Lecture 17: Diffusion Models - I

COMP 5801H/4900A: Generative AI and LLMs

2026-03-10

Sriram Subramanian
Assistant Professor & Canada Research Chair, Carleton University
Faculty Affiliate, Vector Institute for Artificial Intelligence
Faculty Affiliate, Schwartz Reisman Institute for Technology and Society

Carleton University

VECTOR INSTITUTE

SCHWARTZ REISMAN INSTITUTE FOR TECHNOLOGY AND SOCIETY

Outline

- The Intuition & Foundations
- The Forward Process
- The Reverse Process & Training
- Sampling & Acceleration
- Conditioning & Guidance
- Latent Diffusion & Advanced Topics

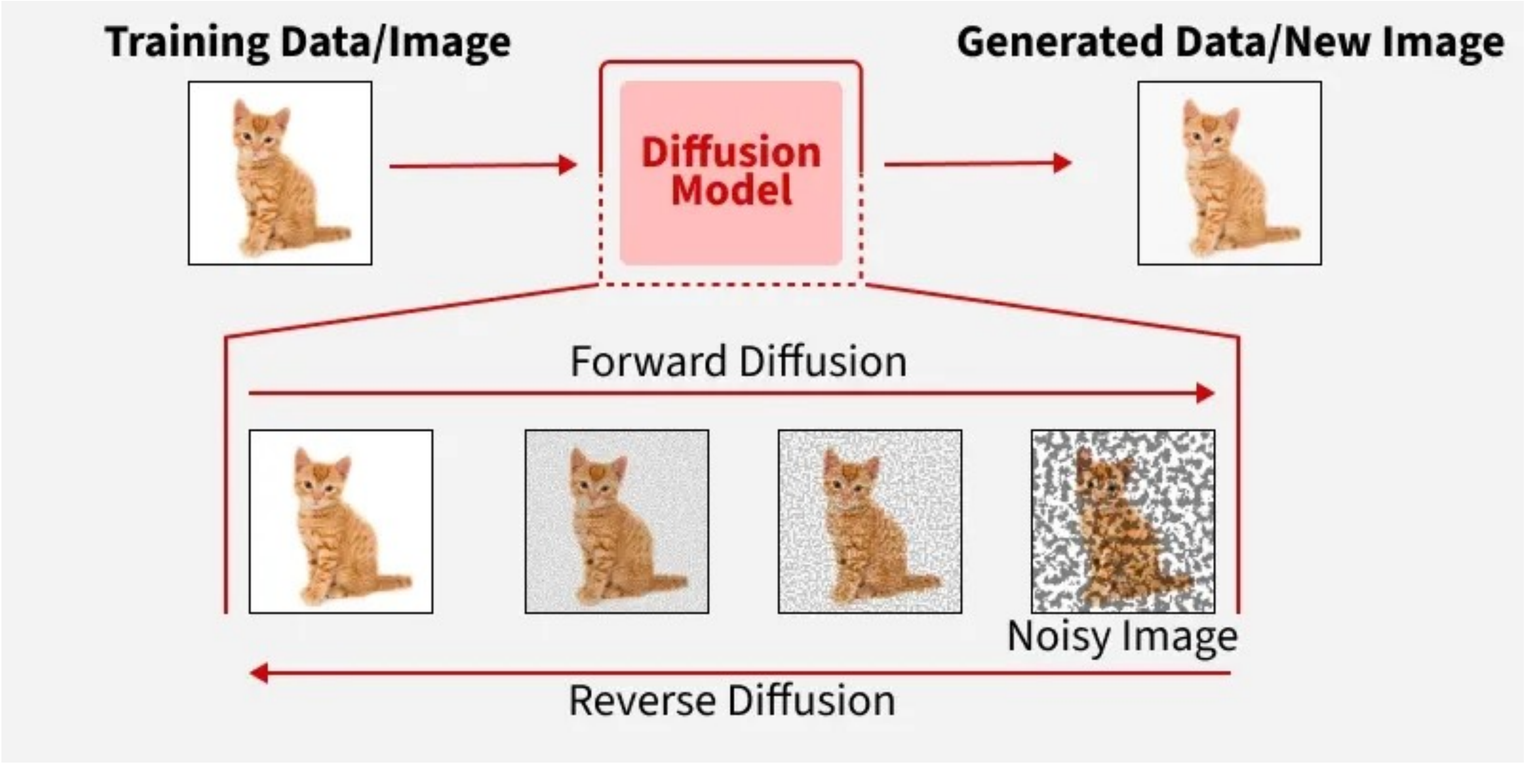
An Introduction to Diffusion Models

- **The New Frontier**: How we moved from the "Adversarial Games" of GANs to the "Generative Physics" of Diffusion.
- **Core Mechanism**: Learning to create by learning to reverse destruction.
- **Impact**: The technology powering DALL-E 3, Midjourney, and Stable Diffusion.

The Diffusion Intuition

- **The Concept**: In physics, Diffusion is a process where a concentrated substance (like a drop of blue ink) spreads out into a medium (like a glass of water) until it becomes a uniform, featureless mixture.
- **The Forward Direction: Destruction**
  - Imagine a high-definition photo of a cat.
  - Every "step" of diffusion adds a tiny bit of random Gaussian noise.
  - Eventually, the structure of the cat is lost, and we are left with a "glass of grey water" (Pure Static).
  - **Crucial Point**: This process is easy and fixed. We don't need a brain to destroy an image; we just need a random number generator.
- **The Reverse Direction: Creation**
  - Now, imagine trying to "un-mix" that ink from the water.
  - To a human or a simple algorithm, this is impossible. You can't tell where a specific molecule of ink came from once it's dispersed.
  - **The AI's Job**: We train a Neural Network to look at a noisy image and say, "*I think the noise that was added in the last step looked like THIS.*"
  - By subtracting that predicted noise, we move one tiny step back toward the clear image.

Diffusion Models



Credit: https://www.geeksforgeeks.org/artificial-intelligence/what-are-diffusion-models/

Why this works for “Imagination"?

- Because the starting point is random noise, the model has to "imagine" a structure to fit that noise.
  - If the noise looks slightly like a cloud, the model will nudge it toward being a cloud.
  - If the noise looks slightly like a face, it nudges it toward a face.
  - **Result**: Every time you start with a different "seed" of noise, you get a completely different, unique creation.

Why Diffusion?

- **Mathematical Stability**
  - **GANs** are a "moving target." Since the Discriminator and Generator are constantly changing, the model can easily diverge or fail.
  - **Diffusion** training is a simple **supervised regression task**. We tell the model: "*Here is a noisy image, and here is the exact noise I added. Predict it.*" This makes the training objective stable and predictable.
- **Superior Distribution Coverage (No Mode Collapse)**
  - GANs often get "stuck" on a few safe, easy-to-draw images (Mode Collapse).
  - Because Diffusion models are Likelihood-based, they are mathematically incentivized to learn every single mode in the training data. This leads to much higher diversity in results.
- **Scalability & High Resolution**
  - Diffusion models handle high-resolution textures better because they process the image iteratively. Instead of trying to generate 1 million pixels perfectly in one "shot," they refine those pixels over 50–100 steps.
  - This "step-by-step" approach allows for incredible compositional control (e.g., following a complex text prompt like "A neon cyberpunk city with a rainy street and a cat in a tuxedo").

GANs vs Diffusion

Feature	GANs	Diffusion
Training Stability	Very Low (Fickle)	**Very High (Stable)**
Diversity	Poor (Mode Collapse)	**Excellent**
Inference Speed	**Instant (1 step)**	Slow (Multiple steps)
Prompt Adherence	Average	Superior

Core Philosophy

- **The Big Idea**: Diffusion models don't learn to "draw" in the traditional sense. Instead, they learn the structure of noise. If a model can perfectly identify what is "noise" in an image, it implicitly understands what the "signal" (the object) is.
- **The Two-Phase Workflow** - The entire philosophy rests on two distinct phases:
  - **The Diffusion (Forward) Phase**: We take a sample from our data distribution and systematically destroy it. This is a **parameter-free** process; we aren't "training" anything here. We are just following a mathematical recipe to turn a dog into static.
  - **The Reverse (Generative) Phase**: This is where the **Neural Network** lives. We show the network the noisy versions and challenge it to predict how to remove that noise.
- **The Role of the "Score" Function**
  - In physics-based terms, the model is learning the **Score Function**—a vector field that points toward higher density (where the "real" data lives).
  - If you are standing in a field of random pixels, the model provides a "map" that says: "*To look more like a face, move these pixels 2 units to the left.*"

Core Philosophy

- **Why "Small Steps" Matter?**
  - Why not just remove all the noise at once?
  - Trying to go from pure noise to a cat in one step is a nearly impossible mapping (this is what GANs try to do).
  - By breaking it into **1,000 tiny steps**, the model only has to make a very small, very easy decision at each stage. This "divide and conquer" strategy is why the results are so incredibly detailed.
- **Summary of the Philosophy**
  - **Forward**: $Data \rightarrow Noise$ (Easy, fixed)
  - **Reverse**: $Noise \rightarrow Data$ (Learned, iterative)
  - **Goal**: To learn a function that can "reverse entropy" within the image domain.

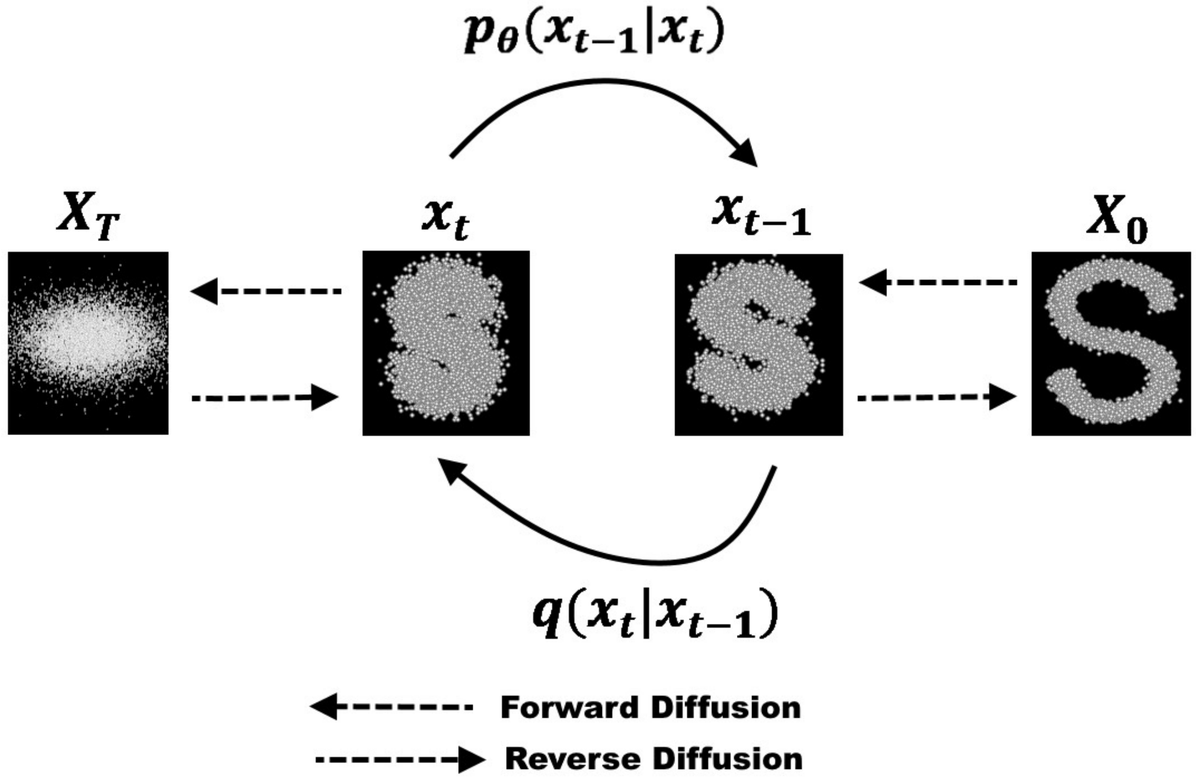
The Forward Process (Destructive)

- **The Objective**: We want to systematically transform our data x_0 into pure Gaussian noise x_T through a sequence of intermediate latent variables $x_1, x_2, \ldots, x_T$.
- **The Step-by-Step Transition:**
  - The forward process adds a small amount of Gaussian noise at each step t. This is defined by a Markov Chain:

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

  - β_t: This is the Variance Schedule. It is a very small value (e.g., from 10^{-4} to 0.02).
  - The Mean ($\sqrt{1-\beta_t}x_{t-1}$): Notice that we slightly "shrink" the previous image before adding noise. This ensures the variance doesn't grow to infinity.
  - The Variance ($\beta_t \mathbf{I}$): The amount of pure white noise added.
- **Why Gaussian Noise?**
  - We use Gaussian noise because of the **Central Limit Theorem** and its convenient mathematical properties. Specifically:
    - It is easy to sample.
    - The sum of two Gaussians is also a Gaussian, which allows us to "jump" steps.

Diffusion Process



Credit: https://towardsdatascience.com/diffusion-models-made-easy-8414298ce4da/

The Markov Chain

- **The Concept**: The forward process is a **Markov Chain**, meaning each state x_t depends only on the state immediately before it (x_{t-1})
- **The Visual Progression**
  - As we move through the chain, we observe a gradual loss of **mutual information** between the current latent and the original image:
    - x_0: The clean data point (e.g., a high-res photo).
    - $x_1 \ldots x_{T/2}$: Structure remains visible, but "salt and pepper" noise begins to blur the edges and textures.
    - $x_{T/2} \ldots x_T$: Semantic content (eyes, shapes, colours) disappears. Only the "ghost" of the global structure remains before it, too, vanishes.

The Markov Chain

- **Why "Markovian"?**
  - By defining the process as $q(x_t | x_{t-1})$, we simplify the joint distribution of the entire sequence. The probability of the whole path is simply the product of every individual step:

$$q(x_{1:T} | x_0) = \prod_{t=1}^{T} q(x_t | x_{t-1})$$

- **Key Takeaway: The "Diffusion" Analogy**
  - In physics, particles move randomly. After a billion tiny random movements, their starting location is irrelevant.
  - In AI, the "particles" are pixel values. After T tiny Gaussian perturbations, the "location" (the original image) is completely forgotten.

The Noise Schedule (β_t)

- **The Variable**: The "Noise Schedule" β_t determines how much noise we add at each step. If we add too much too fast, the model loses the signal early. If we add too little, we waste computation.
- **The Linear Schedule (The DDPM Classic)**
  - Used in the original DDPM (Ho et al. 2020) paper, β_t increases linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$.
  - The Issue: Researchers noticed that with a linear schedule, images become pure noise too quickly. By step 500 of 1000, the image is often already "gone," making the second half of training redundant.

The Noise Schedule (β_t)

- **The Cosine Schedule (The Modern Standard)**
  - Introduced in *Improved Denoising Diffusion Proobabilistic Models* (Nichol & Dhariwal, 2021), this schedule follows a cos^2 curve.
  - The Benefit: It adds noise much more gradually in the middle of the process. This preserves the structure of the image for longer, ensuring the model learns meaningful details at every step of the 1,000-step chain.
- **Why the Schedule Matters**
  - **Learning Efficiency**: A good schedule ensures that every step t presents a "solvable" challenge for the model
  - **Information Decay**:
    - **Linear**: Drops information rapidly; x_{500} is often already featureless.
    - **Cosine**: Drops information smoothly; x_{500} still retains "ghostly" silhouettes of the original data.

The Reparameterization Trick

- **The Problem**: In a Markov Chain, to get to step $t = 500$, you theoretically have to sample 500 times in a row. This is computationally expensive and makes training impossible.
- **The Solution**: Because the noise we add is always Gaussian, we can use the **Reparameterization Trick** to collapse the entire chain into a single equation. We can express any noisy version of the image x_t as a linear combination of the original image x_0 and a single lump sum of noise.

Math of the trick

- Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ (the cumulative product of all alphas up to step t).
- The formula for x_t becomes:

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

- where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
- $\sqrt{\bar{\alpha}_t}$: The "Signal" weight. As t increases, this value drops toward 0.
- $\sqrt{1 - \bar{\alpha}_t}$: The "Noise" weight. As t increases, this value climbs toward 1.

Why this is a "Cheat Code" for Training

- This formula is the reason Diffusion Models **actually work**. During training:
  - We pick a random image x_0 from our dataset.
  - We pick a random time step t (e.g., $t = 342$).
  - We pick a random noise vector ϵ.
  - We use the Reparameterization formula to instantly create x_t.
  - The Challenge: We ask the Neural Network to look at x_t and guess what the noise ϵ was.
- **Physical Interpretation**
  - At $t = 0$, $\bar{\alpha}_t = 1$, so $x_t = x_0$ (100% image).
  - At $t = T$, $\bar{\alpha}_t \approx 0$, so $x_t \approx \epsilon$ (100% noise).

The Endpoint (x_T)

- The Destination: After T steps (usually 1,000), the forward diffusion process has successfully "erased" the original data. We have transitioned from a structured image to a state of **Isotropic Gaussian Noise**.
- What is "Isotropic" Gaussian Noise?
  - Isotropic means "uniform in all directions."
  - Mathematically, this means $x_T \sim \mathcal{N}(0, \mathbf{I})$.
- The covariance matrix is the **Identity Matrix** ($\mathbf{I}$), meaning every pixel is now completely independent of its neighbours. There are no longer any correlations, edges, or shape, just pure, uncorrelated "snow."

The Endpoint (x_T)

- **Why is this endpoint critical?**
  - This endpoint is the "bridge" between training and generation:
  - **During Training**: We ensure the noise schedule β_t is aggressive enough so that by step T, the image is truly destroyed.
  - **During Inference (Generation)**: This is our Starting Point. Because x_T is a simple Gaussian distribution, we can easily sample a random block of static from our computer's memory and tell the model: "*Start here.*"
- **The Mathematical Guarantee**
  - By choosing our variance schedule such that $\bar{\alpha}_T \approx 0$ (the cumulative product of all $1 - \beta_t$), our formula becomes:

$$x_T \approx \underbrace{\sqrt{0} \cdot x_0}_{\text{Original image gone}} + \underbrace{\sqrt{1} \cdot \epsilon}_{\text{Pure noise remains}}$$

The Reverse Process (Generative)

- **The Objective**: If we can reverse the forward process, we can generate data from scratch. We want to find a way to sample from $p(x_{t-1} \mid x_t)$, allowing us to move from pure noise back to the data manifold.
- **The Challenge: The Untractable Reverse**
  - In the forward process, $q(x_t \mid x_{t-1})$ is a simple Gaussian. However, the true reverse $q(x_{t-1} \mid x_t)$ depends on the **entire data distribution**, which we don't know.
  - To "undo" a step of noise, you need to know what a "real" image looks like.
  - **The Solution**: We train a Neural Network (p_θ) to **approximate** this reverse transition.

The Iterative Refinement

- Unlike GANs, which generate an image in a single pass, the Reverse Process is a slow, iterative "clean-up" job:
  - **Start**: Sample $x_T \sim \mathcal{N}(0, \mathbf{I})$ (Random Static).
  - **Step**: Use the model to predict the noise in x_T and subtract it to get x_{T-1}.
  - **Repeat**: Repeat this T times (usually 1,000) until you reach x_0.
- **What is the model actually predicting?**
  - While the math says we are predicting the "Mean" of the previous step, in practice, we train the model to predict the N**oise Residual** (ϵ_θ).
  - Input: A noisy image (x_t) and the current time step (t).
  - Output: A map of the noise that needs to be removed.
  - Logic: "*If I see this pattern in the static at step 500, it's likely part of the noise that shouldn't be there.*"

The Generative Equation

- To get the next (cleaner) step, we use:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z$$

- Where z is a small amount of random noise added back in to keep the process stochastic.
- Derivation depends on **Bayesian Inference and Gaussian properties**. See Ho et al. (2020) “Denoising Diffusion Probabilistic Models” for details
- σ_t defines the "width" of the Gaussian distribution for the reverse step. In the original DDPM paper, the authors explored two main choices for this value:
  - Option A: $\sigma_t^2 = \beta_t$ (Comes from the variance added in the forward step).
  - Option B: $\sigma_t^2 = \tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t$ (Comes from the variance of the posterior $q(x_{t-1} \mid x_t, x_0)$).
- Both options work well in practice, though modern implementations often treat this as a learnable parameter or a fixed schedule to balance image sharpness and diversity.

The Objective Function

- **The Goal:** We want to maximize the likelihood of our model generating real data, $p_\theta(x_0)$. However, directly calculating the probability of a specific image is mathematically "intractable" (impossible to compute directly). Instead, we optimize a proxy called the Variational Lower Bound (VLB), also known as ELBO.
- **The Loss Decomposition:**
  - The total loss for a Diffusion model is derived from the KL Divergence between the forward process q and the reverse process p_θ. It can be broken down into three main terms:

$$L_{VLB} = L_0 + L_1 + \ldots + L_T$$

    - L_T (**The Prior**): Ensures the final step x_T is close to pure Gaussian noise. (Usually constant, as the forward process is fixed).
    - L_t (**The Denoising Task**): It compares the true reverse posterior $q(x_{t-1} \mid x_t, x_0)$ with our model's prediction $p_\theta(x_{t-1} \mid x_t)$.
    - L_0 (**The Reconstruction**): The final step from $x_1 \rightarrow x_0$, ensuring the final pixels are sharp and accurate.

Why Use KL Divergence?

- We use **Kullback-Leibler (KL) Divergence** to measure the "distance" between two probability distributions.
  - **Forward**: We know where the image came from (q).
  - **Reverse**: The model is guessing where it came from (p_θ).
  - **The Mission**: Minimize the distance between these two so that the model's "guess" becomes indistinguishable from the "truth."
- **From Complexity to Simplicity**
  - The full ELBO math is heavy, but because our distributions are Gaussian, the KL Divergence simplifies into a comparison of the **Means**.
  - Instead of optimizing the whole distribution, we just need the model's predicted mean μ_θ to match the posterior mean $\tilde{\mu}_t$.

The Simplified Loss Function

- **The Breakthrough**: While the **ELBO** is mathematically rigorous, it is notoriously difficult to optimize. In the **DDPM** paper, researchers discovered that you can ignore the complex weighting terms in the ELBO and use a "simplified" version that actually produces better image quality.
- **The "Simple" Equation**
  - We don't need to predict the distribution mean or the KL Divergence directly. We simply ask the U-Net to predict the noise ϵ and compare it to the ground truth noise we added:

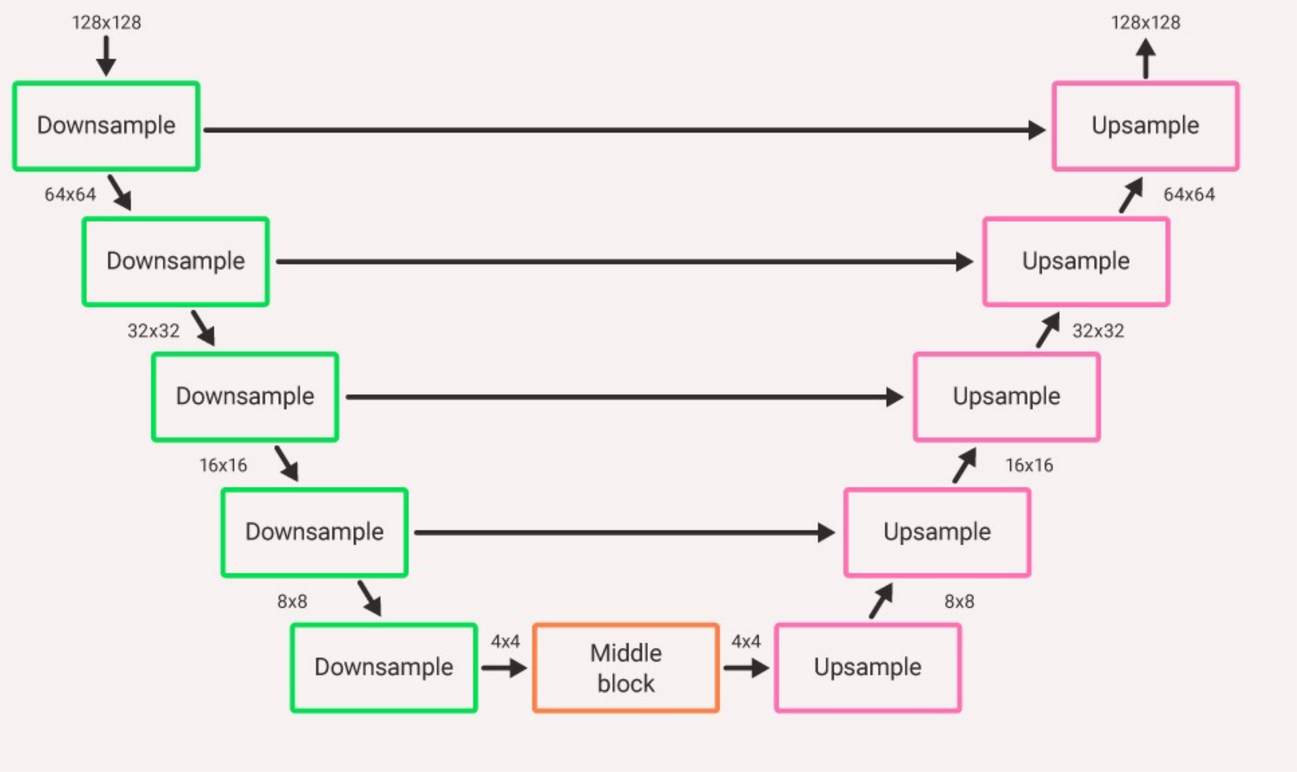
$$\mathscr{L}_{simple}(\theta) = \mathbb{E}_{x_0,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

    - ϵ: The true Gaussian noise we sampled and added to the image.
    - $\epsilon_\theta(x_t, t)$: The U-Net's guess of what that noise looks like.
    - $\| \dots \|^2$: Standard L2 Loss (Mean Squared Error).

Why "Simple" is Better than "Rigorous"

- The full ELBO math puts a lot of weight on the very early steps of the reverse process (near x_T). However, those steps are mostly about global blobs of colour.
  - **The Problem**: The "Rigorous" loss focuses too much on the static and not enough on the fine details.
  - **The Solution**: $\mathscr{L}_{simple}$ effectively "re-weights" the steps, forcing the model to pay more attention to the middle and late stages of denoising where the actual "art" (textures and edges) is created.
- **The Training Loop (Summary)**
  - **Sample** an image x_0 from the dataset.
  - **Pick** a random timestep $t \in [1,T]$.
  - **Generate** random noise ϵ.
  - **Create** x_t using the Jump Formula.
  - **Train** the network to minimize the difference between its guess and the real ϵ.

U-Net



Credit: https://eviltux.com/2024/08/11/training-a-u-net-model-from-scratch/

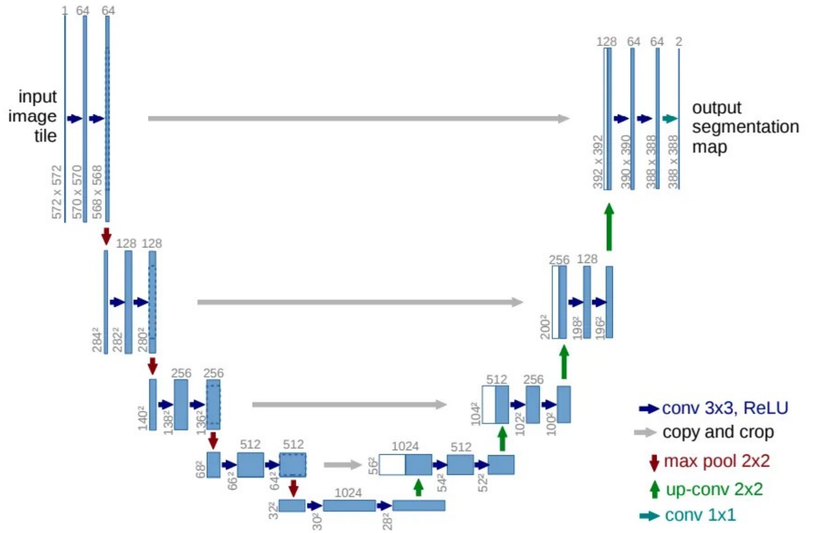


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows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

Credit: Ho et al. “Denoising Diffusion Probabilistic Models”

U-Net

- **Structural Design**
  - The U-Net consists of a symmetrical Encoder-Decoder structure with three critical components:
    - **The Downsampler** (Encoder): Successively reduces the spatial resolution of the image while increasing the "feature depth." This allows the model to understand the global context (e.g., "This noise is shaped like a mountain").
    - **The Bottleneck**: The lowest resolution point where the most abstract, high-level features are processed.
    - **The Upsampler** (Decoder): Successively restores the spatial resolution, allowing the model to focus on local details (e.g., "This specific pixel should be slightly darker to form a shadow").
- **The Secret Sauce: Skip Connections**

Need for Skip connections

- Skip connections take the high-resolution feature maps from the Encoder and concatenate (attach) them directly to the corresponding layers in the Decoder.
  - **In the Encoder**: The feature map is sharp and spatially accurate, but lacks "deep" understanding.
  - **In the Decoder**: The feature map has traveled through the bottleneck; it has "deep" semantic understanding (it knows it's drawing a cat), but it's spatially blurry.
  - **The Merge**: By stitching the two together, the Decoder gets the best of both worlds. It uses the "deep" features to know what to draw and the "skip" features as a spatial template to know exactly where to draw the edges.

Why not simple CNN/Transformers?

Architecture	Flaw in Diffusion
Standard CNN	Usually maintains one resolution; struggles with global structure.
Pure Transformer	Computationally expensive for high-res pixels; ignores spatial local correlations.
U-Net	Maintains spatial precision via skip connections while understanding global context.

Time Embeddings

- **The Problem**: A U-Net is just a pile of weights. If you show it a noisy image, it doesn't know if that image is "slightly dusty" ($t = 10$) or "total chaos" ($t = 950$). Since the strategy for fixing a few pixels is very different from the strategy for building a world out of static, the model needs to know "what time it is."
- **Transforming a Number into a Vector**
- We don't just feed the number "500" into the network. Neural networks struggle with raw integers. Instead, we use **Sinusoidal Positional Embeddings** (the same trick used in Transformers/LLMs).
  - We map the scalar t to a high-dimensional vector using sine and cosine functions of different frequencies.
  - **Why?** This allows the model to "feel" the distance between timesteps. Step 500 and 501 look very similar in this vector space, while step 5 and 500 look very different.

Injection: Where does the time go?

- The time embedding isn't just added at the beginning; it is **injected into every single block** of the U-Net (both the Encoder and the Decoder).
  - The scalar t is turned into a vector.
  - It passes through a small MLP (Multi-Layer Perceptron) to "resize" it.
  - It is **added** to the feature maps after each convolutional layer.
- **The Dynamic Nature of the U-Net**
  - Because of these embeddings, the U-Net effectively becomes **1,000 different models in one**.
  - **At High t**: The time embedding "activates" the parts of the brain that deal with global shapes and rough compositions.
  - **At Low t:** The embedding shifts the model's focus toward high-frequency details, textures, and "polishing" the image.

Sampling (Inference)

- **The Objective**: Now that the model is trained, we move from the **Forward Process** (Data → Noise) to the **Generative Process** (Noise → Data). We start with a blank canvas of static and "carve out" an image.
- **Step 0: The Random Seed**
  - We begin by sampling a starting point x_T from a pure Gaussian distribution:

$$x_T \sim \mathcal{N}(0, \mathbf{I})$$

  - This is why every generated image is unique; different "white noise" leads to different final pixels.

Sampling (Inference)

- **The Iterative Loop (1,000 to 0)**
  - For each timestep t from T down to 1, the model performs a three-step dance:
    - **Predict**: The U-Net looks at the current noisy image x_t and the timestamp t to predict the noise ϵ_θ.
    - **Subtract**: We use our generative equation to remove a portion of that predicted noise.
    - **Refine**: We add back a tiny bit of random noise $\sigma_t z$ to keep the generation "fluid" and prevent the model from collapsing into a blurry average.
- **Why so many steps?**
  - Unlike GANs (which generate in 1 step), Diffusion is a conservative process.
  - One step would try to jump too far and create "artifacts" (glitches).
  - Many small steps allow the model to correct its mistakes. If it places a pixel slightly wrong at step 800, it can fix the alignment at step 799.

The Problem of Speed

- **The Reality Check**: While the mathematical "Reverse Process" we derived is beautiful, it has a massive practical flaw: **Inference Speed**. To generate a single high-quality image using the standard DDPM method, the U-Net must run **1,000 sequential passes**.
- **The Computational Cost**:
  - **Sequential Nature**: Each step $t-1$ depends on the output of step t. You cannot "parallelize" these steps across multiple GPUs.
  - **Latency**: Even on a high-end H100 GPU, running a large U-Net 1,000 times can take 15 to 30 seconds for a single image.
  - **The User Experience Gap**: In a world of instant gratification (like Google Search or TikTok), a 30-second wait is an eternity for a consumer app.

The Problem of Speed

- **Where is the waste?**
  - Researchers realized that the "trajectory" from noise to data is actually quite smooth.
  - **Redundancy**: If you know the direction the pixels are moving at step 500, it's highly likely they are moving in a very similar direction at step 499.
  - **The "Short-Circuit" Idea**: If we can take larger "leaps" down the manifold without falling off the path, we can reduce 1,000 steps to 20 or 50.
- **The Solution: Advanced Schedulers (DDIM)**
  - The most famous solution is DDIM (Denoising Diffusion Implicit Models).
  - **How it works**: It makes the reverse process deterministic. By removing the random noise ($\sigma_t z$) at each step, the math allows us to skip steps (e.g., only calculate every 20th step) while still following the same "flow" toward the final image.
  - **The Result:** 1,000 steps → 25 steps, with almost no loss in visual quality.

The Problem of Speed

- **Impact on the Ecosystem**
  - Solving the speed problem moved Diffusion from a laboratory curiosity to a global phenomenon:
    - **Real-time generation**: Tools like "SDXL Turbo" can now generate images in single steps (< 0.1 seconds).
    - **Mobile Apps**: Efficient sampling allows these models to run on iPhones and laptops rather than requiring a server farm.

DDIM (Denoising Diffusion Implicit Models)

- **The Innovation:** Standard DDPM is Markovian, meaning each step x_{t-1} must be calculated directly from the step immediately before it (x_t). DDIM (Song et al., 2020) changed the math to allow us to skip steps, jumping from x_t to x_{t-k} without losing our way.
- **Non-Markovian Logic**
  - In Diffusion models, we assumed the chain was a sequence of tiny Gaussian steps. DDIM proves that many different reverse processes can result in the same marginal distribution $q(x_t \mid x_0)$.
  - **The "Implicit" Part**: By making the process non-Markovian, we can define a reverse path where x_{t-1} depends on both x_t and the original x_0.
  - **The Result**: We are no longer "chained" to the step-by-step crawl. We can treat the diffusion path as a continuous flow.

DDIM (Denoising Diffusion Implicit Models)

- **The Deterministic Shortcut**
  - DDIM introduces a parameter σ that controls the "stochasticity" (randomness) of the reverse process:
    - If $\sigma > 0$: It behaves like the original DDPM (noisy and slow).
    - If $\sigma = 0$: The process becomes deterministic.
- When the process is deterministic, the path from noise to image is fixed. This allows the model to take **massive leaps**. Instead of 1,000 tiny steps, we can sample at $t = \{1000, 950, 900, \ldots, 0\}$.
- Because DDIM is deterministic ($\sigma = 0$), it enables **Image Inversion**. You can take a real photo, "reverse" it into noise, and then bring it back. This is the mathematical foundation for:
  - **Image Editing**: Changing only the hair colour of a person while keeping the face the same.
  - **Consistency**: Generating the same character in different poses.

Discrete vs. Continuous

- **The Concept**: Up until now, we have treated Diffusion as a sequence of T **discrete steps** $(1,2,\ldots,1000)$. However, if we imagine the number of steps T approaching infinity, the time steps become infinitely small. The process transforms from a chain into a **continuous flow**.
- **The SDE Framework**
  - When we treat time as continuous ($t \in [0,1]$), the forward and reverse processes are described by **Stochastic Differential Equations (SDEs)**:
    - Forward SDE: Gradually injects noise until the data distribution is destroyed.

$$dx = f(x,t)dt + g(t)dw$$

    - Reverse SDE: The "miracle" of diffusion math is that for every forward SDE, there is a corresponding reverse-time SDE that can recreate the data.

$$dx = [f(x,t) - g(t)^2 \nabla_x \log p_t(x)]dt + g(t)d\bar{w}$$

Discrete vs. Continuous

- **Score-Based Modelling**
  - In this continuous world, we don't think about "predicting noise" (ϵ); we think about the **Score Function**: $\nabla_x \log p_t(x)$.
  - **Definition**: The score function is a vector field that points toward regions of higher data density (where the "real" image features live).
  - **The Intuition**: Imagine you are lost in a foggy forest. The "Score" is like a compass that always points toward the path. Even if you are in total noise, the score tells you exactly which way to move to find a recognizable shape.
- **Why the Continuous View Matters**
  - Switching from discrete steps to SDEs (pioneered by **Yang Song et al., 2021**) unlocked several "superpowers":
    - **Black-Box ODE Solvers**: We can use advanced math solvers (like Runge-Kutta) to navigate the reverse process, making generation even faster and more stable than DDIM.
    - **Probability Flow ODE**: We can turn the stochastic (random) SDE into a deterministic ODE. This allows for exact likelihood computation—meaning we can calculate the exact "probability" of an image existing.
    - **Unified Theory**: It proved that DDPM, DDIM, and Score-Matching are all just different ways of looking at the same underlying continuous physics.

Probability Flow ODEs

- **The Concept**: While SDEs involve a "random walk" with constant jitter, Yang Song et al. (2021) proved that for every SDE, there exists a **corresponding Ordinary Differential Equation (ODE)** that shares the same marginal probability densities. This is the **Probability Flow ODE**.

- **Eliminating the Randomness**

  - In a Stochastic Differential Equation (SDE), the movement of pixels is unpredictable because of the noise term (dw). In a Probability Flow ODE, we replace the noise with a purely deterministic direction:

$$\frac{dx}{dt} = f(x,t) - \frac{1}{2} g(t)^2 \nabla_x \log p_t(x)$$

  - The Result: If you start with a specific point of noise x_T, you will **always** end up at the exact same image x_0. There is no "jitter" or randomness during the descent.

The "Unique" Benefits of ODEs

- By turning Diffusion into a deterministic ODE, we unlock capabilities that are impossible with random walks:
  - **Exact Inversion (The "Reverse" Button)**: You can take a real image, encode it into a specific noise latent, and then decode it back **perfectly**. This is the secret to high-fidelity **image editing** and **style transfer**.
  - **Manipulating Latent Space**: Because the paths are smooth and deterministic, we can perform "vector math" on noise. We can interpolate between two different noise samples to see a smooth transition between two different generated images.
  - **Speed via ODE Solvers**: We can use century-old mathematical tools like the **Runge-Kutta** or **Euler** methods to solve the path in as few as **10 to 20 steps** without the "drift" associated with random sampling.