

Chapter 2

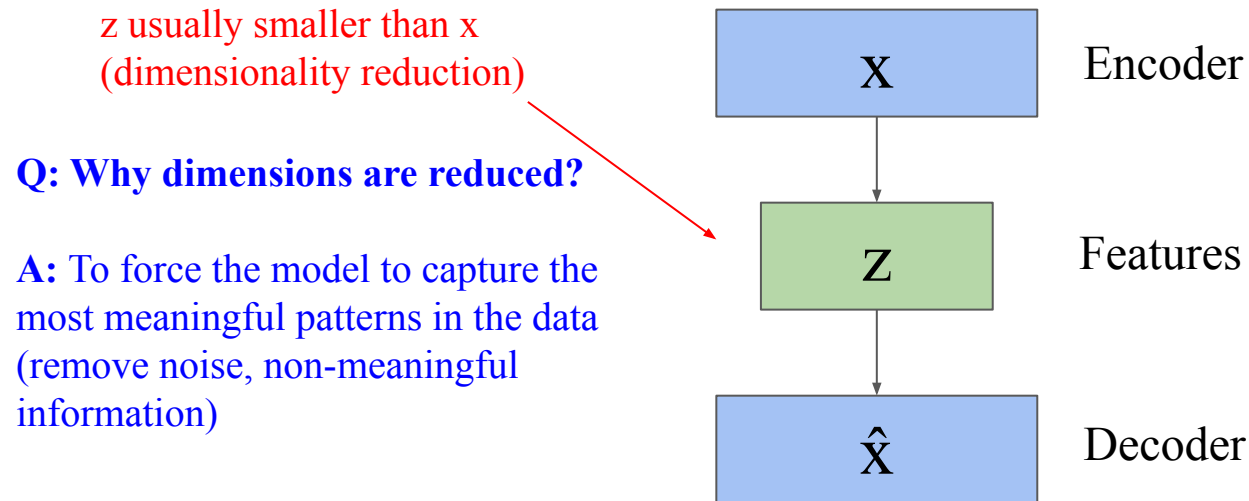
Variational Autoencoders and Diffusion Models

Outline

- Autoencoder Architecture Recap
- Variational Autoencoders (VAE)
- Diffusion Models: DDPM, DDIM
- Stable Diffusion and Latent Diffusion
- Comparison: VAEs vs Diffusion

Autoencoders

- Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data.



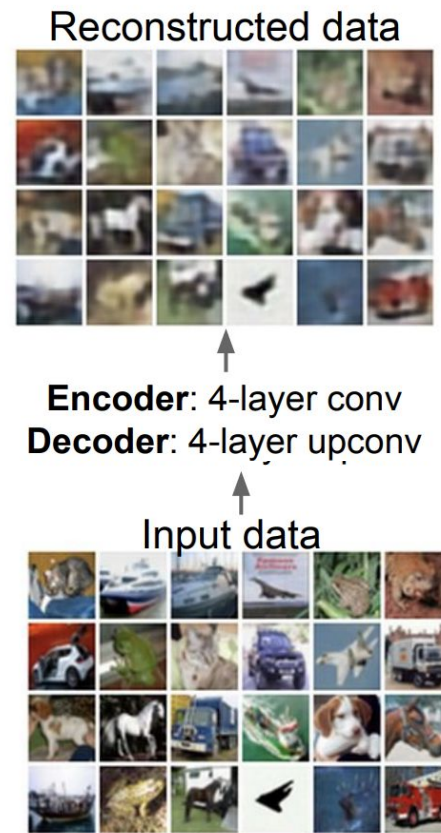
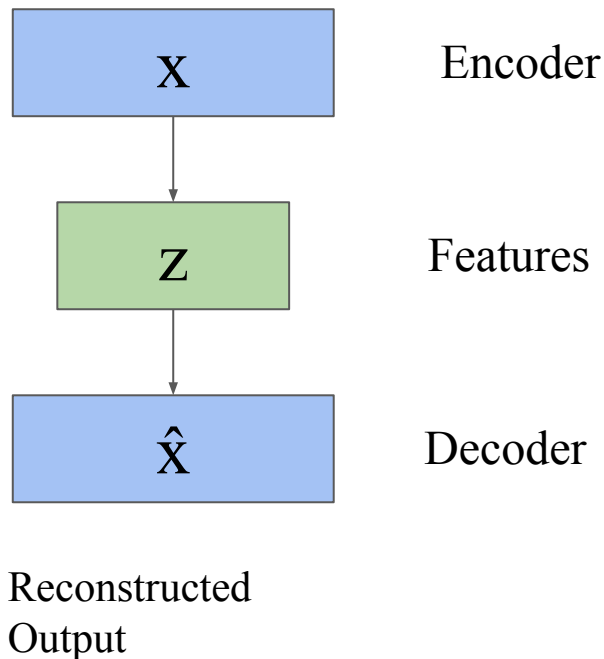
- Encoder:** Compresses input data (x) into a lower-dimensional representation (z).
- Decoder:** Reconstructs the original input from z (aiming for $x \approx$ reconstructed x).

Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data

“Autoencoding” - encoding input itself



Learning Feature Representations in an Autoencoder

1. Design:

Encoder:

- Maps the input data \mathbf{x} to a lower-dimensional space \mathbf{z} (latent representation).

$$z = f_{\theta}(x)$$

Decoder:

- Reconstructs the input from \mathbf{z} back to $\hat{\mathbf{X}}$

$$\hat{x} = g_{\phi}(z)$$

Learning Feature Representations in an Autoencoder

2. Training :

The autoencoder learns by minimizing the reconstruction loss:

$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

(Which loss function to be used depends on data)

3. Backpropagation and Optimization :

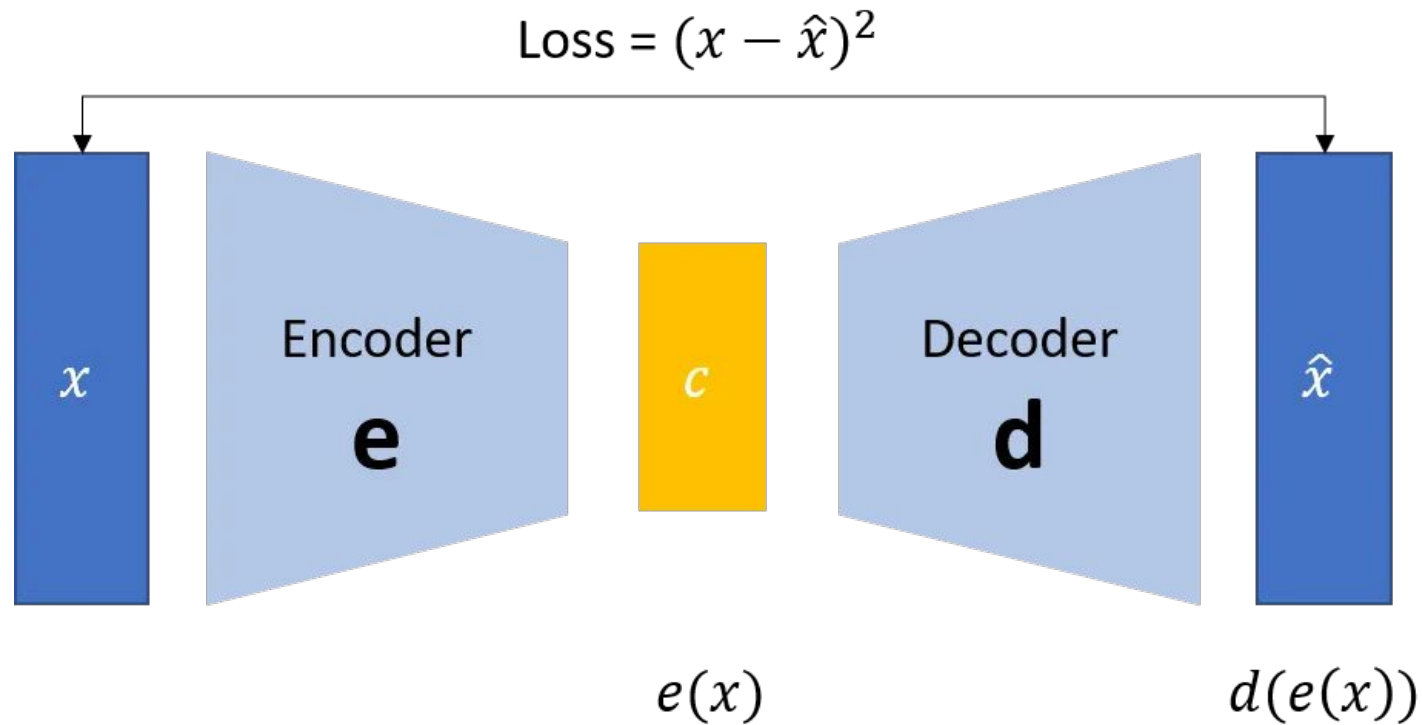
The encoder learns to extract meaningful patterns and compress the data.

Learning Feature Representations in an Autoencoder

4. Latent Space = Feature Representation

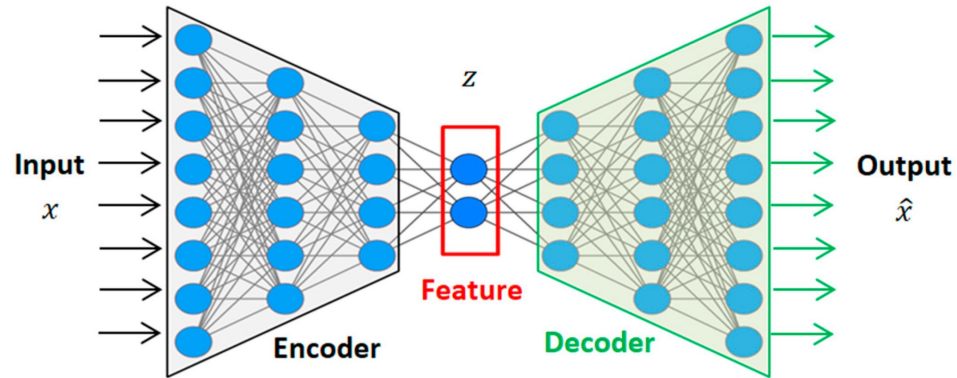
The output of the encoder, the vector z , is the learned feature representation.

Autoencoders

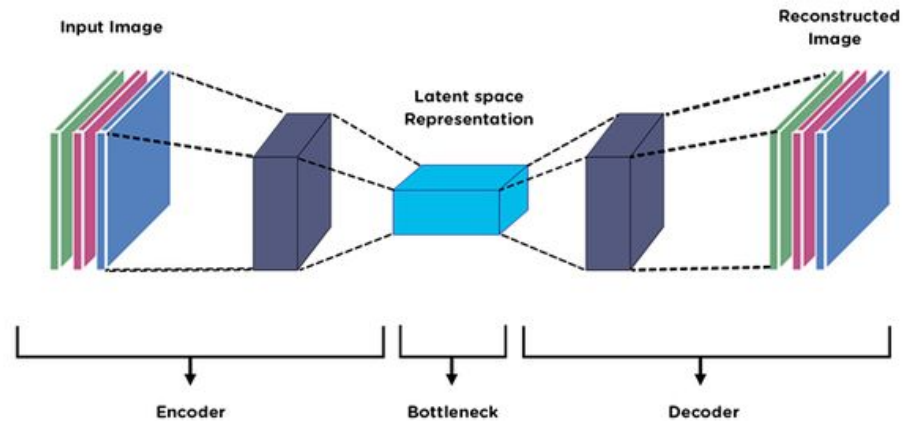


Autoencoders

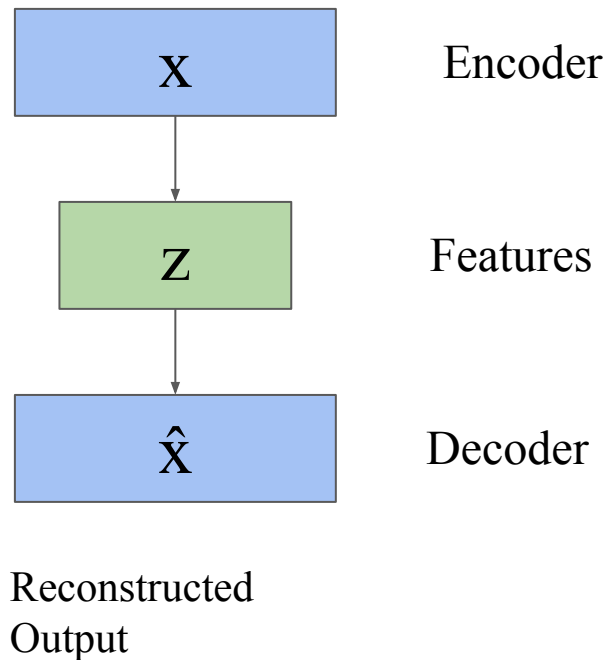
MLP



CNN



Autoencoders



- The model learns to encode only the training data into specific z values.
- If you pick a random z (say, from a normal distribution), the decoder likely has never seen such a z before.



The decoder may produce garbage output, not a valid image because it doesn't know what to do with unfamiliar z values!

Autoencoders

- Denoising
- Dimensionality Reduction
- Pre-training for Deep Learning Models

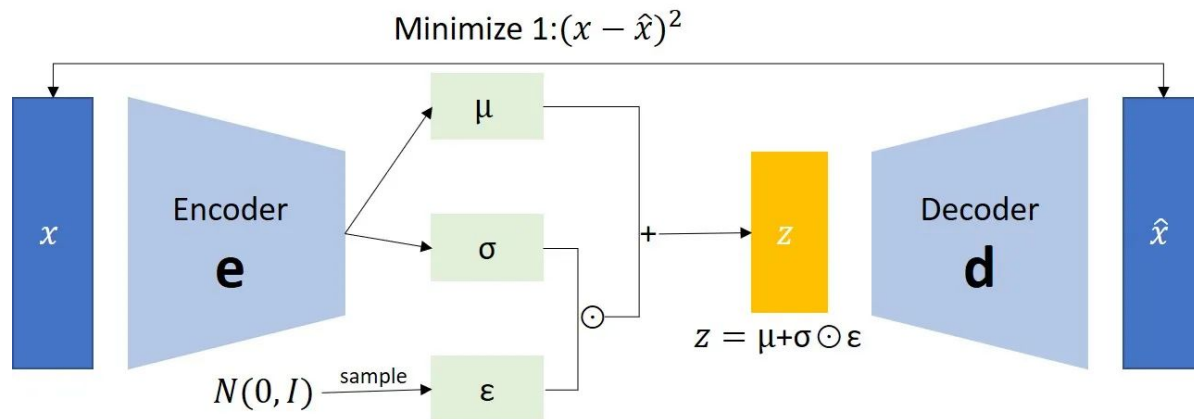
Q: How do we make autoencoder a generative model?

A: Variational Autoencoder

Whereas traditional autoencoders map input data to a single point in latent space, VAEs take a probabilistic approach by mapping data to a distribution across the latent space.

Variational Autoencoders

- A Variational Autoencoder is a generative model that learns a probabilistic mapping between observed data x and a set of latent variables z by maximizing a variational lower bound on the data likelihood.



$$\text{Minimize 2: } \frac{1}{2} \sum_{i=1}^N (\exp(\sigma_i) - (1 + \sigma_i) + \mu_i^2)$$

Defines the approximate
posterior distribution $q(z|x)$

➤ Instead of encoding x into a single value z , outputs of an encoder are,

- μ (mean vector)
- σ (Log variance vector)

➤ It define a Gaussian distribution for each input,

$$q(z|x) = \mathcal{N}(\mu, \sigma^2)$$

➤ Our goal is to sample z from a this distribution. But, if we do direct sampling than it will block backpropagation.

➤ That's why we sample from a standard normal, $\epsilon \sim \mathcal{N}(0, I)$

And then we compute,

$$z = \mu + \sigma \odot \varepsilon$$

Element-wise
multiplication



- **Decoder** takes the sampled z and tries to reconstruct the original input x .

Output is \hat{x} , an approximation of x .

Loss Optimization

1. Reconstruction Loss

$$(x - \hat{x})^2$$

- Measures how well the decoder reconstructs the original input.
- Could be MSE (for images), or cross-entropy (for text/labels).

2. KL Divergence Loss

$$\frac{1}{2} \sum_{i=1}^N (\exp(\sigma_i) - (1 + \sigma_i) + \mu_i^2)$$

- Loss between the learned distribution $q(\mathbf{z}|\mathbf{x})$ and the prior $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$.
- Regularizes the latent space, encouraging the model to keep \mathbf{z} close to a standard normal distribution.

$$\text{Total Loss} \quad \mathcal{L} = \text{Reconstruction Loss} + \text{KL Divergence}$$

Applications: VAE

- **Data Generation**

- To generate realistic data samples.
- By sampling from the latent space, VAEs can create entirely new data points resembling the original data set. This might come in handy during tasks like data augmentation

- **Anomaly detection**

- Can learn the normal distribution of data and identify anomalies or outliers.

Applications: VAE

- **Dimensionality Reduction**

- Compress high-dimensional data into more meaningful latent space representations. This simplifies analysis and visualization.

- **Image and Video Processing**

- Can also enhance and generate visual content. VAEs can remove noise from low-quality images and even create video sequences by predicting future frames based on temporal relationships in data.

Challenges and Future Directions

- **Model complexity**

- Balancing complexity and performance remains a critical challenge for VAEs. Overly complex models risk overfitting, while overly simple models might lack expressiveness.

- **Modal Collapse**

- When a model generates limited diversity in its samples.
- Need advanced regularization techniques and hybrid approaches that combine VAEs with other generative models

Challenges and Future Directions

- **Interpretability**

- Understanding and interpreting the latent space remains a challenge for VAEs.
- Going forward, expect researchers to focus on developing more interpretable latent representations, allowing better insights into the underlying data structures.

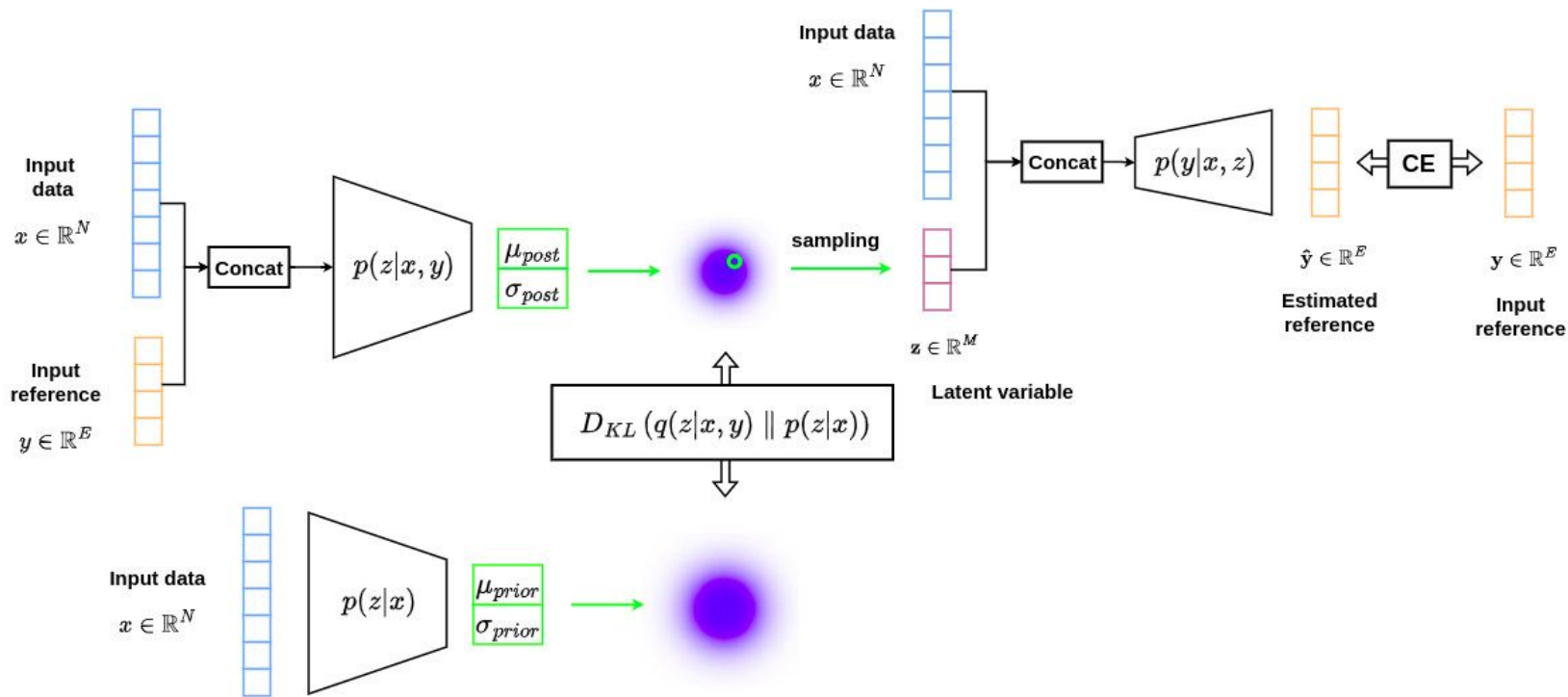
In Autoencoder, the user has no control over the specific outputs generated.

Example: A conventional VAE trained on the previously mentioned MNIST data set will generate new samples of handwritten digits from 0 to 9, but it cannot be constrained to output only 4s and 7s.

Conditional VAE (cVAE)

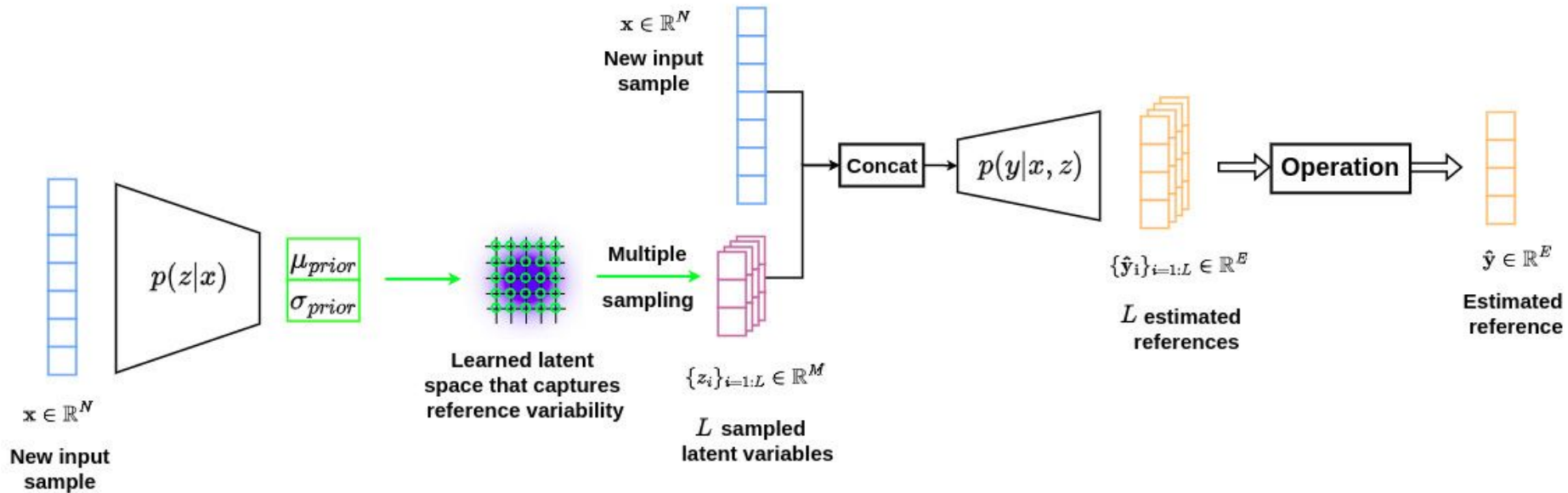
- Enable outputs conditioned by specific inputs, rather than solely generating variations of training data at random.
- Achieved by incorporating elements of supervised learning (or semisupervised learning) alongside the traditionally unsupervised training objectives of conventional autoencoders.

cVAE: Training Phase



The goal of VAEs is to learn an embedding (latent) space that efficiently represents the distribution of the X input in a lower dimensional space for easier interpretation.

cVAE: Inference Phase



During inference, a new sample x is given as input to $p(z|x)$ and several points z_i are sampled in the corresponding latent space to generate a set of plausible outputs y_i that will represent the learned variability of the references for a given x

Practical Applications: cVAE

- Controlled Data Generation
- Image-to-Image Translation
- Text Generation
- Speech Synthesis
- Drug Discovery / Molecule Generation
- Anomaly Detection

Discuss: Most Recent Progress in cVAE.
(i.e. Interpretable CVAE, Multimodal VAEs, etc.)

Diffusion Models

Diffusion models are generative models that “diffuse” training data with random noise, then learn to reverse the diffusion process to output new images.

- Among the neural network architectures at the forefront of generative AI, most notably represented by popular text-to-image models.
- Attracted a lot of attention after OpenAI, Nvidia and Google managed to train large-scale models.

Examples: Stability AI’s Stable Diffusion, OpenAI’s DALL-E, Midjourney, Google’s Imagen.

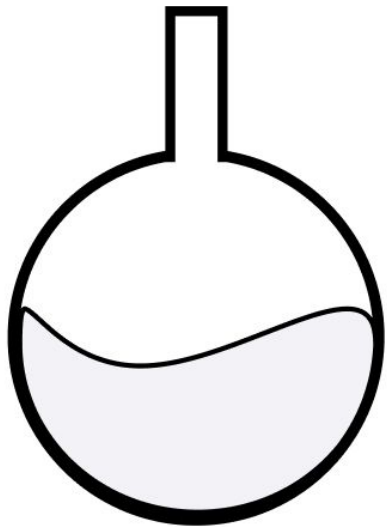
Diffusion Models go beyond just creating high-quality images. They have gained popularity by addressing the well-known challenges associated with adversarial training in GANs.

Diffusion Models offer advantages in terms of training stability, efficiency, scalability, and parallelization.



The Intuition Behind Diffusion Models

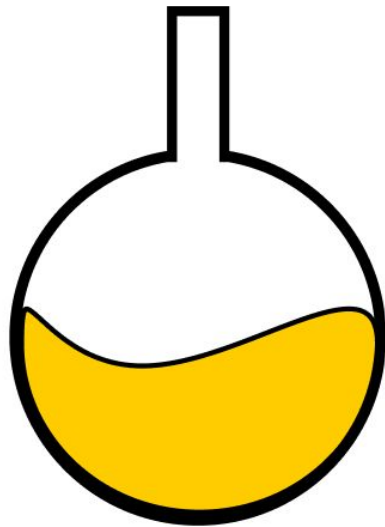
Picture an ink drop dispersing in water. Initially distinct, the drop gradually blends until it's indistinguishable. This physical diffusion process is the cornerstone of understanding diffusion models in AI. Unlike the irreversible nature of ink in water, diffusion models uniquely reverse this process, transforming diffused noise back into clear, structured data.



transparent
water



adding a small
quantity of
another liquid



water is no longer
transparent

Main Component

- At the heart of the reverse engineering lies the neural network, a powerful tool capable of learning and mimicking complex functions.
- For diffusion models, the goal is to teach a neural network to counteract the diffusion process, transforming a noisy, unstructured image into its original, clear form.

Training a Neural Network

1. Adding Noise

Iteratively add gaussian noise to a sprite (Bob the Sprite). Notice, after adding sufficient noise to Bob the Sprite becomes unrecognizable from random noise.



Bob the Sprite!

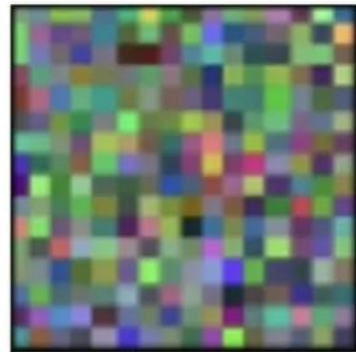


Probably Bob



Well, Bob or Fred...

...

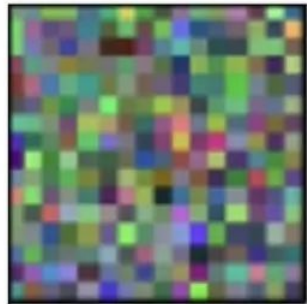


No Idea

Training a Neural Network

2. Predictive Noise

By analyzing the noisy images, it learns to identify and predict the specific noise in each one. Removing this predicted noise effectively reverses the diffusion, bringing us closer to the original image.



No Idea



A person?

...



Looks like Fred



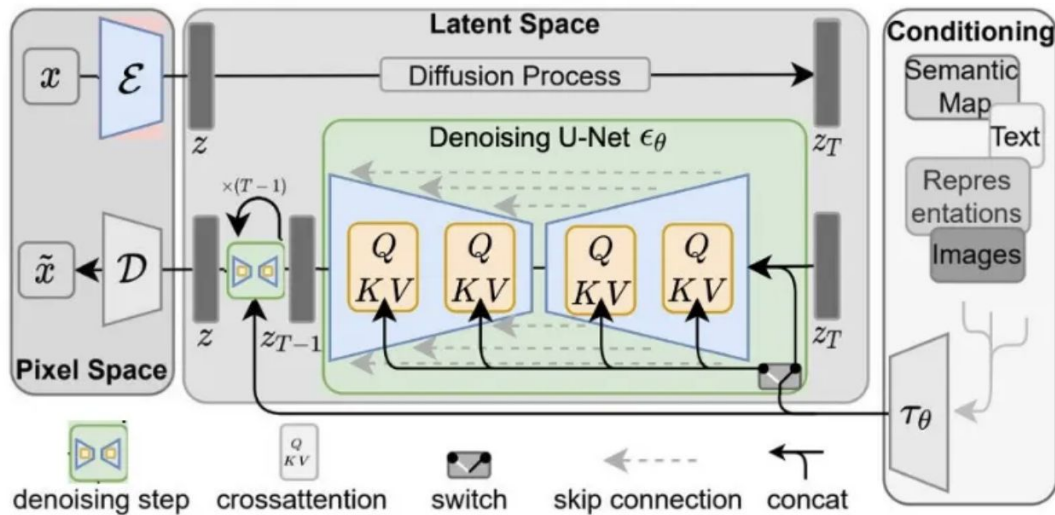
Fred

Importance of Timestep

- Each timestep represents a distinct phase in the image's transformation, where new layers of noise are added.
- The number of timesteps is a key factor, influencing both the model's performance and its computational demands.

Diffusion Model: Architecture (Specifically Latent DM)

Forward Diffusion Processes



Diffusion Model: Training Process

In each batch of the training process, the following steps are taken:

1. Sampling a random timestep t for each training sample within the batch (e.g. images)
2. Adding Gaussian noise by using the closed-form formula, according to their timesteps t
3. Converting the timesteps into embeddings for feeding the U-Net or similar models (or other family of models)
4. Using the images with noise and time embeddings as input for predicting the noise present in the images
5. Comparing the predicted noise to the actual noise for calculating the loss function
6. Updating the Diffusion Model parameters via backpropagation using the loss function

Diffusion Model: Forward Process

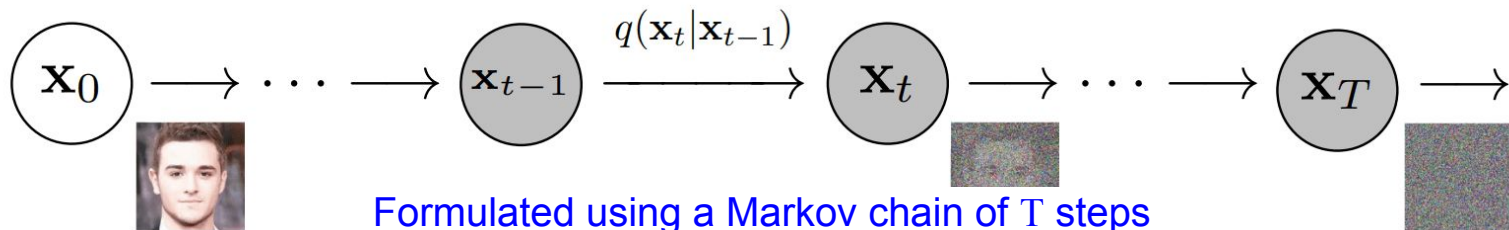
- The forward process involves gradually adding noise to the data over several time steps, effectively destroying the original data to reach pure noise.

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

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I})$$

Noisy data at time t

Noisy variance
schedule



Diffusion Model: Reverse Process

- The reverse process aims to reconstruct the original data by progressive denoising the noisy data, step by step.
- As $T \rightarrow \infty$, the latent \mathbf{x}_T becomes nearly an isotropic Gaussian distribution.
- If we learn the reverse distribution $q(\mathbf{x}_{t-1} | \mathbf{x}_t)$, we can sample \mathbf{x}_T from $N(0, I)$, run the reverse process, and obtain \mathbf{x}_0 a novel data point from the original data distribution.

Diffusion Model: Reverse Process

- By applying the reverse formula for all timesteps ($p_{\theta}(\mathbf{x}_0:\square)$, called the trajectory), we can transform \mathbf{x}_T into a sample from the data distribution.

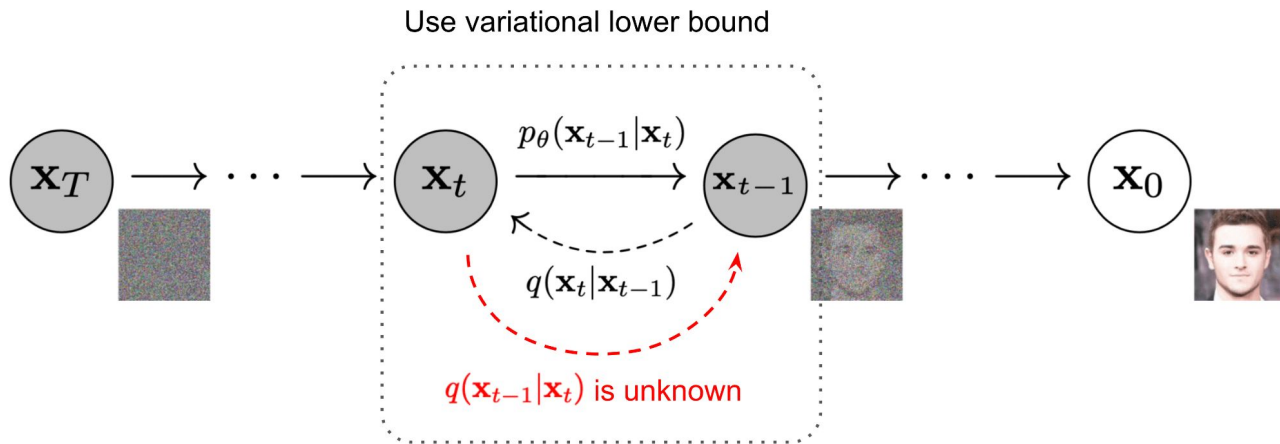
$$p_{\theta}(\mathbf{x}_{0:T}) = p_{\theta}(\mathbf{x}_T) \prod_{t=1}^T p_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

- By conditioning the model on timestep \mathbf{t} , it learns to predict the Gaussian parameters: the mean $\mu_{\theta}(\mathbf{x}_{\square}, \mathbf{t})$ and covariance $\Sigma_{\theta}(\mathbf{x}_{\square}, \mathbf{t})$ for each timestep.

Diffusion Model: Reverse Process

- Since $q(\mathbf{x}_{t-1} | \mathbf{x}_t)$ is also Gaussian, for sufficiently small β_t , we can set p_θ to be Gaussian and parameterize only its mean and variance

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



Diffusion Model: Reverse Process

- By applying the reverse process across all timesteps (the full trajectory), we can transform \mathbf{x}_T into samples from the original data distribution."

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

- Conditioning the model on timestep t allows it to learn the Gaussian parameters mean $\mu_{\theta}(\mathbf{x}_t, t)$ and covariance $\Sigma_{\theta}(\mathbf{x}_t, t)$ for each step.

Diffusion Model: Noise Schedule

- The noise schedule determines how noise variance changes over time in both forward and reverse processes.
- β can be fixed to a constant or chosen as a schedule over the T timesteps. In fact, one can define a variance schedule, which can be linear, quadratic, cosine etc.
- The original DDPM authors utilized a linear schedule increasing from $\beta_1=10^{-4}$ to $\beta_T=0.02$

Diffusion Model: Noise Schedule

Importance of Noise Schedule:

- Controls Difficulty of Learning
- Affects Sample Quality
- Stability in Training

Diffusion Model: Positional Encoding

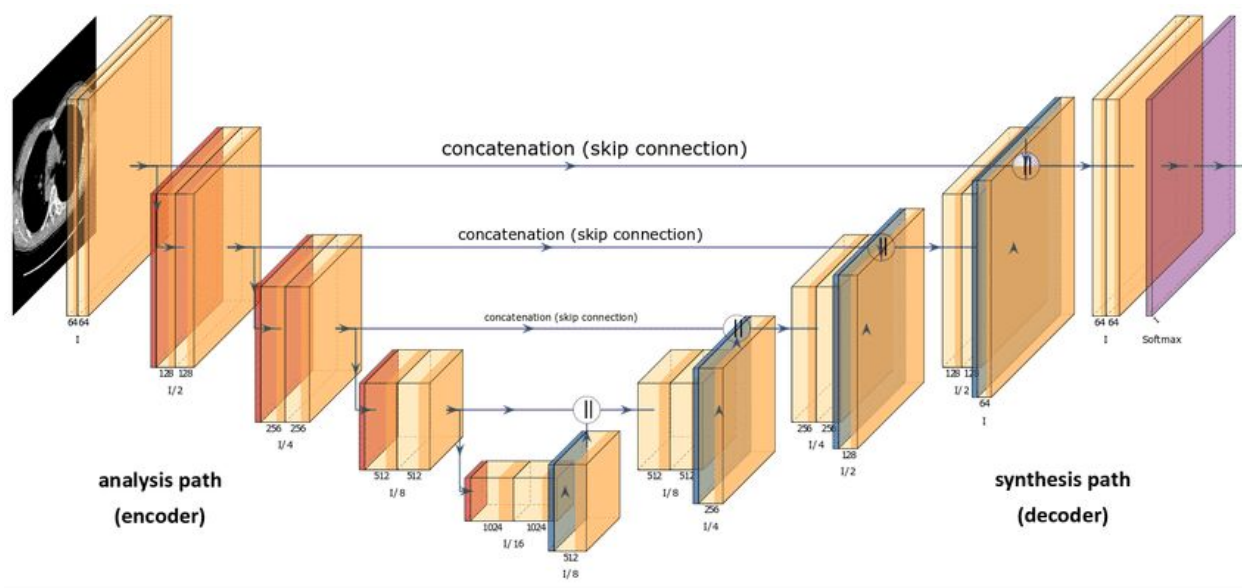
- Positional Encoding injects information about the time step into the model, allowing it to understand the position in the diffusion process.

Why Positional Encoding?

- To Inject Spatial or Sequential Information
- To Encode the Timestep

Diffusion Model: Neural Network Architecture (U-NET)

At the heart of the diffusion model lies the neural network that learns to predict the noise added at each step.



**Contracting Path
(Encoder)**

**Expansive Path
(Decoder)**

Bottleneck

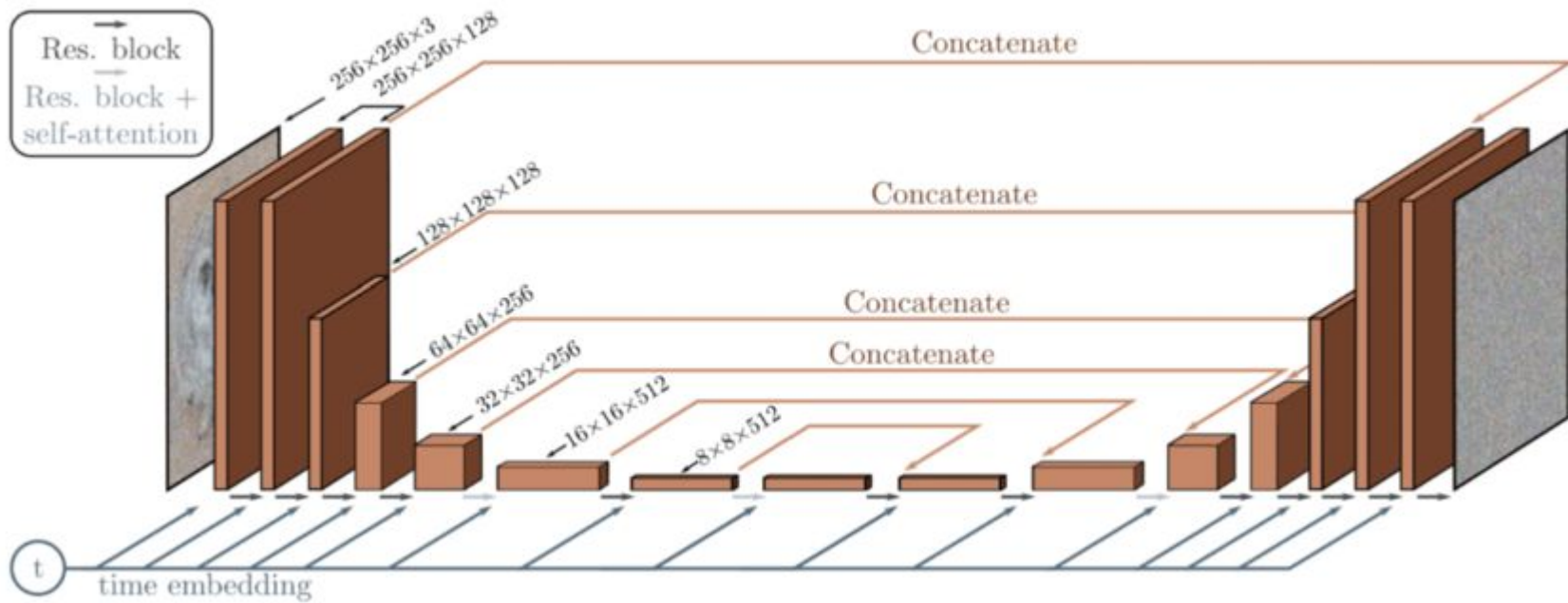
(Most compressed and abstract information is stored)

Contracting Path: Aims to decrease the spatial dimensions of the image, while also capturing relevant information about the image.

Expanding Path: Aims to upsample the feature map and produce a relevant segmentation map using the patterns learnt in the contracting path.

Importance of Skip connections: They help the network recover fine details that would otherwise be lost during the downsampling process.

Diffusion Model Architecture Based on U-Net



Loss Calculation

Objective: To find the reverse Markov transitions that maximize the likelihood of the training data.

$$L_{CE} = -\mathbb{E}_{q(x_0)} \log p_{\theta}(x_0) \leq \mathbb{E}_{q(x_{0:T})} \left[\log \frac{q(x_{1:T} | x_0)}{p_{\theta}(x_{0:T})} \right] = L_{VLB}$$

Jensen's inequality for minimizing the cross entropy as the learning objective


Score-based Diffusion Model (2021)

- Tackle generative learning using score matching and Langevin dynamics.
- **Score-matching** refers to the process of modeling the gradient of the log probability density function, also known as the score function.
- **Langevin dynamics** is an iterative process that can draw samples from a distribution using only its score function.

- A generative model that learns the gradient of the data distribution's log-density:

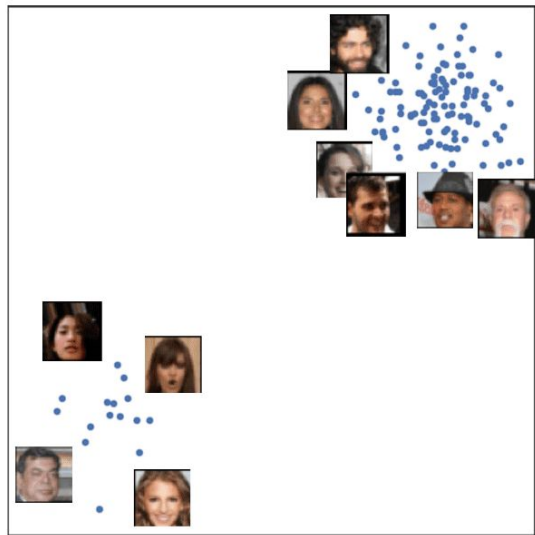
$$s_{\theta}(x_t, t) \approx \nabla_{x_t} \log p_t(x_t) \quad \text{Score function}$$

Neural network (usually
a U-Net or transformer)
estimating the score



(It points towards regions where $p(x)$ is larger (i.e. towards the data manifold).)

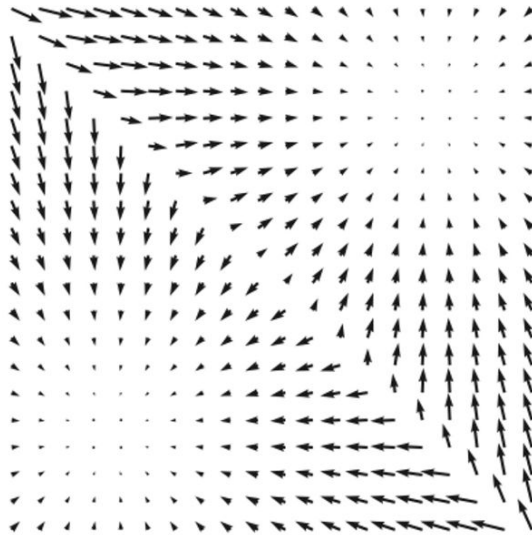
Score-based Diffusion Model



Data samples

$$\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \stackrel{\text{i.i.d.}}{\sim} p(\mathbf{x})$$

score
matching



Scores

$$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$$

Langevin
dynamics



New samples

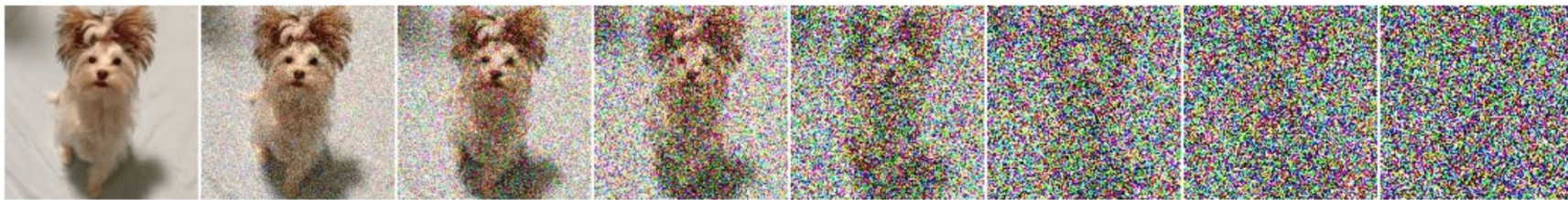
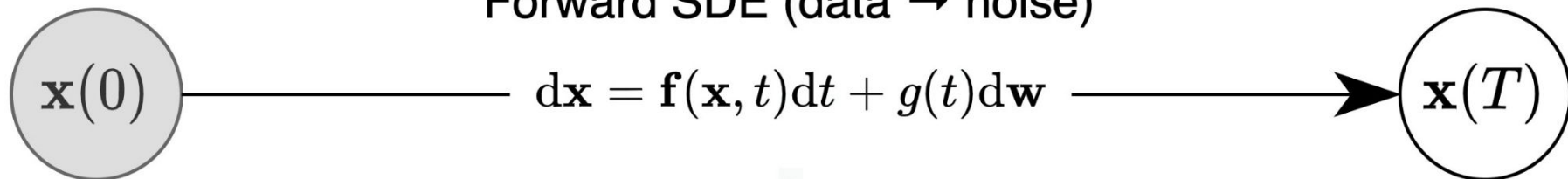
Score-based Diffusion Model

- This is the training process. You show the model (a neural network) your noisy, not-quite-right pictures and teach it to predict what the "art critic's advice" (the score) should be. The goal is for the model's guess ($s_{\theta}(x)$) to become as good as the true, perfect advice ($\nabla_x \log p(x)$).
- **Example:** You show the robot millions of slightly wrong cat drawings along with the correct instructions to fix them. Through practice, the robot learns to give those correction instructions by itself

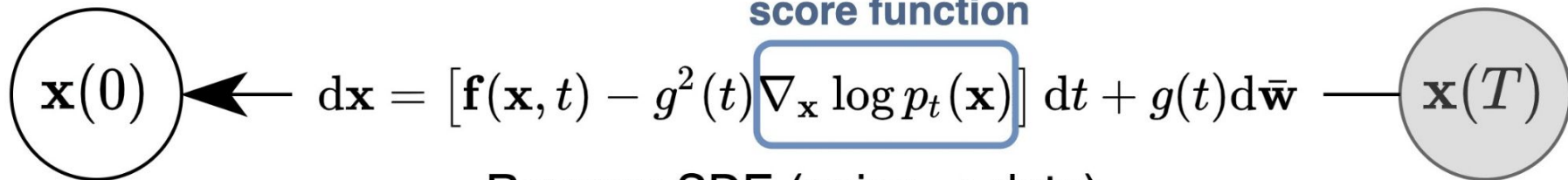
Langevin Dynamics

- This is the sampling process: how the model creates a *new* picture from scratch. It starts with just random noise (like TV static).
- How it works: The model then uses its learned "score" function. It looks at the noise and says, "To make this look more like a cat, I should nudge the pixels in *this* direction." It does this over and over again, making a small correction at each step.
- Example: The robot starts with a blank, staticky canvas. It asks itself, "What tiny change would make this look 0.1% more like a cat?" It makes that change. It asks again and makes another tiny change. It repeats this process hundreds of times. The static slowly transforms into a brand new, coherent cat picture.

Forward SDE (data \rightarrow noise)



score function



Reverse SDE (noise \rightarrow data)

Forward SDE

- $x(0)$: starting point—a perfect, clean image from your dataset (e.g., a cat photo).
- $dx = \dots$: This equation is a recipe for how to change the image (x) by a tiny, tiny amount.
- $f(x,t) dt$: This part is the deterministic drift. It's a planned, predictable push. In many models, this is set up to simply pull the image towards zero (make it fainter).
- $g(t) dw$: This is the random diffusion. This is the crucial part that adds a tiny bit of random noise (dw) at each step. The $g(t)$ part controls how *much* noise to add at a given time t .
- $x(T)$: The end result after following this recipe for a long time (T)—complete, pure randomness (static).

Reverse SDE

- $x(T)$: Your starting point—pure noise (static).
- The New $dx = \dots$ Recipe: This looks complex but has a clear purpose.
- $f(x,t) dt$: This is the same "drift" term from the forward process, but now it's pulling in the reverse direction.
- $-g^2(t) \nabla_x \log p_\square(x) dt$: This is the magic! This is the "score" term. It acts as a corrective force. At every tiny step, the model uses the score (the "instructions") to nudge the random noise *away* from chaos and *toward* something that looks like a clean cat image. The $-$ sign means it's reversing the flow of the forward process.
- $+g(t) dw$: This is the same random noise term. Interestingly, a little randomness is needed in the reverse process to generate diverse and high-quality samples.
- $x(0)$: The final result—a brand new, clean image that never existed before.

Case Study:

Design a system that can automatically detect student engagement and reactions from classroom video data while a professor is teaching. The system should be able to differentiate between states like attentive, distracted, confused, bored, engaged in discussion, and taking notes. Please, take prior permission of video recording from respective faculty.

Permissions and Ethics:

- Draft a one-page consent form (purpose, data collected, storage duration, sharing policy, right to withdraw).
- Faculty's approval
- Provide non-participating seating options or face-blurring for opted-out students.

Case Study:

Deliverable:

1. Processed video(s) with bounding boxes and IDs.
2. Demonstration ready
3. Report (max 3pages) covering:
 - Approach used.
 - Models chosen and justification.
 - Engagement formula.
 - Example outputs.
 - Ethical considerations.

Stable Diffusion Model (Rombach et al., 2022)

- Based on a particular type of diffusion model called Latent Diffusion model, proposed in High-Resolution Image Synthesis with Latent Diffusion Models and created by the researchers and engineers from CompVis, LMU and RunwayML.
- The model was initially trained on 512x512 images from a subset of the LAION-5B database.
- This is particularly achieved by encoding text inputs into latent vectors using pretrained language models like CLIP.



An image generated with Stable Diffusion 3.5 based on the text prompt a photograph of an astronaut riding a horse.

WATERMARK_STRICT annotation guide

Classic
watermark



WATERMARK

Advertisement **and**
watermark



WATERMARK

Brand text covering
Image, watermark
looks



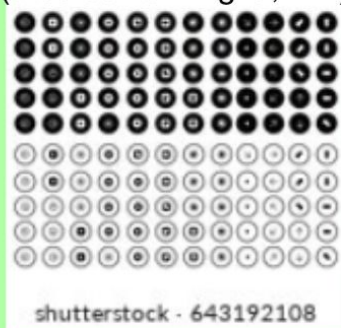
WATERMARK

No trademark/brand,
but "100%" watermark
characteristics



WATERMARK

Text not covering image
(also containing ®, TM)



NO WATERMARK

Subtle text not covering
main parts of the image



NO WATERMARK

Logo



NO WATERMARK

Advertisement

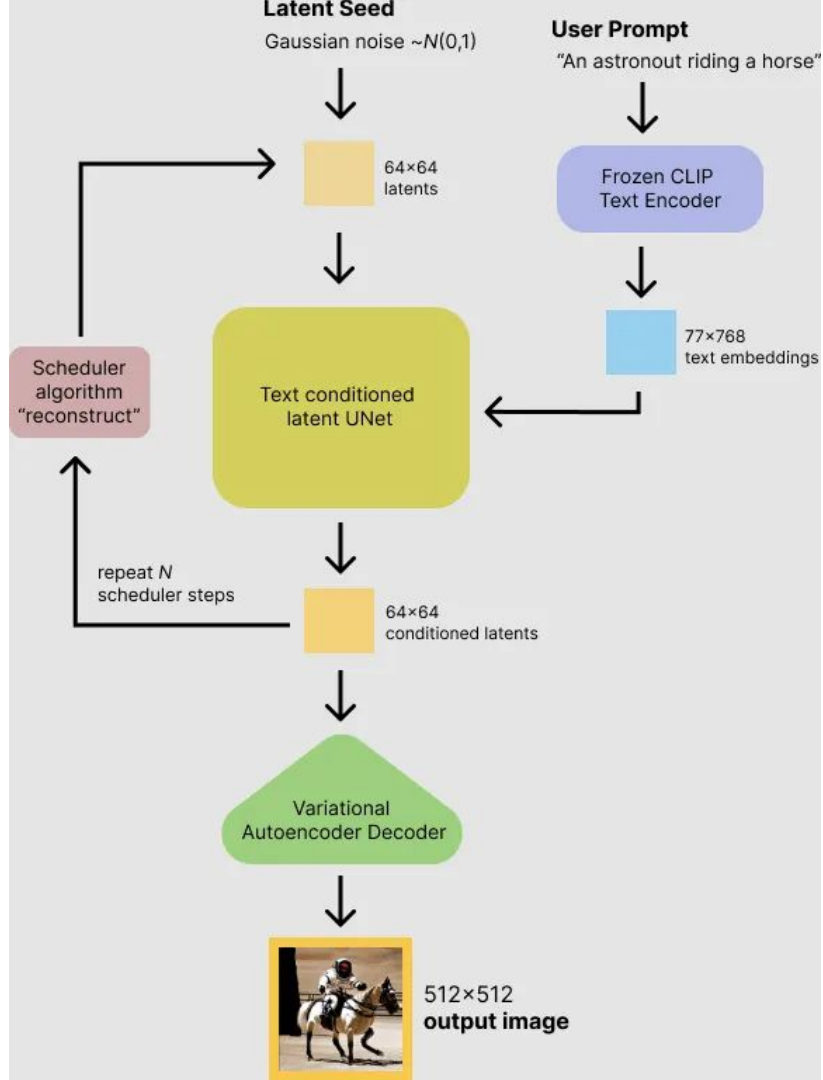


NO WATERMARK

Stable Diffusion Model: Training

- Stable Diffusion is a large text to image diffusion model trained on billions of images
- Image diffusion model learn to denoise images to generate output images. Stable Diffusion uses latent images encoded from training data as input.
- Further, given an image z_0 , the diffusion algorithm progressively add noise to the image and produces a noisy image z_t , with t being how many times noise is added. When t is large enough, the image approximates pure noise.
- Given a set of inputs such as time step t , text prompt, image diffusion algorithms learn a network to predict the noise added to the noisy image z_t .

Stable Diffusion Inference Process



Comparison: VAEs vs Diffusion Models

Aspect	CVAEs (Conditional VAEs)	Diffusion Models
Core Idea	Conditional generation using latent space with labels or attributes	Gradual denoising of random noise to generate data
Training	Reconstruction + KL divergence with conditioning	Noise prediction at each diffusion step
Speed	Fast, single decoding step	Slow, many denoising steps (faster with improvements)
Output Quality	Often blurry, limited detail	High-quality, sharp, diverse samples
Applications	Controlled generation, representation learning	State-of-the-art image, video, text-to-image generation