$ $x = g(z)$

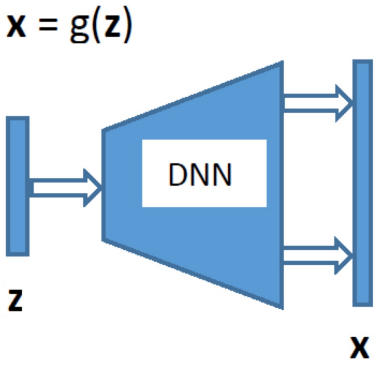
New image

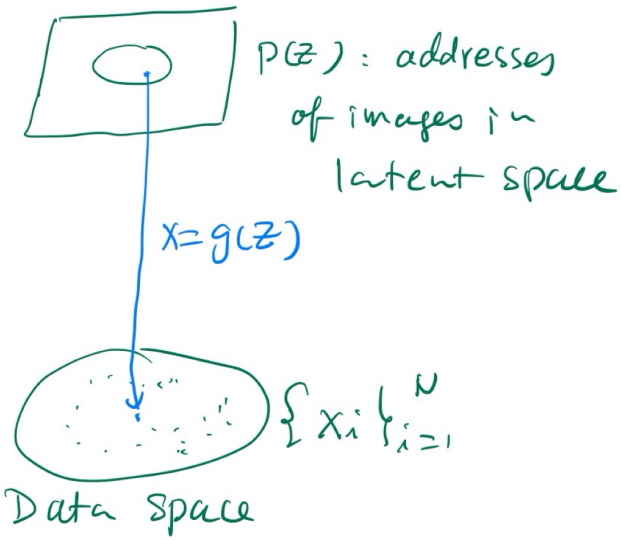
Today: GAN & Diffusion models

Task :

* Given : $\{x_i\}_{i=1}^{N}$, $p(z)$

* Learn : $g : z \longrightarrow x$





* Generates fake images

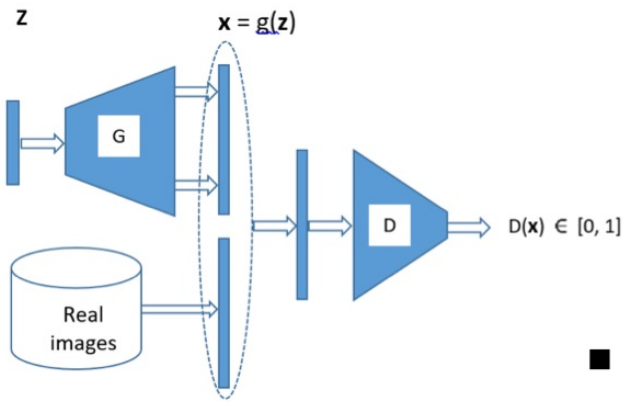
* How to gain the ability to generate realistically looking images ?

What drives the improvement of fake money production techniques?

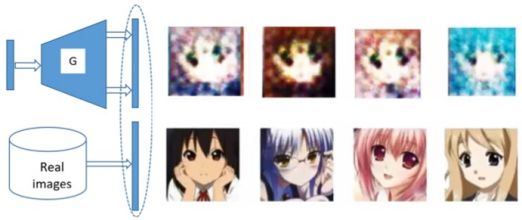
* Technology
* police :
    * Try to detect counterfeit money
    * Force counterfeiters to improve their techniques

GAN borrows this idea!

Generative Adversarial Networks (GAN)



- The generator G and the discriminator D are trained alternately.
- When training D, the objective is to tell real and fake examples apart, i.e., to determine θ_d such that
  - $D(\mathbf{x})$ is 1 or close to 1 if $\mathbf{x}$ is a real example.
  - $D(\mathbf{x})$ is 0 or close to 0 if $\mathbf{x}$ is a fake example.
- When training G, the objective is to fool D, i.e, to generate fakes examples that D cannot tell from real examples.
- Notation: θ_g: parameters for the generator. θ_d: parameters for the discriminator.



- At the beginning, the parameters of G are random. It generates poor images.

- GAN learns a discriminator D tell to real and fake images apart.

- Then GAN improves G.
- The improved G can generate images that fool the initial weak D.

- Then, D is told that the images on the first row are actually fake.
- It is therefore improved using this knowledge.

- Next, G will learn to improve further to fool this smarter D.

Training D

Traing G

The GAN training algorithm (Goodfellow et al 2014)

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

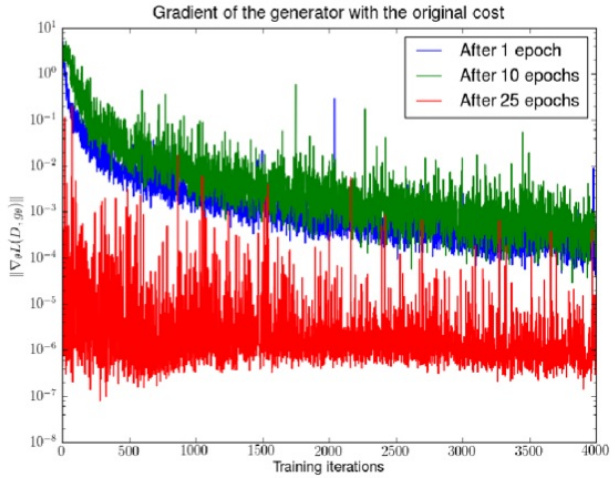
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Problem with loss for G

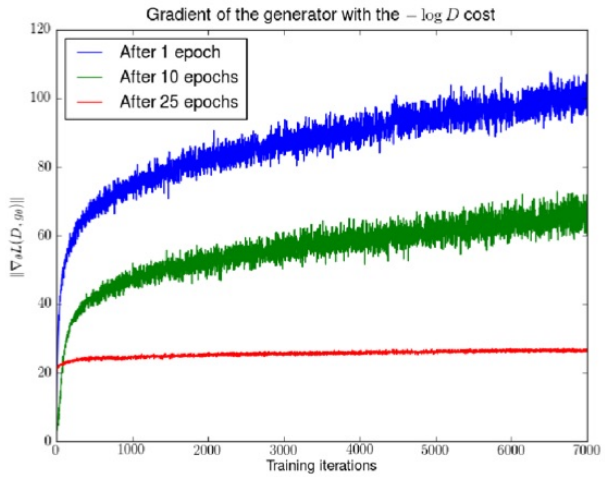
Gradient of Original Generator Cost Function



- How the gradient of the original cost function change as we train the discriminator to optimum.
- Arjovsky et al. ICLR 2017: " First, we trained a DCGAN for 1, 10 and 25 epochs. Then, with the generator fixed we train a discriminator from scratch and measure the gradients with the original cost function. We see the gradient norms decay quickly, in the best case 5 orders of magnitude after 4000 discriminator iterations. Note the logarithmic scale "

Alternative loss for G

Gradient of the Alternative Generator Cost Function



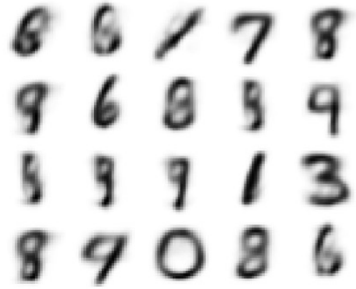
- How the gradient of the alternative cost function change as we train the discriminator to optimum.
- With a poor generator (the one after only one epoch of training), the gradient of the alternative cost function goes to infinite.
- The general trend is the same with a better generator (the ones after 10, or 25 epochs of training), except now the gradient increase at a slower pace.

Empirical Results

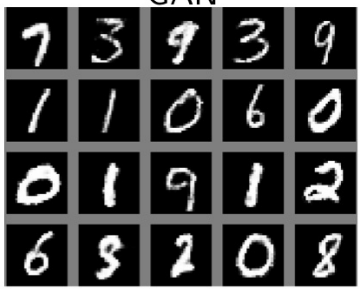
- GAN generates sharper images than VAE.

Reasons to be discussed later

VAE

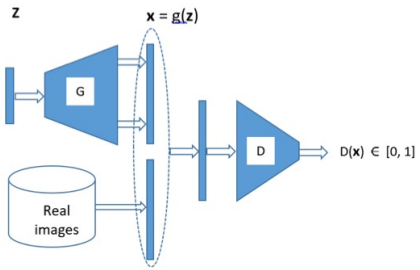


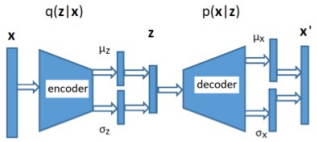
GAN



Tutorial 7, HA 8

2 Theoretical Analysis of GAN





The above analysis leads to the following view on GAN:

- The objective of GAN is to learn the parameters of the generator so as to minimize the JS divergence $JS(p_r||p_g)$ between the real data distribution p_r and the generator distribution p_g.
- The introduction of a discriminator is to approximate the JS divergence.

Recall that the objective of VAE is to learn parameters of the decoder by minimizing the KL divergence $KL(p_r||p_g)$ between p_r and p_g (called the decoder distribution in VAE context). The introduction of an encoder $q(\mathbf{z}|\mathbf{x})$ is to provide an approximation (an upper bound) for the KL divergence

From Lol-2

General ML Setup

P —assume / sampling→ data —learning→ Q (model)

P: unknown

Want : Q to be close to P

Need "distance" measure

* Kullback-Leibler (KL) divergence
* Jensen-shanon (JS) divergence
* Earth-mover / Wasserstein distance (later)
* Maximum mean descrepancy (MMD)
* Total Variation

Why GAN generates more realistic images than VAE?

- VAE minimize the KL divergence

$$KL(p_r||p_g) = \int p_r(\mathbf{x}) \log \frac{p_r(\mathbf{x})}{p_g(\mathbf{x})} d\mathbf{x}$$

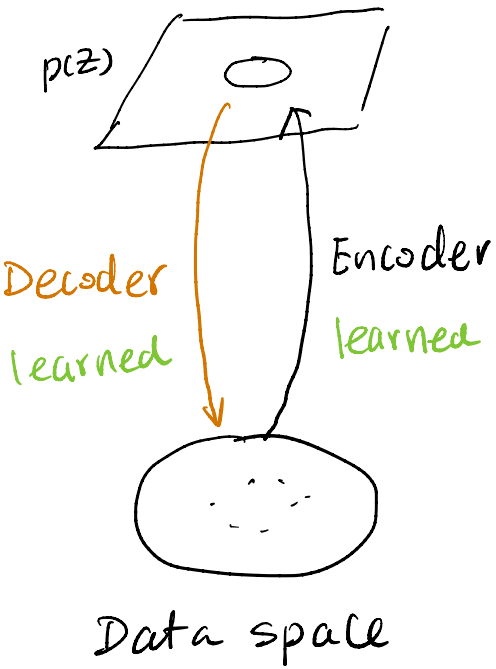
- GAN minimize the JS divergence

$$JS(p_r||p_g) = \frac{1}{2}KL(p_r||p_a) + \frac{1}{2}KL(p_g||p_a)$$

where $p_a = (p_r + p_g)/2$.

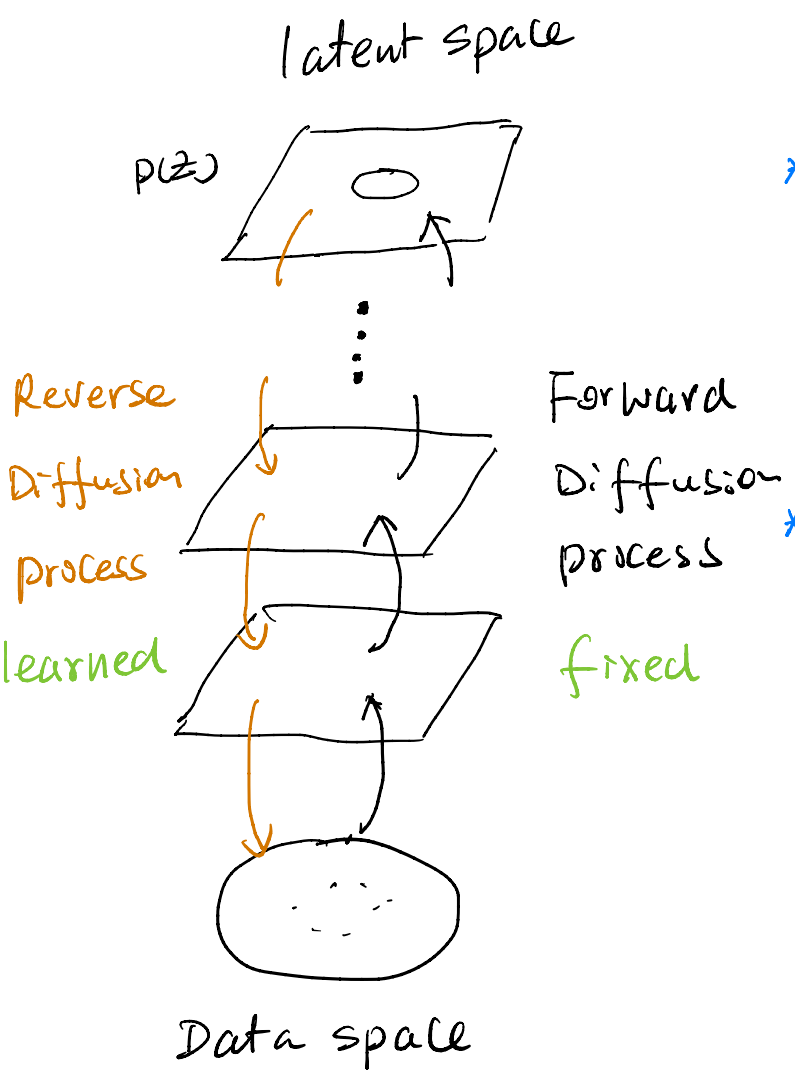
$\{x^i\}$, $p(z)$

VAE



* one step encoding
* Reversed in one step.

HARDER

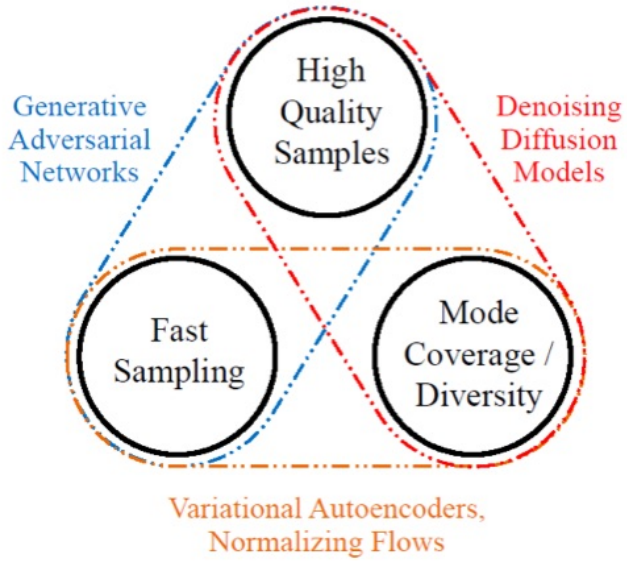


Diffusion Models

* Map data to latent space in multiple steps
* Reversing the process step by step

EASIER & leads to better images

Introduction



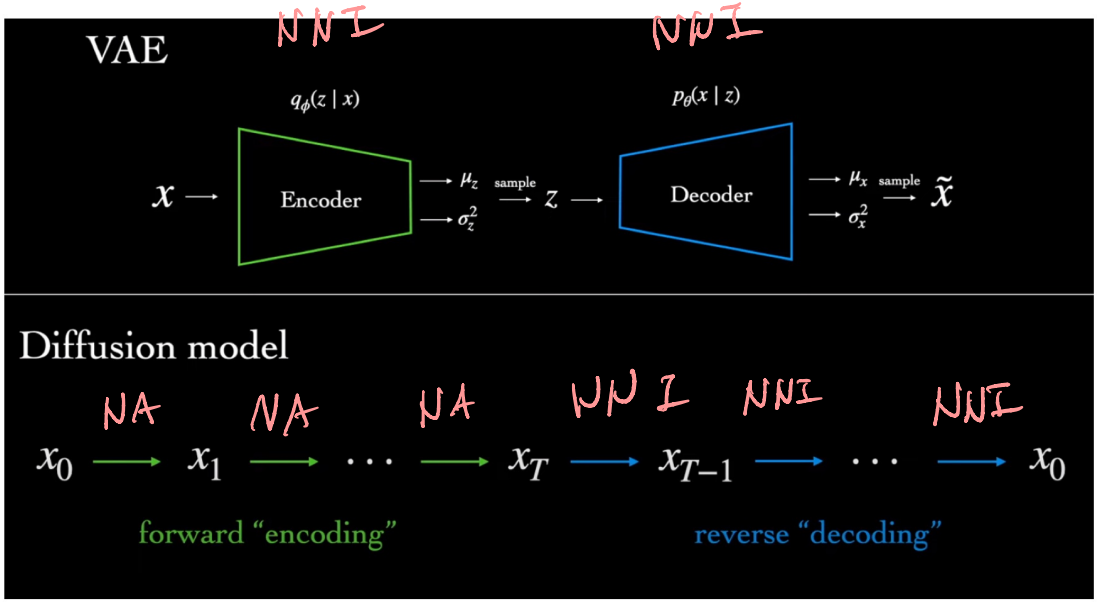
Xiao et al, Tackling the Generative Learning Trilemma with Denoising Diffusion GANs, ICLR 2022

- GANs generate high-quality samples rapidly, but have poor mode coverage.
- VAEs and normalizing flows cover data modes faithfully, but they often suffer from low sample quality
- Diffusion models generate high-equality images (beats GAN) and have good mode coverage, but are slow in generating images (a weakness being addressed).

Outline

Denoising Diffusion Probabilistic Models (DDPM)

what happens to one training example

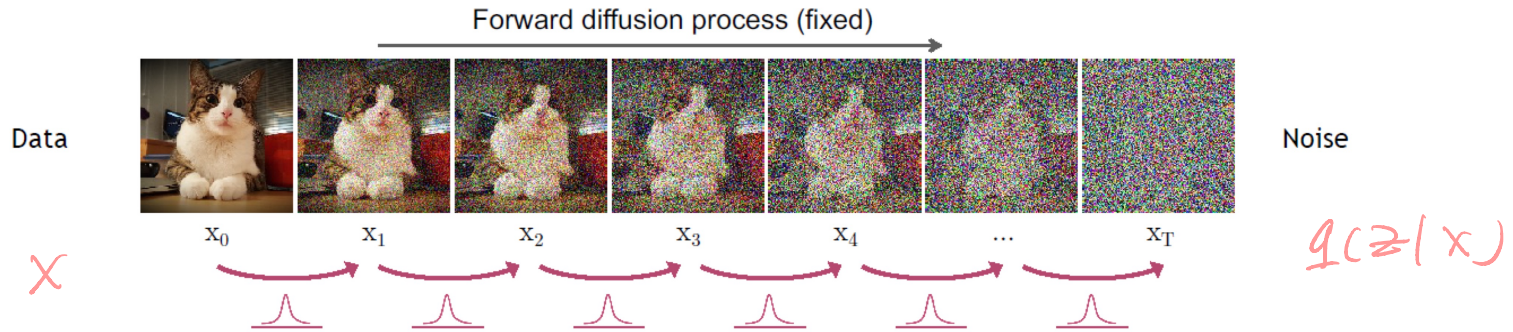


NNI: neural net inference

NA : noise addition, fixed

Denoising Diffusion Probabilistic Models (DDPM)

what happens to one training example



x

$q(z|x)$

$p(x|z)$

z

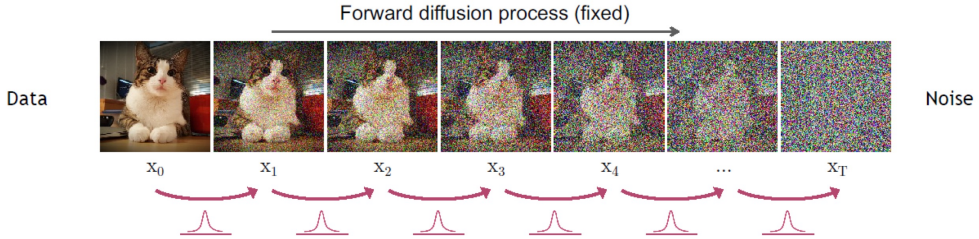
preparation

$$X \sim N(\mu_X, \sigma_X^2)$$

$$Y \sim N(\mu_Y, \sigma_Y^2)$$

$$X+Y \sim$$

DDPM: The Forward/Diffusion Process [1]



Noise Schedule

Cosine schedule

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos^2\left(\frac{\frac{t}{T}+s}{1+s}\cdot\frac{\pi}{2}\right), \quad \beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$$

Linear schedule

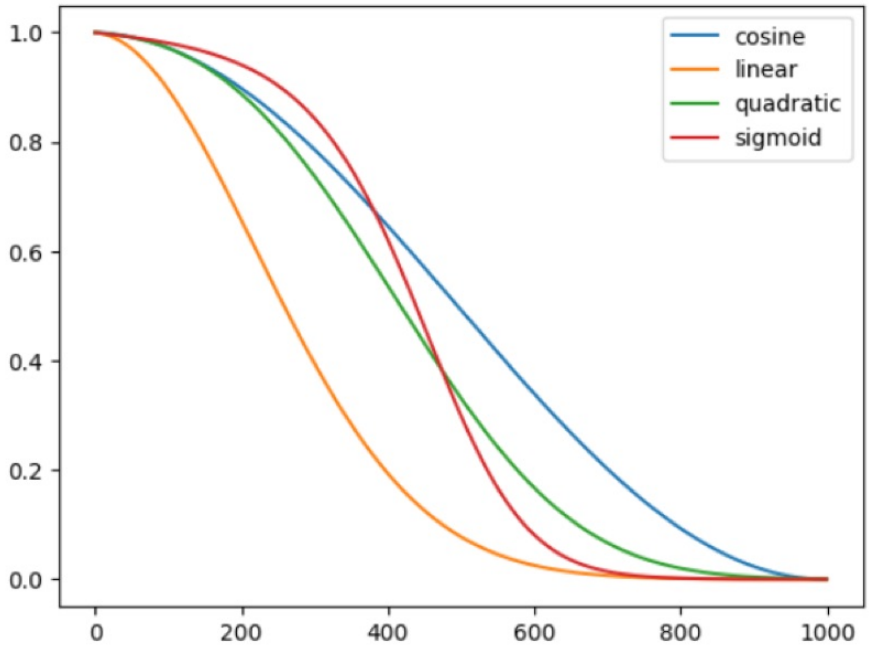
$$\beta_t = \beta_1 + \frac{t-1}{T-1}(\beta_T - \beta_1)$$

Quadratic schedule

$$\beta_t = \left[\sqrt{\beta_1} + \frac{t-1}{T-1}\left(\sqrt{\beta_T} - \sqrt{\beta_1}\right)\right]^2$$

Sigmoid schedule

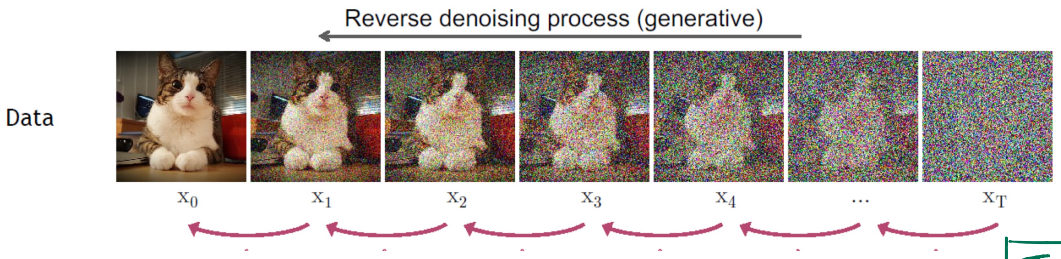
$$\beta_t = \beta_1 + \frac{\beta_T - \beta_1}{1 + e^{\tau\left(1-2\cdot\frac{t-1}{T-1}\right)}}$$



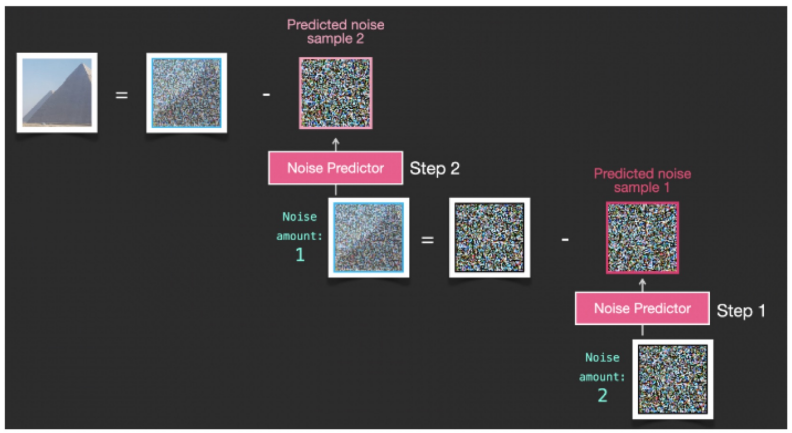
How to Sample x_t ?

DDPM: The Reverse Process

To be learned



Intuition



Forward

noise added at step t

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \varepsilon_t \qquad \varepsilon_t \sim N(0, I)$$

$$= \underbrace{\sqrt{\alpha_t}\, x_{t-1}}_{\text{suppress signal}} + \underbrace{\sqrt{\beta_t}\, \varepsilon_t}_{\text{Add noise}}$$

Reverse

$$x_{t-1} = \underbrace{\frac{1}{\sqrt{\alpha_t}}}_{\text{Enhance Signal}} \left(x_t - \underbrace{\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}}_{\text{Denoise}} \varepsilon_\theta(x_t, t) \right) + \underbrace{\sigma_t z}_{\text{technical reason}}$$

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \bar{\varepsilon}_t \qquad \bar{\varepsilon} \sim N(0, I)$$

ε_θ is an estimation of $\bar{\varepsilon}_t$, compound noise (To be learned)

$\sigma_t = \sqrt{\beta_t}$,

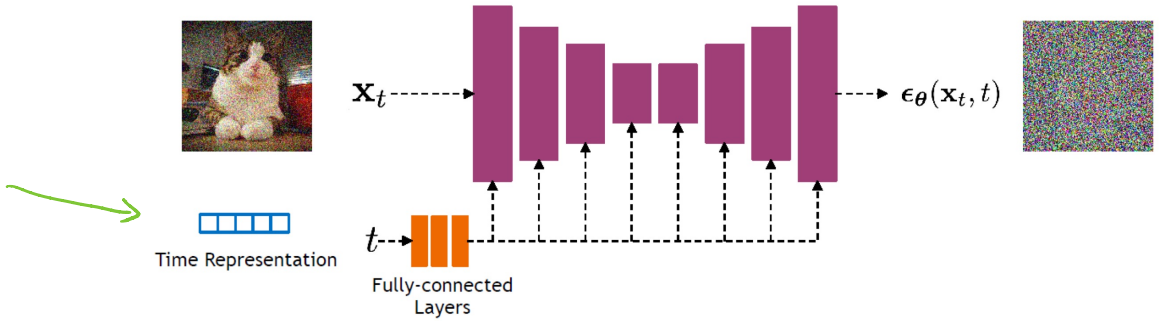
$z \sim N(0, I)$

DDPM: Noise Predictor

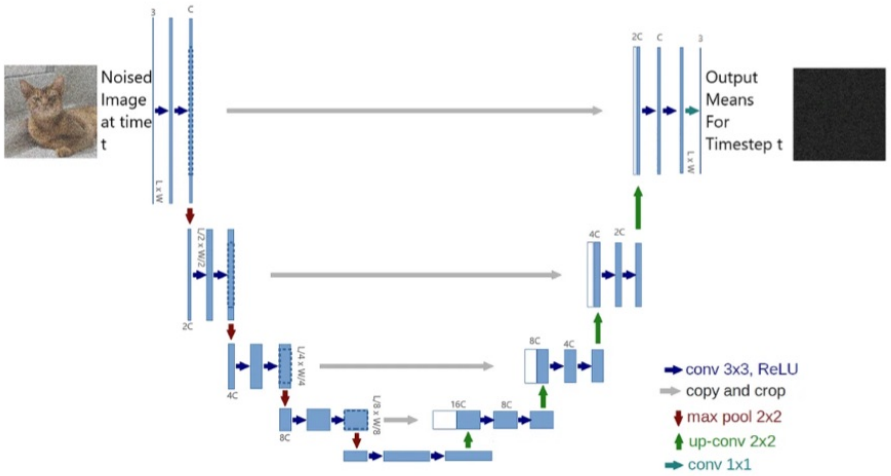
for predicting compound error $\tilde{\epsilon}_t$

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

Sinusoidal position embedding



DDPM: U-Net



https://betterprogramming.pub/diffusion-models-ddpms-ddims-and-classifier-free-guidance-e07b297b2869

- Input and output have the same size.
- Deeper layers extract more general features the deeper it goes.
- Skip connections reintroduce detailed features into the decoder.

DDPM: Training the Noise Predictor

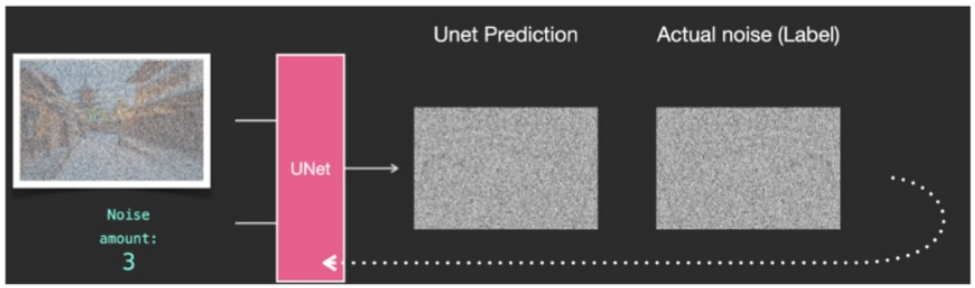
$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t \qquad \bar{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

- Input: A noisy image $\mathbf{x}_t$, and embedding of time step t
- Output: An approximation $\epsilon_\theta(\mathbf{x}_t, t)$ of the compound noise $\bar{\epsilon}_t$ that was added to $\mathbf{x}_0$ to create $\mathbf{x}_t$.

Training Objective: $\min ||\bar{\epsilon}_t - \epsilon_\theta(\mathbf{x}_t, t)||^2$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \ldots, T\})$
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



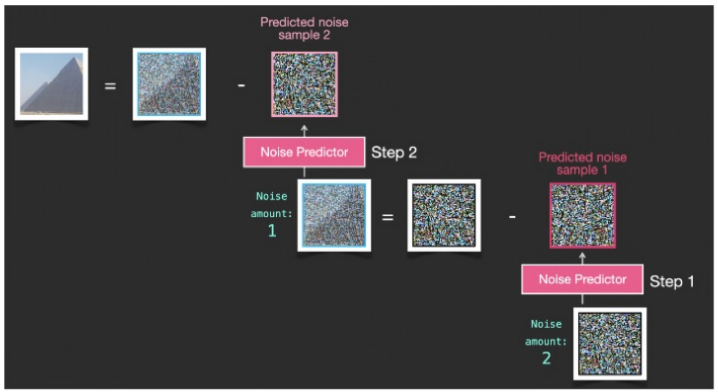
http://jalammar.github.io/illustrated-stable-diffusion/

DDPM: Sampling/Image Generation

- $\mathbf{x}_t$ is obtained from x_0 by adding compound noise $\bar{\epsilon}_t$.
- Now, we have an approximation of $\bar{\epsilon}_t$, i.e., $\epsilon_\theta(\mathbf{x}_t, t)$, we can do denoising to generate images

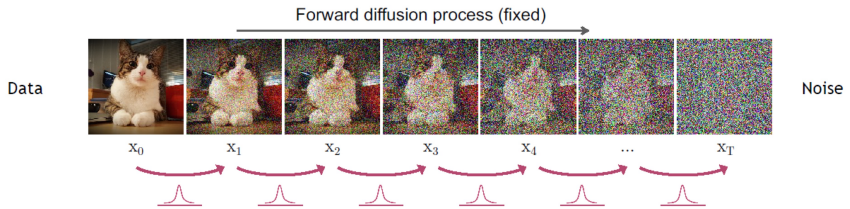
Algorithm 2 Sampling

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



http://jalammar.github.io/illustrated-stable-diffusion/

Theory



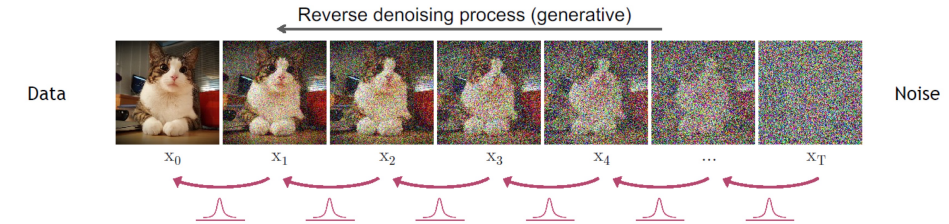
$$\begin{aligned} \mathbf{x}_0 &\sim q(\mathbf{x}) && \text{(data distribution)} \\ \mathbf{x}_t &= \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon_t && \epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \end{aligned}$$

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

$(x_1, x_2, \cdots, x_T)$

$$q(\mathbf{x}_{0:T}) = q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)$$



$$\begin{aligned} \mathbf{x}_T &\sim p(\mathbf{x}_T) && \text{(Gaussian)} \\ \mathbf{x}_{t-1} &= \mu_\theta(\mathbf{x}_t, t) + \sigma_t\mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) && \text{(often } \sigma_t^2 = \beta_t\text{)} \end{aligned}$$

To be determined

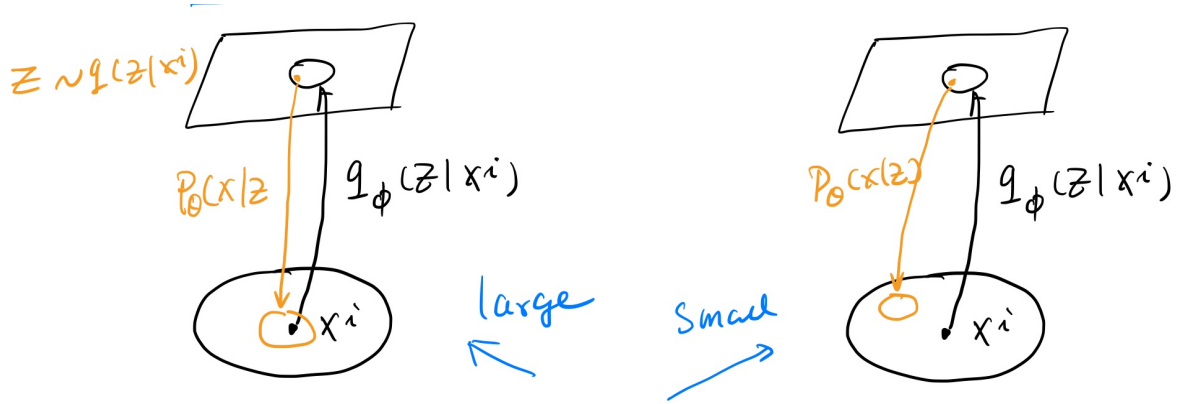
$$\begin{aligned} p(\mathbf{x}_T) &= \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}) \\ p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) &= \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I}) \end{aligned}$$

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

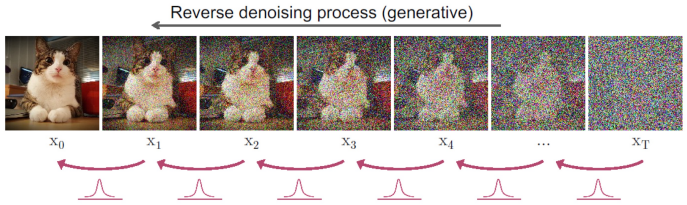
minimize NLL

$$E_{x_0 \sim q(x_0)}[-\log p_\theta(x_0)]$$

Recall: Reconstruction Loss of VAE



$$\max L(x^i, \theta, \phi) = E_{z \sim q(z|x^i)}[\log p(x^i|z)]$$



$$\begin{aligned} \mathbf{x}_T &\sim p(\mathbf{x}_T) && \text{(Gaussian)} \\ \mathbf{x}_{t-1} &= \mu_\theta(\mathbf{x}_t, t) + \sigma_t \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) && \text{(often } \sigma_t^2 = \beta_t\text{)} \end{aligned}$$

↓ To be determined

$$\begin{aligned} p(\mathbf{x}_T) &= \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}) \\ p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) &= \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I}) \end{aligned}$$

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

minimize NLL

$$E_{x_0 \sim q(x_0)}[-\log p_\theta(x_0)]$$

It is found that, to minimize the NLL, $\mu_\theta(x_t, t)$ should be parameterized as follows:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\theta(x_t, t)\right)$$

where ε_θ is an estimation of the compound error $\bar{\varepsilon}_t$:

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

Summary

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

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
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** $\mathbf{x}_0$

* Training: Learn $\epsilon_\theta(x_t, t)$ to predict the compound noise added to x_0 to obtain x_t

* Sampling: $x_T \sim N(0, I)$

$x_{t-1} \sim p(x_{t-1} \mid x_t)$ $\quad t = T, T-1, \dots, T_1$

$T = 1{,}000$ (e.g.)

Tutorial & HA

2 Making Diffusion Models More Efficient : DDIM Sampling

DDPM Sampling

t-1 t

... 998 999 1000

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
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** $\mathbf{x}_0$

DDIM sampling can be accelerated by carrying it out at a subset of time points:

$$\tau_1 = 1 < \tau_2 < \ldots < \tau_t = T$$

$$\mathbf{x}_{\tau_{i-1}} = \sqrt{\bar{\alpha}_{\tau_{i-1}}}\,\hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_{\tau_{i-1}} - \sigma_{\tau_i}^2}\,\hat{\epsilon}_{\tau_i} + \sigma_{\tau_i}\mathbf{z} \qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon(x_t, t)}{\sqrt{\bar{\alpha}_t}}$$

$$\hat{\epsilon}_t = \frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\,\hat{\mathbf{x}}_0}{\sqrt{1-\bar{\alpha}_t}}$$

τ_{i-1} τ_i

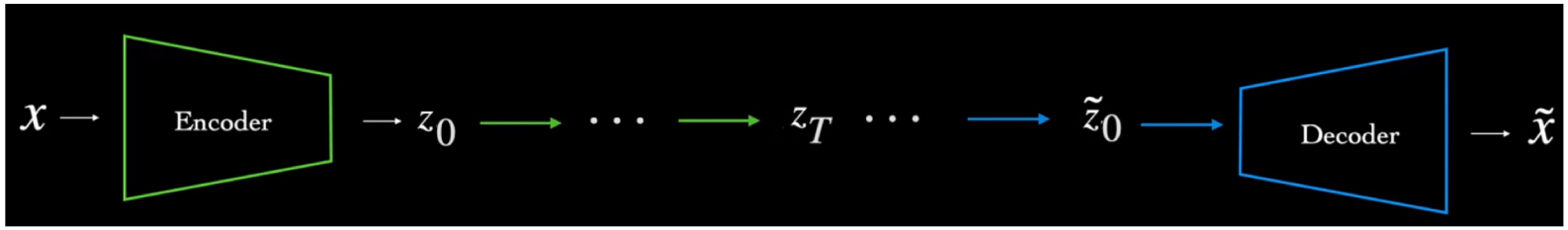
... 960 980 1000

$x_{\tau_i} \Rightarrow \hat{x}_0, \hat{\epsilon}_{\tau_i}$

$\Rightarrow x_{\tau_{i-1}}$

2 Making Diffusion Models More Efficient : Latent Diffusion Model (LDM)

LDM

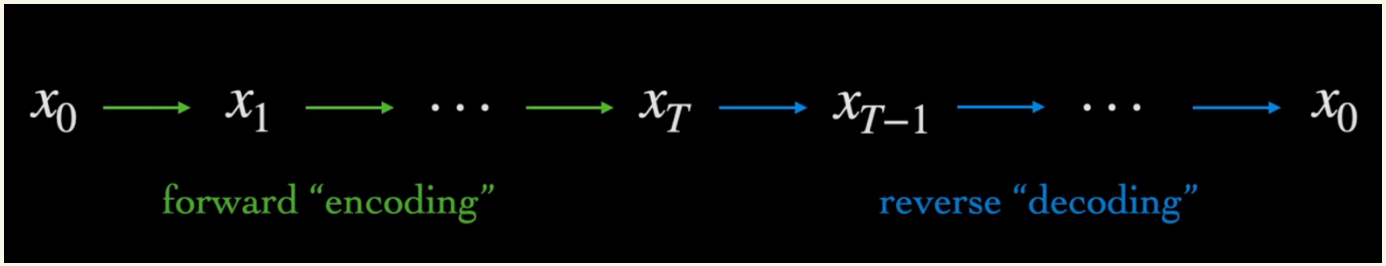


Phase 1: Pre-train a VAE to map data to a latent space, e.g., from $3 \times 512 \times 512$ to $6 \times 64 \times 64$.

Phase 2: Learn a diffusion model for data in the latent space.

$$L_{LDM} = \mathbb{E}_{\mathcal{E}(\mathbf{x}),t,\epsilon}||\epsilon - \epsilon_\theta(\mathbf{z}_t, t)||^2$$

DDPM



3 Stable Diffusion

Demo: * cat in park

* Cat in water

* cat professor giving lecture

Ways to Use Stable Diffusion

- Generate images from texts:

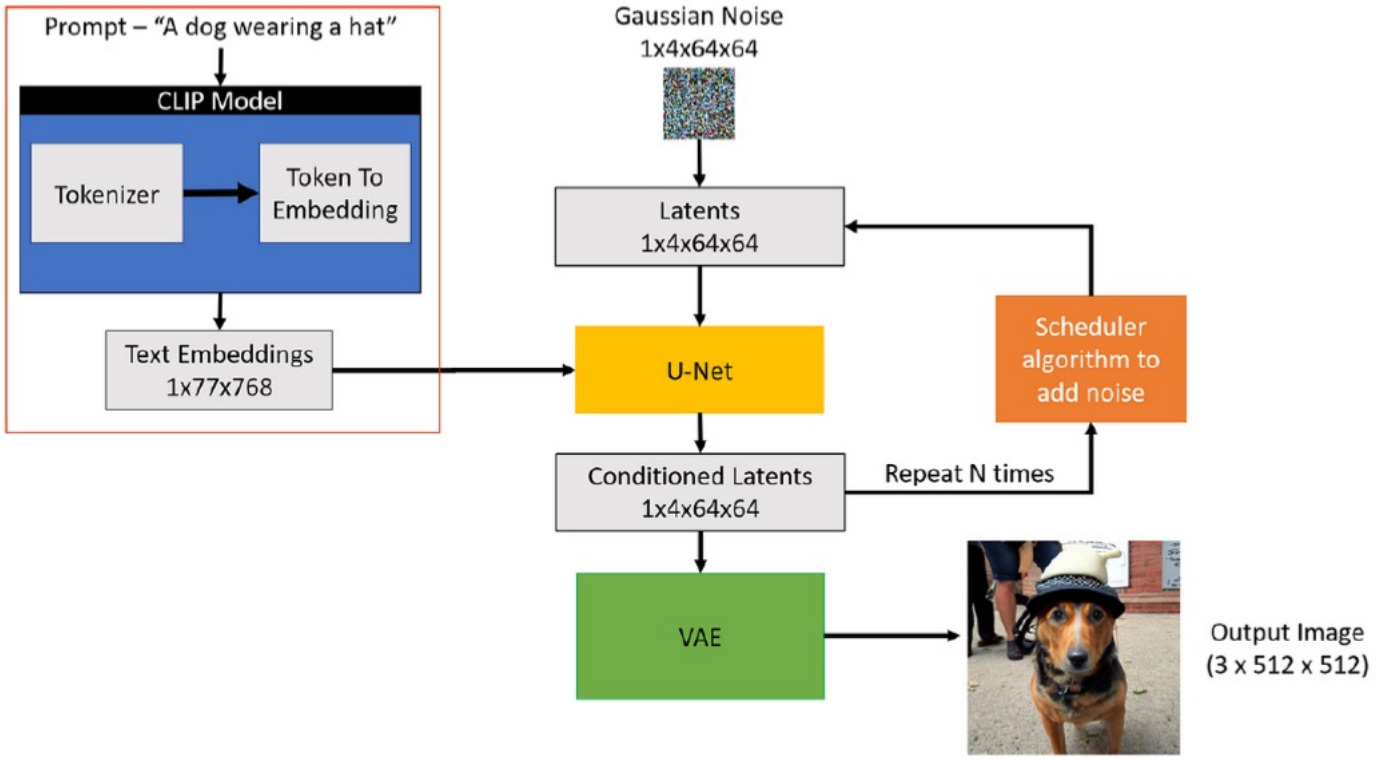
http://jalammar.github.io/illustrated-stable-diffusion/

- Alter images using text prompts:

http://jalammar.github.io/illustrated-stable-diffusion/

Stable Diffusion Architecture

- Left: A text encoder converts a text prompt y into a vector $\tau(y)$
- Right: The text embedding $\tau(y)$ is used to guide image generation in LDM

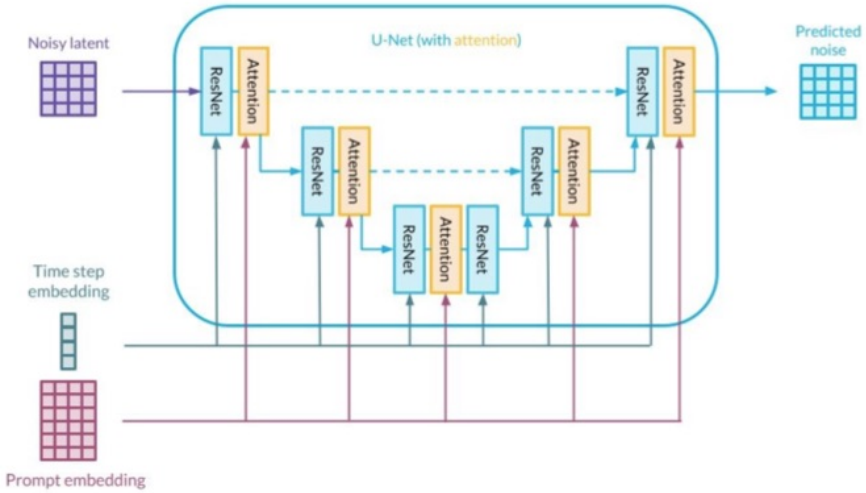


Text encoder

LDM reverse process

Text Guidance

- $\tau(y)$ is injected to the error predictor $\epsilon_\theta(\mathbf{z}_t, t, y)$ via cross-attention.



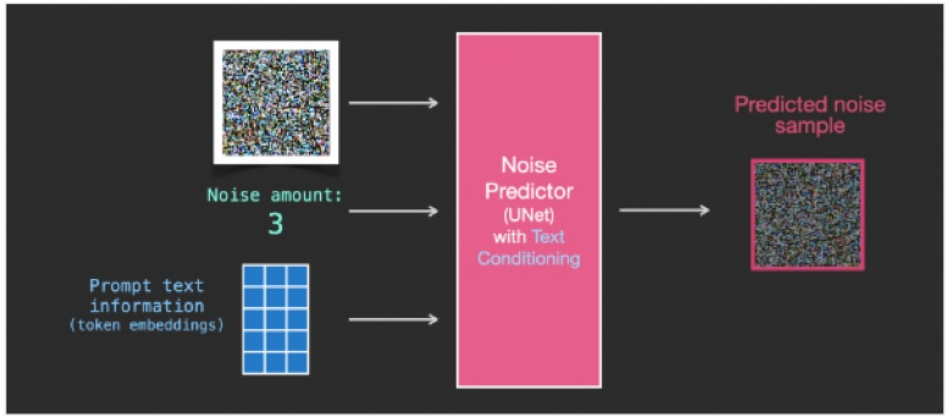
http://jalammar.github.io/illustrated-stable-diffusion/

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

Training Objective: $\min ||\bar{\epsilon}_t - \epsilon_\theta(\mathbf{z}_t, t, y)||^2$

Text Guidance

- At each step of the generation process, the text embedding serves as a reminder of the kind of image we desire.
- For example, if you have an input prompt "cat", you can think of conditioning as telling the noise predictor:
  - "For the next denoising step, the image should look more like a cat. Now go on with the next step."



http://jalammar.github.io/illustrated-stable-diffusion/

Stable Diffusion is Based on LDM

- Stable Diffusion was trained on pairs of images and captions taken from LAION-5B, a publicly available dataset consisting of 5 billion image-text pairs.
  - For each text-image pair, the image is first mapped to the latent space, and then noise is added to it repeatedly to finally get $\mathbf{z}_T$
  - Various versions z_t of the image are used to train the noise predictor $\epsilon_\theta(\mathbf{z}_t, t, y)$ to denoise z_t by taking y into consideration.

    If z_t is a blurred version of a cat, it should be denoised in one way
    If z'_t is a blurred version of a dog it should be denoised in another

- The training was carried out on 256 Nvidia A100 GPUs for a total of 150,000 GPU-hours, at a cost of $600,000.