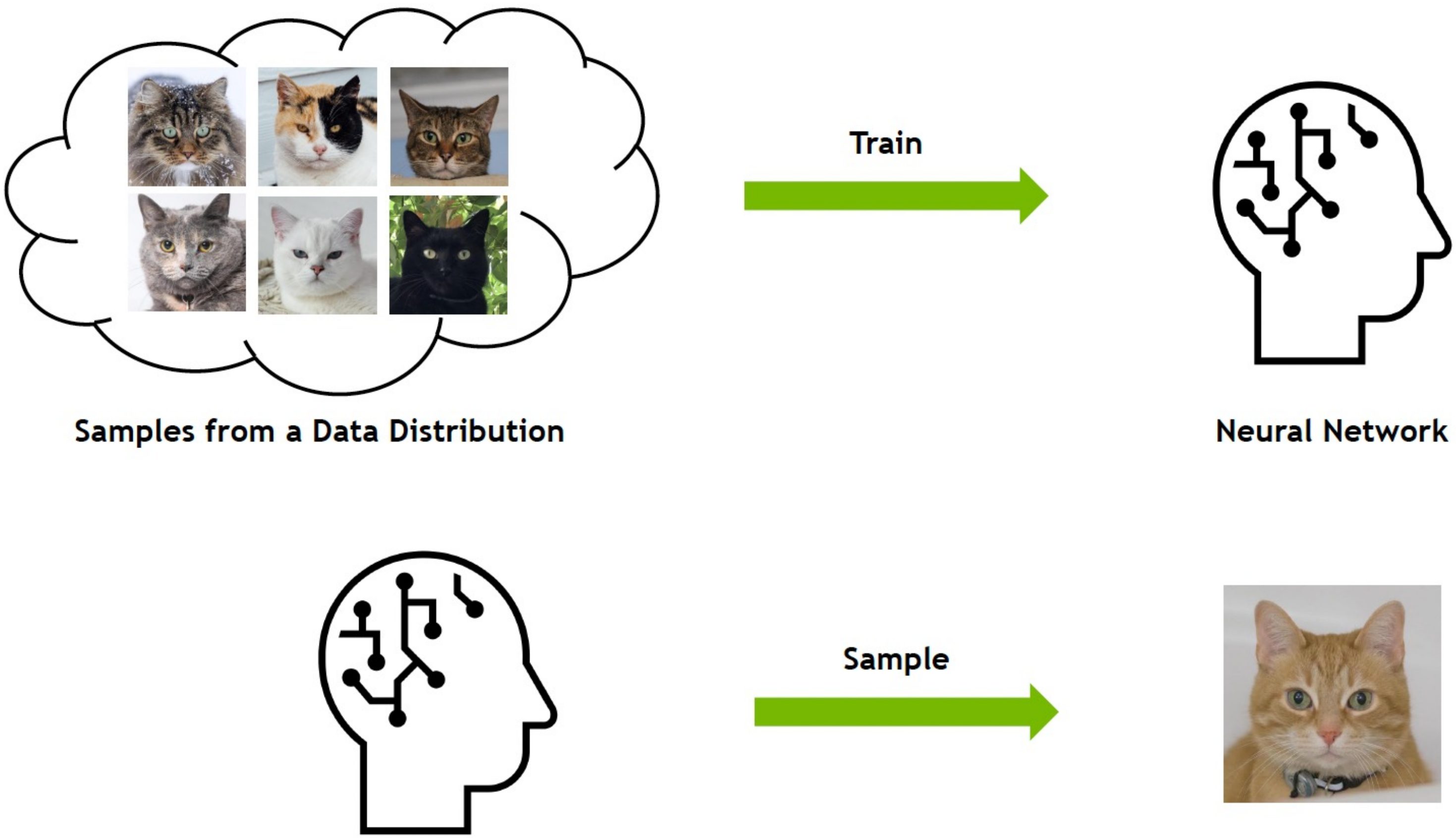
Image Generation Using Diffusion Models

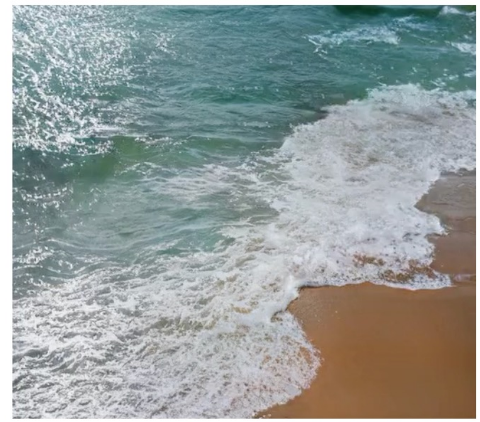
Lecture 19
Subir Varma

Deep Generative Learning: Learning to Generate Data

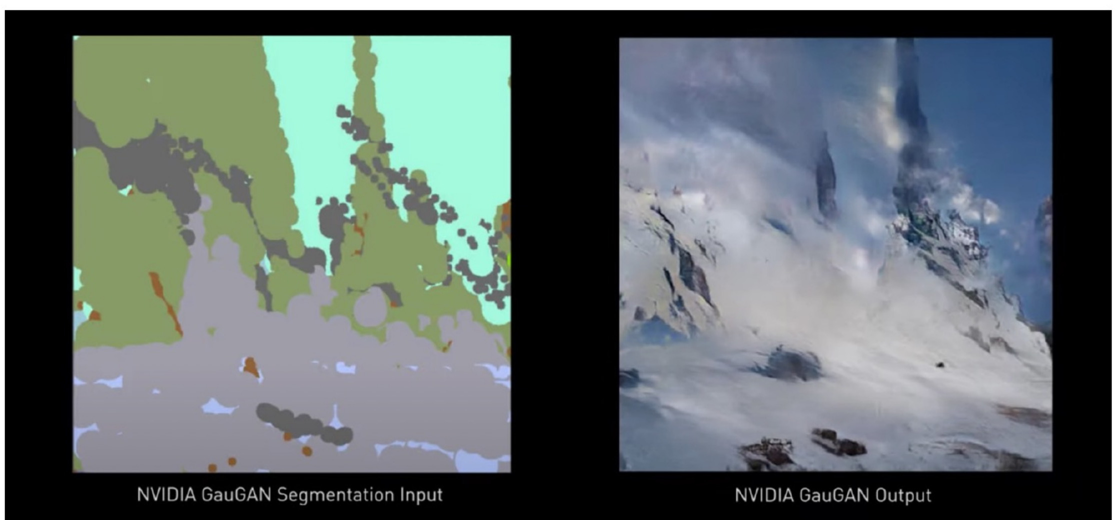


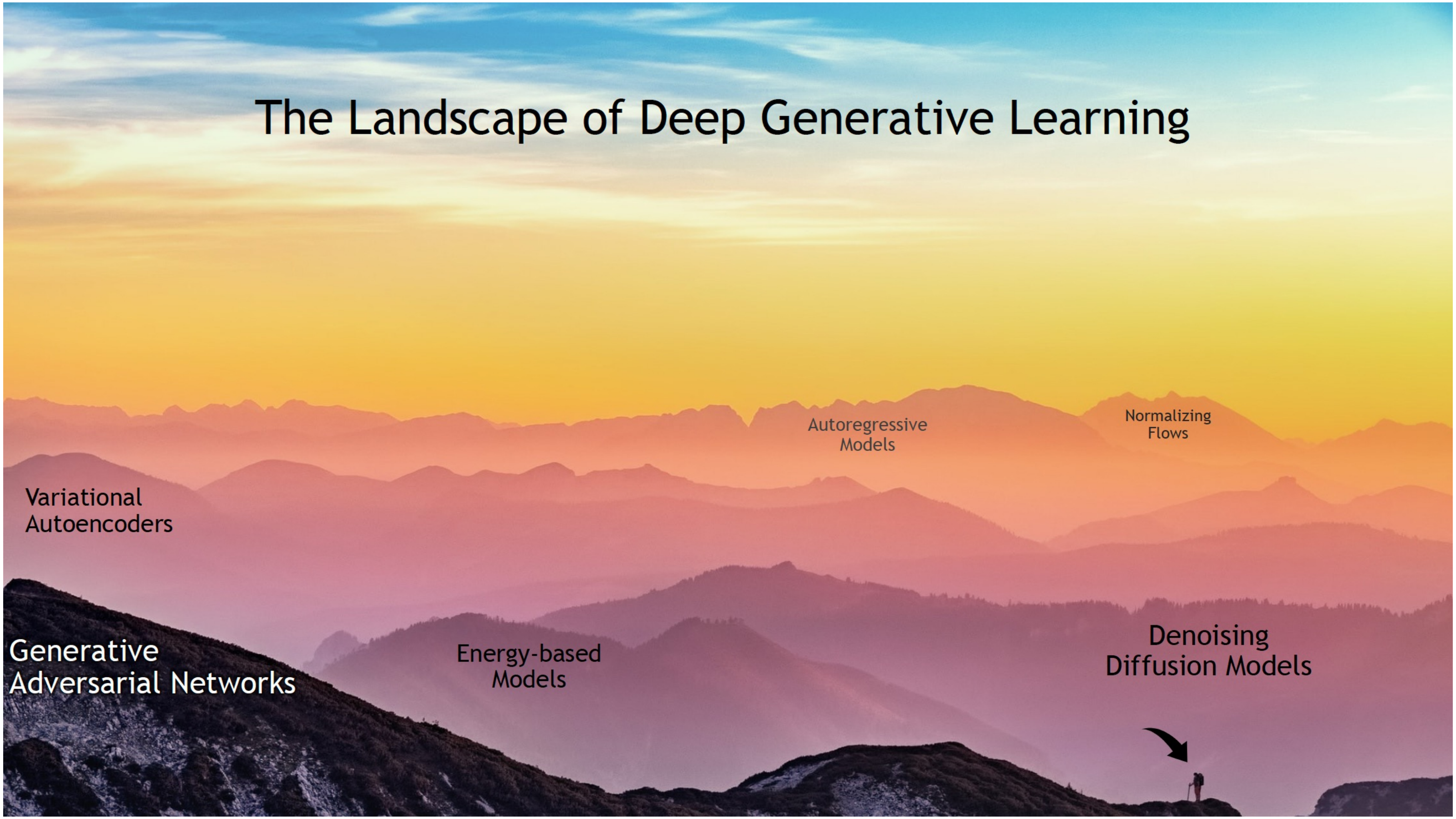
Applications

Content Generation



Artistic Tools




The Landscape of Deep Generative Learning
Autoregressive Models
Normalizing Flows
Variational Autoencoders
Generative Adversarial Networks
Energy-based Models
Denoising Diffusion Models

DeNoising Diffusion Models

Emerging as powerful generative models, outperforming GANs

"Diffusion Models Beat GANs on Image Synthesis"
Dhariwal & Nichol, OpenAI, 2021

"Cascaded Diffusion Models for High Fidelity Image Generation"
Ho et al., Google, 2021

Text to Image Generation

DALL·E 2

“a teddy bear on a skateboard in times square”

“Hierarchical Text-Conditional Image Generation with CLIP Latents” Ramesh et al., 2022

Imagen

A group of teddy bears in suit in a corporate office celebrating the birthday of their friend. There is a pizza cake on the desk.

“Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding”, Saharia et al., 2022

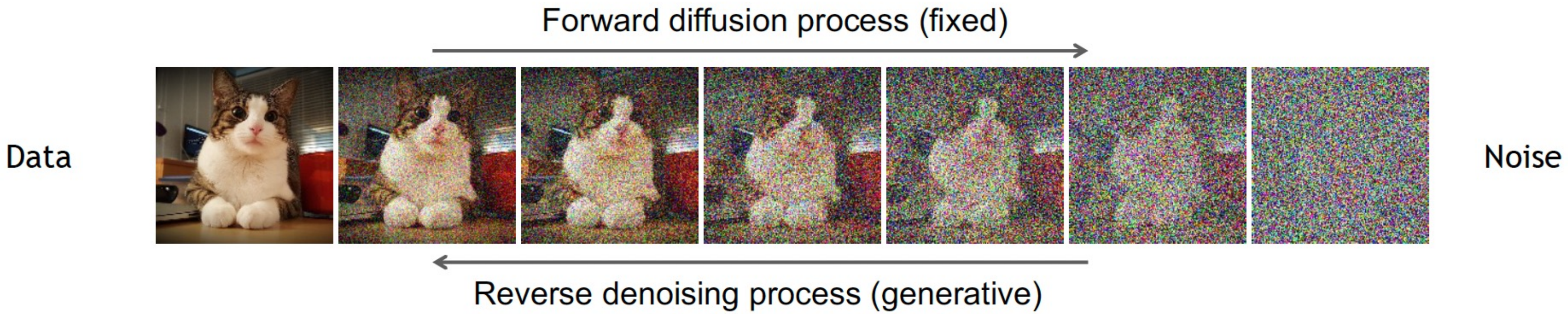
Denoising Diffusion Probabilistic Models

Denoising Diffusion Models

Learning to generate by denoising

Denoising diffusion models consist of two processes:

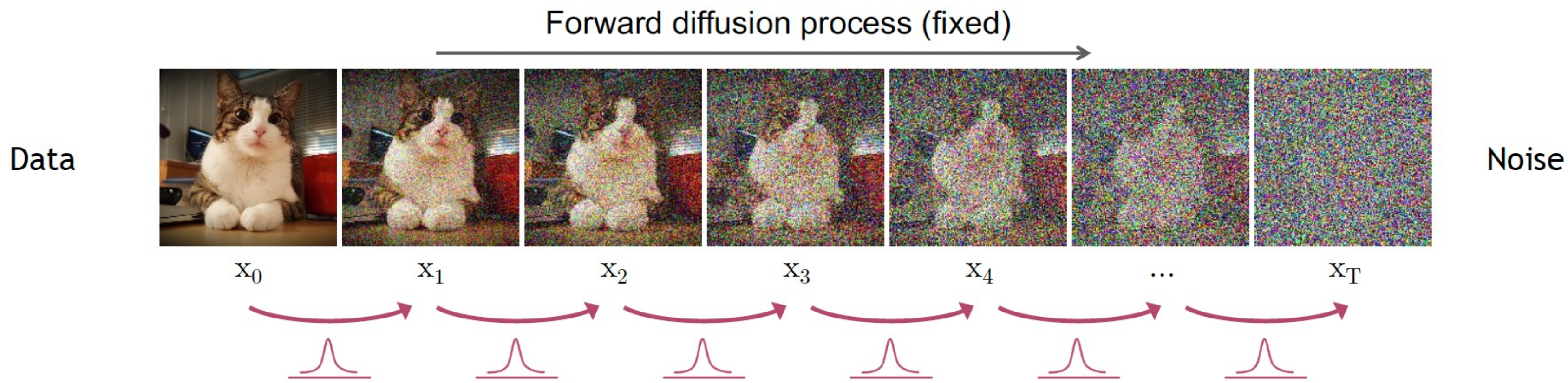
- Forward diffusion process that gradually adds noise to input
- Reverse denoising process that learns to generate data by denoising



Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

Forward Diffusion Process

The formal definition of the forward process in T steps:



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

The sequence β_i is chosen such that $\beta_1 < \beta_2 < \ldots < \beta_T < 1$, i.e., the amount of noise being added increases monotonically,

Re-Parametrization Trick

Note: We will use the notation $N(X;\mu,\Sigma)$ for a mutivariate Gaussian (or Normal) Distribution X with mean vector $\mu = (\mu_1, \ldots, \mu_N)$ and covariance matrix Σ (please see the Appendix at the end of this chapter for a short introduction to multivariate Gaussian Distributions). For the special case when the covariance matrix is a diagonal with a common variance σ^2, this reduces to $N(X;\mu,\sigma^2 I)$ where I is a $N \times N$ identity matrix.

Note: Given a Gaussian Distribution $N(X;\mu,\sigma^2 I)$, it is possible to generate a sample from it by using the **Re-Parametrization Trick**, which states that a sample X can be expressed as

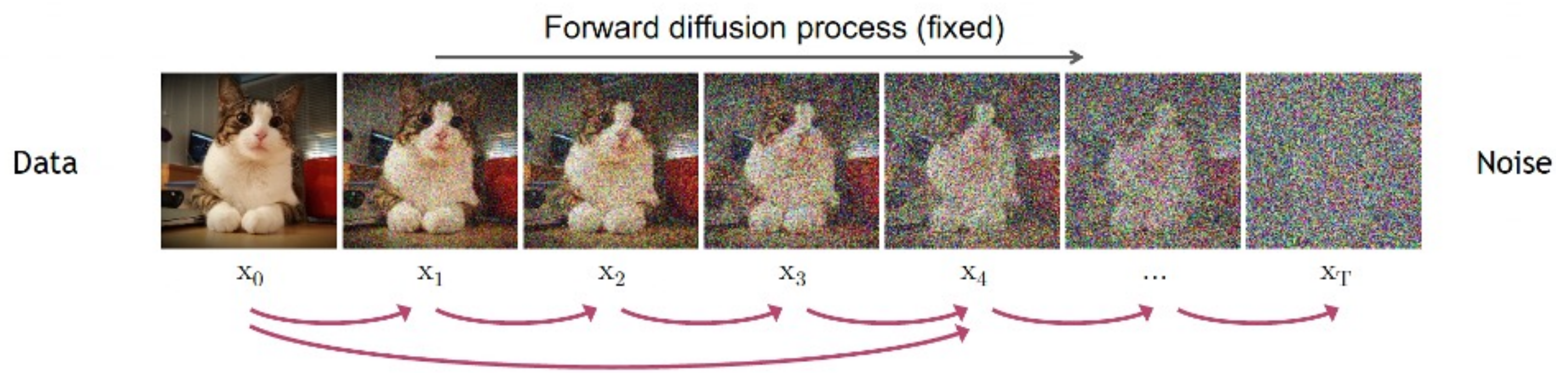
$$X = \mu + \sigma\epsilon$$

where the random vector ϵ is distributed as per the Unit Gaussian Distribution $N(0, I)$.

By using the Re-parametrization Trick, we can sample X_t from the distribution $q(X_t|X_{t-1})$

$$X_t = \sqrt{1-\beta_t}X_{t-1} + \sqrt{\beta_t}\epsilon_{t-1}$$

Diffusion Kernel



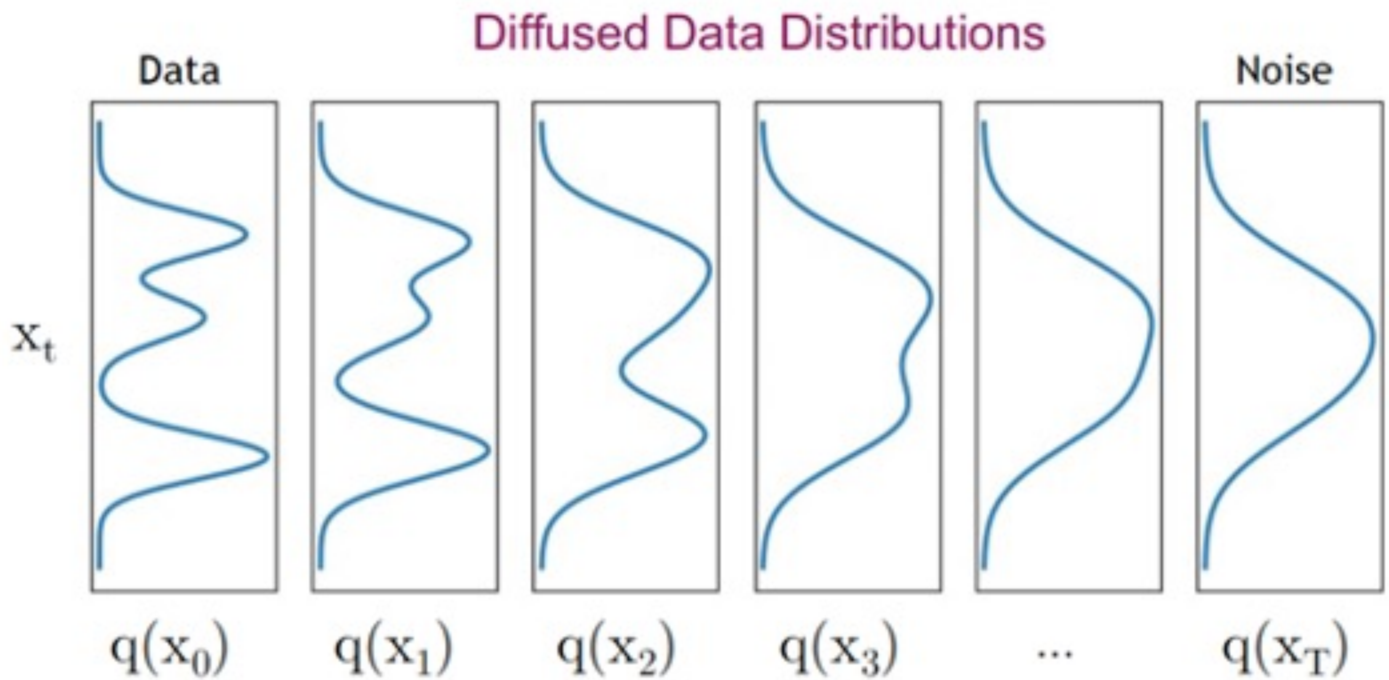
$$X_t = \sqrt{\gamma_t} X_0 + \sqrt{(1-\gamma_t)}\epsilon$$

$$\gamma_t = \prod_{i=1}^{t} \alpha_i \text{ where } \alpha_t = 1 - \beta_t$$

$\gamma_1 \geq \gamma_2 \geq \ldots \geq \gamma_T$ so that $\gamma_t \to 0$ as t increases,

this implies that the distribution of X_T approaches $N(0, I)$ which is also referred to as "white" noise.

What Happens to a Distribution in the Forward Diffusion?



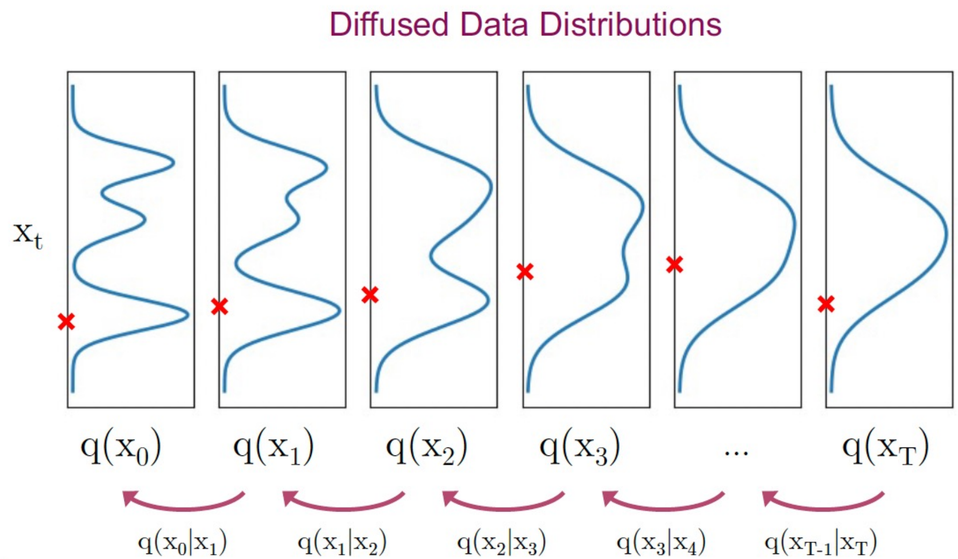
Generative Learning by Denoising

Recall, that the diffusion parameters are designed such that $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Generation:

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

Iteratively sample $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}_{\text{True Denoising Dist.}}$



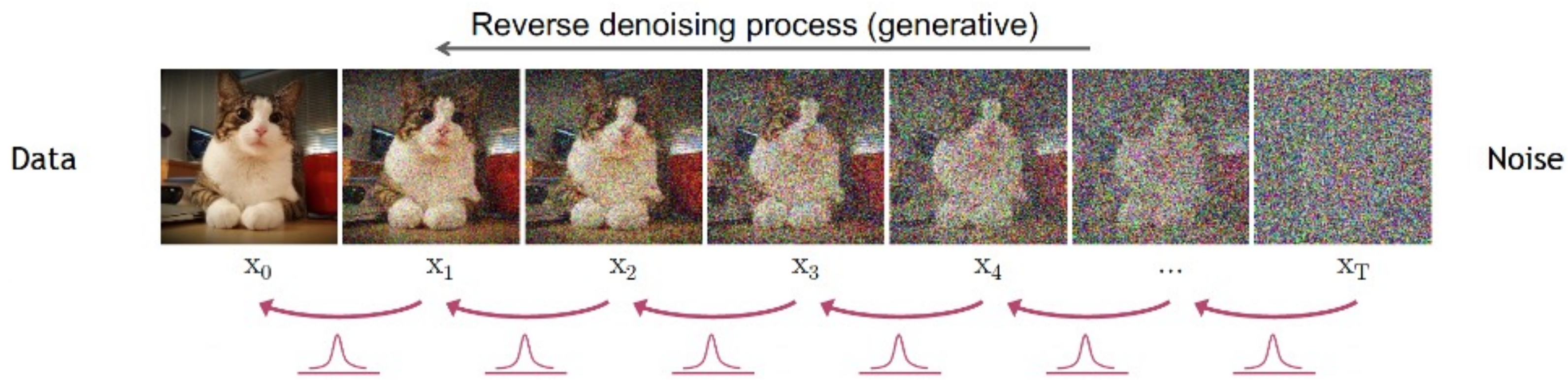
In general, $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is intractable.

Can we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$? Yes, we can use a Normal distribution if β_t is small in each forward diffusion step.

$$p_\theta(X_{t-1}|X_t) \approx q(X_{t-1}|X_t).$$

Approximation

Reverse Denoising Process

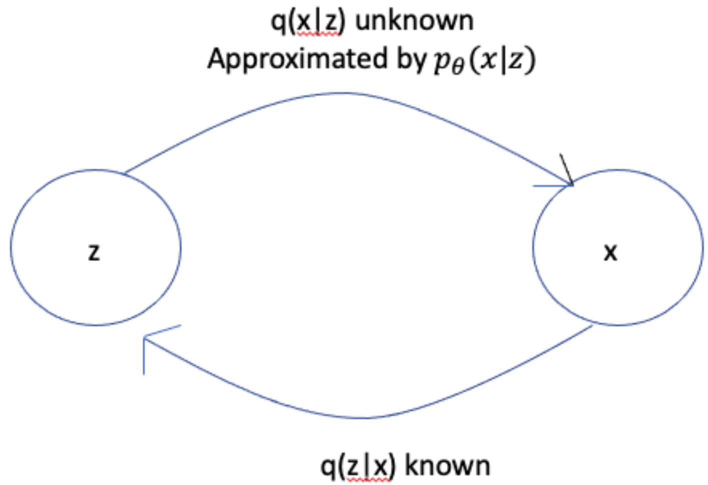


$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})$$

Approximate using a Gaussian Distribution

ELBO (Evidence Lower Bound)



Lets assume that we can approximate $q(X|Z)$ by another (parametrized) distribution $p_\theta(X|Z)$.

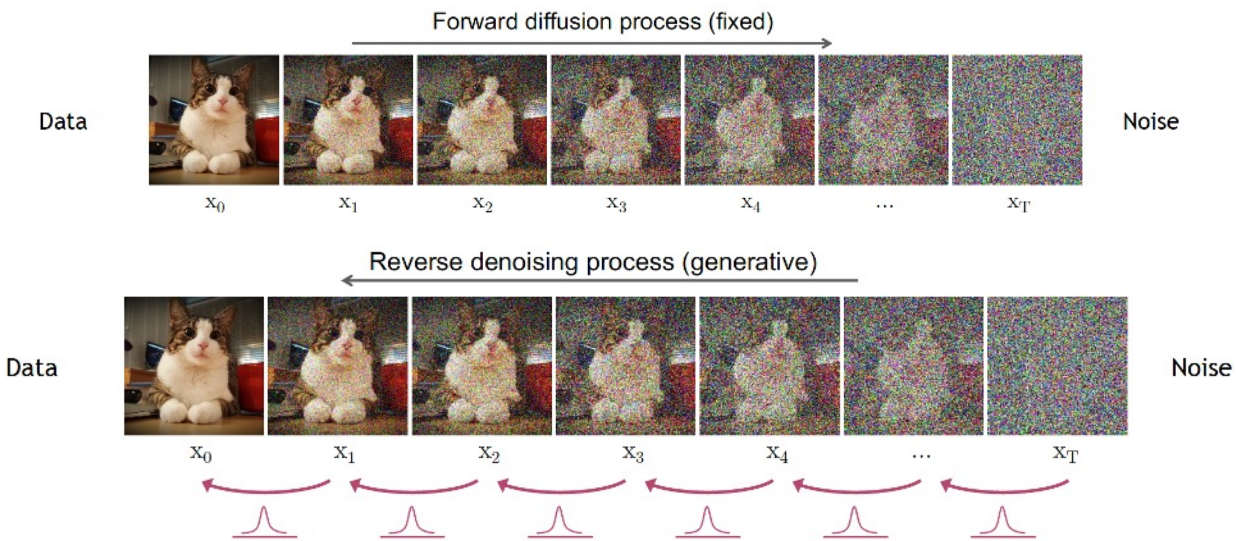
Minimize "distance" between the distributions →

$$D_{KL}(q(X|Z), p_\theta(X|Z)) = \log q(X) - \sum_Z q(Z|X) \log \frac{p_\theta(X,Z)}{q(Z|X)}$$

This is equivalent to minimizing →

$$ELBO = \sum_Z q(Z|X) \log \frac{q(Z|X)}{p_\theta(X,Z)}$$

Extend ELBO to the Chain



$$L_{ELBO} = E_{q(X_{0:T})} \frac{\log q(X_{1:T}|X_0)}{p_\theta(X_{0:T})}$$

with

$$p_\theta(X_{0:T}) = p(X_T) \prod_{t=1}^{T} p_\theta(X_{t-1}|X_t)$$

$$q(X_{1:T}|X_0) = \prod_{t=1}^{T} q(X_t|X_{t-1})$$

ELBO Formula

$$L_{ELBO} = E_q\left[\log\frac{q(X_T|X_0)}{p_\theta(X_T)} + \sum_{t=2}^{T}\log\frac{q(X_{t-1}|X_t,X_0)}{p_\theta(X_{t-1}|X_t)} - \log p_\theta(X_0|X_1)\right]$$

Which is the same as

$$L_{ELBO} = E_q\left[D_{KL}(q(X_T|X_0)||p_\theta(X_T)) + \sum_{t=2}^{T}D_{KL}(q(X_{t-1}|X_t,X_0)||p_\theta(X_{t-1}|X_t)) - \log p_\theta(X_0|X_1)\right]$$

Defining

$$L_T = D_{KL}[q(X_T|X_0)||p_\theta(X_T)]$$
$$L_t = D_{KL}\,[q(X_t|X_{t+1},X_0)||p_\theta(X_t|X_{t+1})] \quad 1 \le t \le T-1$$
$$L_0 = -\log p_\theta(X_0|X_1)$$

L_{ELBO} can be written as

$$L_{ELBO} = L_T + L_{T-1} + \ldots + L_0$$

Critical Formula

Gaussian Distribution!

$$q(X_{t-1}|X_t, X_0) = N(X_{t-1}; \tilde{\mu}(X_t, X_0), \tilde{\beta}_t I)$$

where

$$\tilde{\beta}_t = \frac{1-\gamma_{t-1}}{1-\gamma_t}\beta_t$$

$$\tilde{\mu}(X_t, X_0) = \frac{\sqrt{\alpha_t}(1-\gamma_{t-1})}{1-\gamma_t}X_t + \frac{\sqrt{\gamma_{t-1}}\beta_t}{1-\gamma_t}X_0$$

This implies

$$L_t = E\left[\frac{1}{2||\Sigma_\theta(X_t, t)||_2^2}||\tilde{\mu}_t(X_t, X_0) - \mu_\theta(X_t, t)||^2\right] \quad 1 \le t \le T-1$$

Loss Function

$$L_t = E\left[\frac{1}{2||\Sigma_\theta(X_t,t)||_2^2}||\tilde{\mu}_t(X_t,X_0)-\mu_\theta(X_t,t)||^2\right] \quad 1 \le t \le T-1$$

Is equivalent to

$$L_t = E\left[\frac{\beta_t^2}{2\alpha_t(1-\gamma_t)||\Sigma_\theta(X_t,t)||_2^2}||\epsilon_t-\epsilon_\theta(X_t,t)||^2\right]$$

Estimate of that noise

ϵ_t is the noise that is added to the image X_0 during training in order to get X_t

Follows from $\tilde{\mu}(X_t,X_0) = \frac{\sqrt{\alpha_t}(1-\gamma_{t-1})}{1-\gamma_t}X_t + \frac{\sqrt{\gamma_{t-1}}\beta_t}{1-\gamma_t}X_0$ and $X_0 = \frac{1}{\sqrt{\gamma_t}}(X_t - \sqrt{1-\gamma_t}\epsilon_t)$

So that $\tilde{\mu}(X_t,X_0) = \frac{1}{\sqrt{\alpha_t}}\left\{X_t - \frac{\beta_t}{\sqrt{1-\gamma_t}}\epsilon_t\right\}$

Training Objective Weighing

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \Bigg[\underbrace{\frac{\beta_t^2}{2\sigma_t^2 (1-\beta_t)(1-\bar{\alpha}_t)}}_{\lambda_t} ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, t)||^2 \Bigg]$$

The time dependent λ_t ensures that the training objective is weighted properly for the maximum data likelihood training.

However, this weight is often very large for small t's.

Ho et al. NeurIPS 2020 observe that simply setting $\lambda_t = 1$ improves sample quality. So, they propose to use:

$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t \sim \mathcal{U}(1, T)} \Big[||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon}_{\mathbf{x}_t}, t)||^2 \Big]$$

DDPM Algorithm: Training

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

- An image is sampled from the training dataset, with (unknown) distribution $q(X_0)$
- A single step index t of the forward diffusion process is sampled from the set $1, \ldots, T$, using an Uniform Distribution
- A random noise vector ϵ is sampled from the Gaussian Distribution $N(0, I)$
- The noisy image X_t is sampled at the t^{th} step, given by $X_t = \sqrt{\bar{\gamma}_t}X_0 + \sqrt{1-\gamma_t}\epsilon$. This is then fed into the Deep Learning model to generate an estimate of the noise that was added to X_0 in order to obtain X_t, given by $\epsilon_\theta(\sqrt{\bar{\gamma}_t}X_0 + \sqrt{1-\gamma_t}\epsilon, t)$.
- The difference between the actual noise sample ϵ and its estimate ϵ_θ is used to generate a gradient descent step.

DDPM Algorithm: Sampling

Algorithm 2 Sampling

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

Once we have a trained model, we can use it to generate new images by using the procedure outlined in Algorithm 2. At the first step we start with a Gaussian noise sample X_T, and then gradually de-noise it in steps $X_{T-1}, X_{T-2}, \ldots, X_1$ until we get to the final image X_0. The de-noising is carried out by sampling from the Gaussian Distribution $N(\mu_\theta(X_t, t), \beta_t I)$, so that

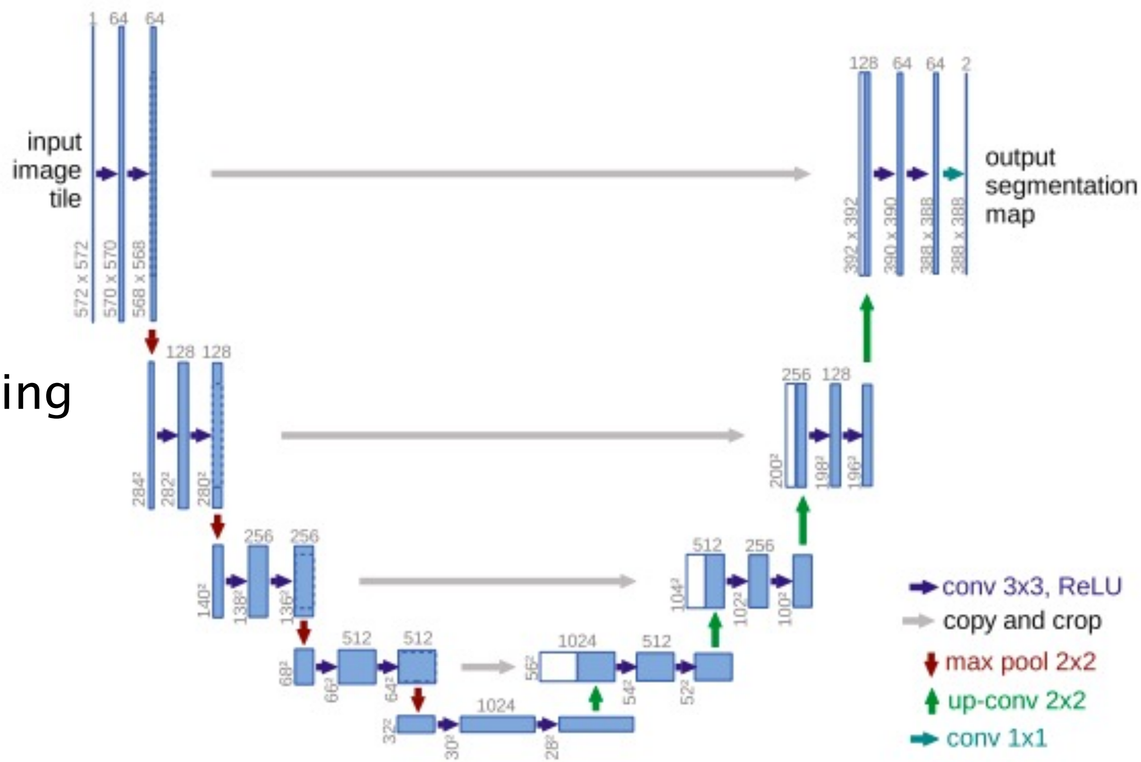
$$X_{t-1} = \mu_\theta(X_t, t) + \sqrt{\beta_t}\epsilon$$

$\mu_\theta(X_t, t)$ is computed by running the model to estimate ϵ_θ and then using the following equation to get μ_θ

$$\mu_\theta(X_t, t) = \frac{1}{\sqrt{\alpha_t}} \left\{ X_t - \frac{\beta_t}{\sqrt{1-\gamma_t}} \epsilon_\theta(X_t, t) \right\}$$

The Neural Network

Downsampling



Upsampling

Use Conv2DTranspose
(see Chollet page 245)

Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Conv2DTranspose

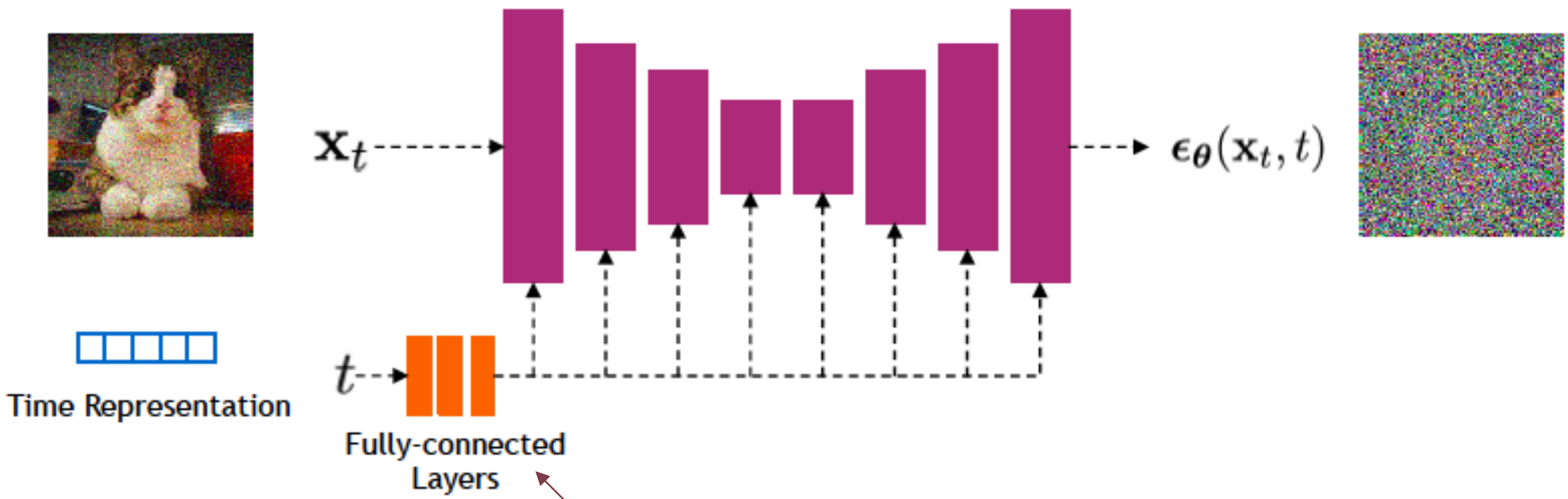
Example:

(100,100,64) → Conv2D(128,3,strides=2, padding='same') → (50,50,128)

(50,50,128) → Conv2DTranspose(64,3,strides=2, padding='same') → (100,100,64)

Inverse Convolutions!

The Neural Network (cont)



Time representation is similar to positional embedding in Transformers

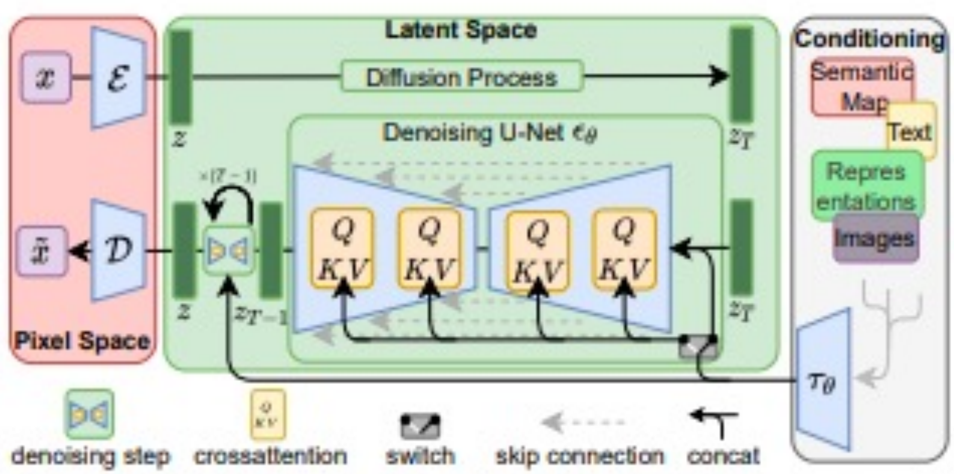
Latent Diffusion Models

Two aspects to image re-construction:

- Semantic Re-Construction: Reconstructing specific objects and how they relate to each other
- Texture Re-Construction: Difference in pixel content

Diffusion Models excel at Semantic Re-Construction, but spend a lot of their processing on Texture Re-Construction

Latent Diffusion Models



- Part 1 of the model (in red) consists of an autoencoder whose Encoder E converts the original image X into a lower dimensional image Z in the Latent Space that is perceptually equivalent to the image space, but a significantly reduced computational complexity (since the high frequency, imperceptible details are abstracted away). This autoencoder is trained by a combination of perceptual loss and patch based adversarial objective (see Esser, Rombach, Ommer).
- A Diffusion Model is then run on this modified image (in green). The reconstructed image Z is fed into the Decoder of the Auto Encoder to get the final image $\hat{X}_0$. Since the Diffusion Model operates on a much lower dimensional space, they are much more computationally efficient. Also it can focus on important, semantic bits of data rather than the imperceptible high frequency content.

Conditional Diffusion Models

Text-to-image generation

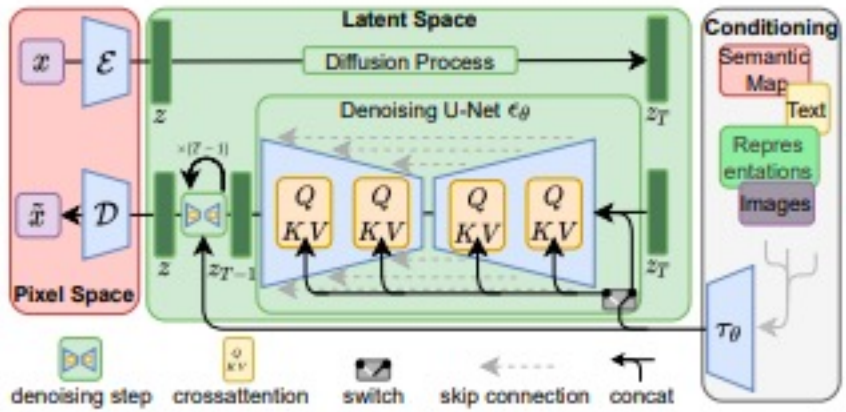
DALL·E 2

“a propaganda poster depicting a cat dressed as french emperor napoleon holding a piece of cheese”

IMAGEN

“A photo of a raccoon wearing an astronaut helmet, looking out of the window at night.”

Conditional Diffusion Models



- As shown in the box on the right, the tokenized input y is first passed through a network τ_θ that converts it into a sequence $\zeta^{M\times d_\tau}$, where M is the length of the sequence and d_τ is the size of the individual vectors. This conversion is done by means of an unmasked Transformer that is implemented using N Transformer blocks consisting of global self-attention, layer-normalization and position-wise MLP layers.
- ζ is mapped on to each stage of the de-noising section of the Diffusion Model using a cross-attention mechanism as shown in Figure **gen17**. In order to do so, the Self Attention modules in the ablated UNet model (see Dhariwal and Nichol for a description) are replaced by a full Transformer consisting of T blocks of with alternating layers of self-attention, position-wise MLP and cross-attention. The exact structure is shown in Figure **gen18**, with the shapes of the various layers involved.

The Cross Attention is implemented by using the flattened "image" tensor of shape $h.w \times d.n_h$ to generate the Query, and using the text tensor of shape $M \times d_\tau$ to generate the Key and Value.

input	$\mathbb{R}^{h\times w\times c}$
LayerNorm	$\mathbb{R}^{h\times w\times c}$
Conv1x1	$\mathbb{R}^{h\times w\times d\cdot n_h}$
Reshape	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
$\times T$ { SelfAttention	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
MLP	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
CrossAttention	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
Reshape	$\mathbb{R}^{h\times w\times d\cdot n_h}$
Conv1x1	$\mathbb{R}^{h\times w\times c}$

Further Reading

- Das and Varma: Chapter Image Generation Using Diffusion Models, Chapter GenerativeModels