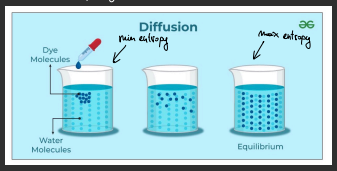
Used for:

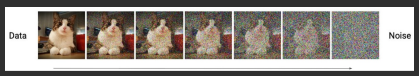
- image/video generation
- remove objects from an image
- drug discovery
- Whatever

The idea of "diffusion"

In physics



In data science:



the entropy increase adding noise

destruction of information

OVERVIEW - 3 STEPS

Forward process

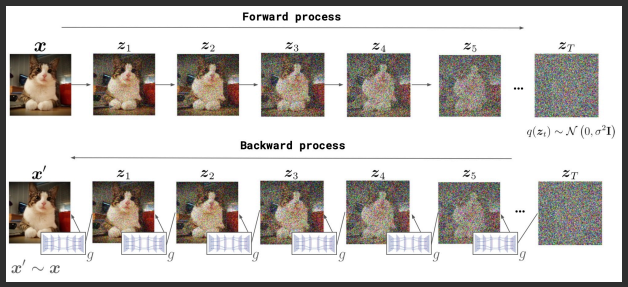
Add T times noise to data.

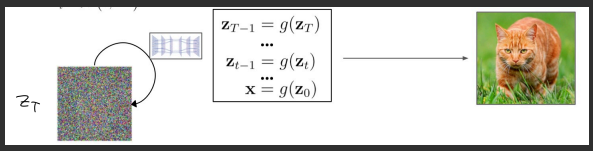
Backward process

- Remove the noise using a deep learning model.
- Being able to remove noise, it means that we learned an approximation of $P(X)$

Generate images

After training the model we sample $z_T \sim \mathcal{N}(0, \sigma^2 I)$ and reconstruct an image using the model:





Forward Process

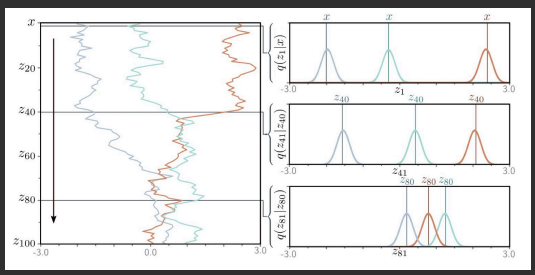
- How to add noise to x, T times:

$$\mathbf{z}_1 = \sqrt{1-\beta_1} \cdot \mathbf{x} + \sqrt{\beta_1} \cdot \epsilon_1 \qquad \epsilon_t \sim \mathcal{N}(0, \sigma^2 \mathbf{I}), \quad \beta_t \in [0,1]$$

$$\mathbf{z}_t = \sqrt{1-\beta_t} \cdot \mathbf{z}_{t-1} + \sqrt{\beta_t} \cdot \epsilon_t \qquad \forall t \in 2, \ldots, T$$

β_t balances how fast we move towards the Normal

- Example: $x \in \mathbb{R}$, $T = 100$



3 Connection to Theoretical Concepts

This process is deeply related to **Stochastic Processes** and **Markov Chains**:

📌 Connection to Markov Chains

The transition from z_t to z_{t-1} follows a **Markov process**, meaning:

$$q(z_t|z_{t-1}, z_{t-2}, \ldots, z) = q(z_t|z_{t-1})$$

That is, each step **only depends on the previous step** and not on the full history.

$$q(\mathbf{z}_1 \mid \mathbf{x}) = Norm_{\mathbf{z}_1}[\sqrt{1-\beta_1}\mathbf{x}, \beta_1 \mathbf{I}]$$
$$q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) = Norm_{\mathbf{z}_t}[\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t \mathbf{I}] \qquad \forall t \in 2, \ldots, T$$

📌 Connection to the Central Limit Theorem (CLT)

- The final state (z_T) **resembles a Gaussian distribution**.
- This can be **explained by the Central Limit Theorem**, which states that if we apply **enough small random transformations**, the result will tend toward a normal distribution.

The more steps we take (T) the more $x \sim \mathcal{N}(0,1)$

So the Joint distribution is:

$$q(\mathbf{z}_{1\ldots T} \mid \mathbf{x}) = q(\mathbf{x} \mid \mathbf{z}_1) \prod_{t=2}^{T} q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) = q(x|z_1)\, q(z_2|z_1)\, q(z_3|z_2) \ldots q(z_T|z_{T-1})$$

Optimize the Forward Process – Diffusion Kernel

Instead of applying t times the noise, there is another efficient way:

Not efficient:

$$\mathbf{z}_1 = \sqrt{1-\beta_1} \cdot \mathbf{x} + \sqrt{\beta_1} \cdot \epsilon_1$$
$$\mathbf{z}_t = \sqrt{1-\beta_t} \cdot \mathbf{z}_{t-1} + \sqrt{\beta_t} \cdot \epsilon_t \qquad \forall t \in 2, \ldots, T$$

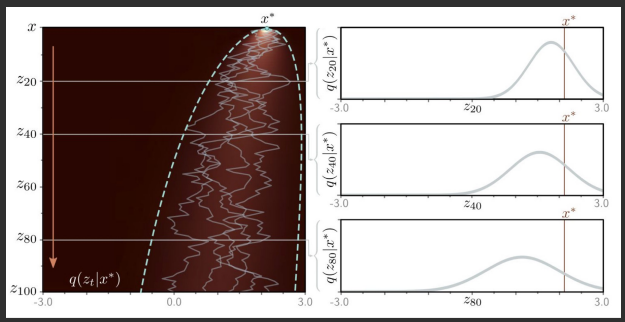
$$q(\mathbf{z}_1 \mid \mathbf{x}) = Norm_{\mathbf{z}_1}[\sqrt{1-\beta_1}\mathbf{x}, \beta_1 \mathbf{I}]$$
$$q(\mathbf{z}_t \mid \mathbf{z}_{t-1}) = Norm_{\mathbf{z}_t}[\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t \mathbf{I}] \qquad \forall t \in 2, \ldots, T$$

$$\Longrightarrow$$

$$\underbrace{\mathbf{z}_t = \sqrt{\alpha_t} \cdot \mathbf{x}}_{\text{In one step we compute } z_t} + \sqrt{1-\alpha_t} \cdot \epsilon_t \qquad \alpha_t = \prod_{s=1}^{t}(1-\beta_s)$$

- This term **accumulates** the effect of all past noise steps.
- It acts like a "scaling factor" that controls how much of the original image x remains in z_t.

$$q(\mathbf{z}_t \mid \mathbf{x}) = Norm_{\mathbf{z}_t}[\sqrt{\alpha_t} \cdot \mathbf{x}, (1-\alpha_t)\mathbf{I}]$$



Each small plot shows how the probability distribution of z_t evolves over time, **now computed using the diffusion kernel**:

- **Top plot:** $q(z_{20}|x^*)$
  - The distribution is **narrow and centered** around x^*.
  - At early stages, z_t is still **close** to the original data.
- **Middle plot:** $q(z_{40}|x^*)$
  - The distribution **widens**, meaning more uncertainty has been introduced.
  - The mean is still **near** x^*, but there is now **a spread due to added noise**.
- **Bottom plot:** $q(z_{80}|x^*)$
  - The distribution becomes **much wider**, indicating that z_t is mostly noise.
  - x^* is now **far in the tail of the distribution**, meaning its original structure is almost lost.

Since we use the **diffusion kernel**, each of these distributions is computed **analytically**, without needing to simulate all the previous time steps.

Backward Process

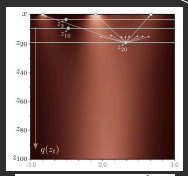
- We need to compute $q(z_{t-1}|z_t)$:

$$q(z_{t-1}|z_t) \underset{\text{bayes}}{=} \frac{q(z_t|z_{t-1})\, q(z_{t-1})}{q(z_t)} \quad \text{Where}$$

$q(z_t|z_{t-1})$ is known from the forward process

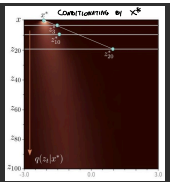
$$q(z_{t-1}) \overset{\text{total probability}}{=} \int q(z_{t-1}|x)\, P(x)\, dx$$ cannot be computed since we don't know $P(X)$

$q(z_t) = ???$



to solve this problem we condition also by x (the data we started with)

$$q(z_{t-1}|z_t, x) = \frac{q(z_t|z_{t-1}, x)\, q(z_{t-1}|x)}{q(z_t|x)} \propto q(z_t|z_{t-1}, x)\, q(z_{t-1}|x) \overset{\text{markov}}{=} q(z_t|z_{t-1})\, \overbrace{q(z_{t-1}|x)}^{\text{Diffusion Kernel}}$$



Conclusion:

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

CONCLUSION

The params chosen are usually:

$$T >> 1 \;,\; \beta_t \longrightarrow 0$$

This make possible:

$$q(z_{t-1} \mid z_t, x)$$

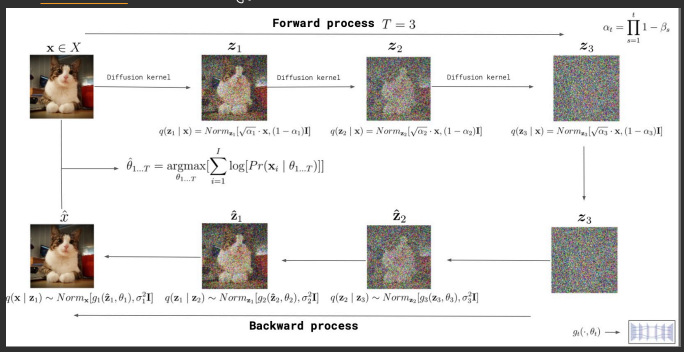
to be close to:

$$q(z_{t-1} \mid z_t)$$

So it does not depend on x anymore.

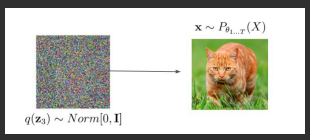
TRAINING

learn $g_t(\cdot, \theta_t)$



INFERENCE

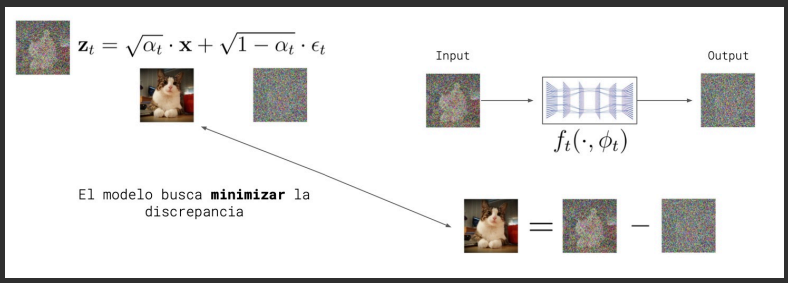
found $g_t(\cdot, \theta_t)$ i sample z_3 from $q(z_3) \sim Norm[0, I]$ and



i let the backward process reconstruct the image with the sampled $q(z_3)$

Reparametrization

- z_t is seen as $z_t = \sqrt{\alpha_t} + \sqrt{1-\alpha_t}\,\varepsilon_t$
- z_t is separated in two components so that the model learn how to remove noise



Algorithm 18.1: Diffusion model training

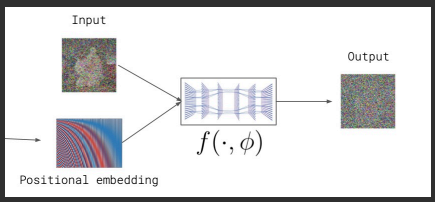
Input: Training data $\mathbf{x}$
Output: Model parameters ϕ_t
repeat
  for $i \in \mathcal{B}$ **do** // For every training example index in batch
    $t \sim \text{Uniform}[1, \ldots T]$ // Sample random timestep
    $\boldsymbol{\epsilon} \sim \text{Norm}[\mathbf{0}, \mathbf{I}]$ // Sample noise
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

Stable diffusion

- instead of training $f_t(\cdot, \phi_t)$ for each $t \in [\![1, t]\!]$ we learn $f(\cdot, \phi)$



Time-Step Information: The embedding tells the model, "We're currently at step t." Because the noise level and the denoising strategy differ for early vs. late steps, the model needs to know *which step* it's on.

- Usually $f(\cdot, \phi)$ is a U-Nets

What Does the U-Net $f(\cdot; \phi)$ Do?

- **Input**: A noisy image (e.g., z_t) plus the step embedding t.
- **Output**:
  - Depending on the implementation, the U-Net may predict the **noise** present in the image (which is then subtracted to denoise),
  - **Or** it may directly output a partially denoised version of the image.

By repeating this process step by step, the noise is gradually removed until the final (clean) image is reached at $t = 0$.

U-Nets

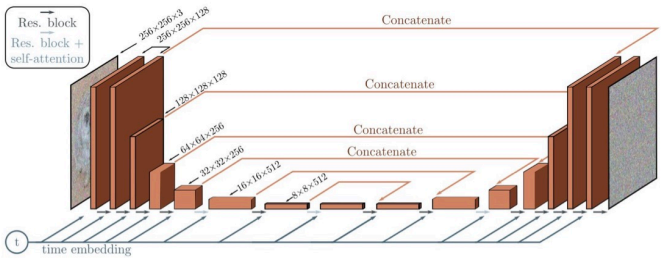


Figure 18.9 U-Net as used in diffusion models for images. The network aims to predict the noise that was added to the image. It consists of an encoder which reduces the scale and increases the number of channels and a decoder which increases the scale and reduces the number of channels. The encoder representations 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 (figure 12.5) through a shallow neural network and adding the result to the channels at every spatial position at every stage of the U-Net.

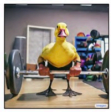
Conclusions

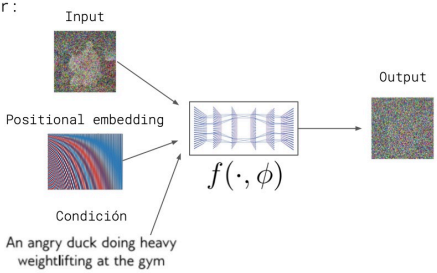
- **Diffusion models** allow us to approximate $P(X)$ by **reconstructing** the destruction of information (backward process) performed on x by $P(X)$ (forward process).
- If the **number of steps** in this destruction is sufficiently large, the reconstructions **approach a normal distribution**.
- After applying various **reparameterizations**, the training process **becomes simpler**, with a **cost function equivalent to Mean Squared Error (MSE)**.
- Diffusion models allow **generating data** with **high realism and variety**.
- The **only drawback** is that you need to **apply the deep learning model (U-Net)** T **times** to generate a single sample.
- There are solutions to this issue, such as **Stable Diffusion**.

Condicionate the model

¿Cómo condicionar?

Supongamos que:

$\boldsymbol{x} \longrightarrow$ 

$\mathbf{y} \longrightarrow$ An angry duck doing heavy weightlifting at the gym

Entonces podemos hacer: