Lecture 9: Variational Autoencoders - I
COMP 5801H/4900A: Generative AI and LLMs

2026-02-03

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

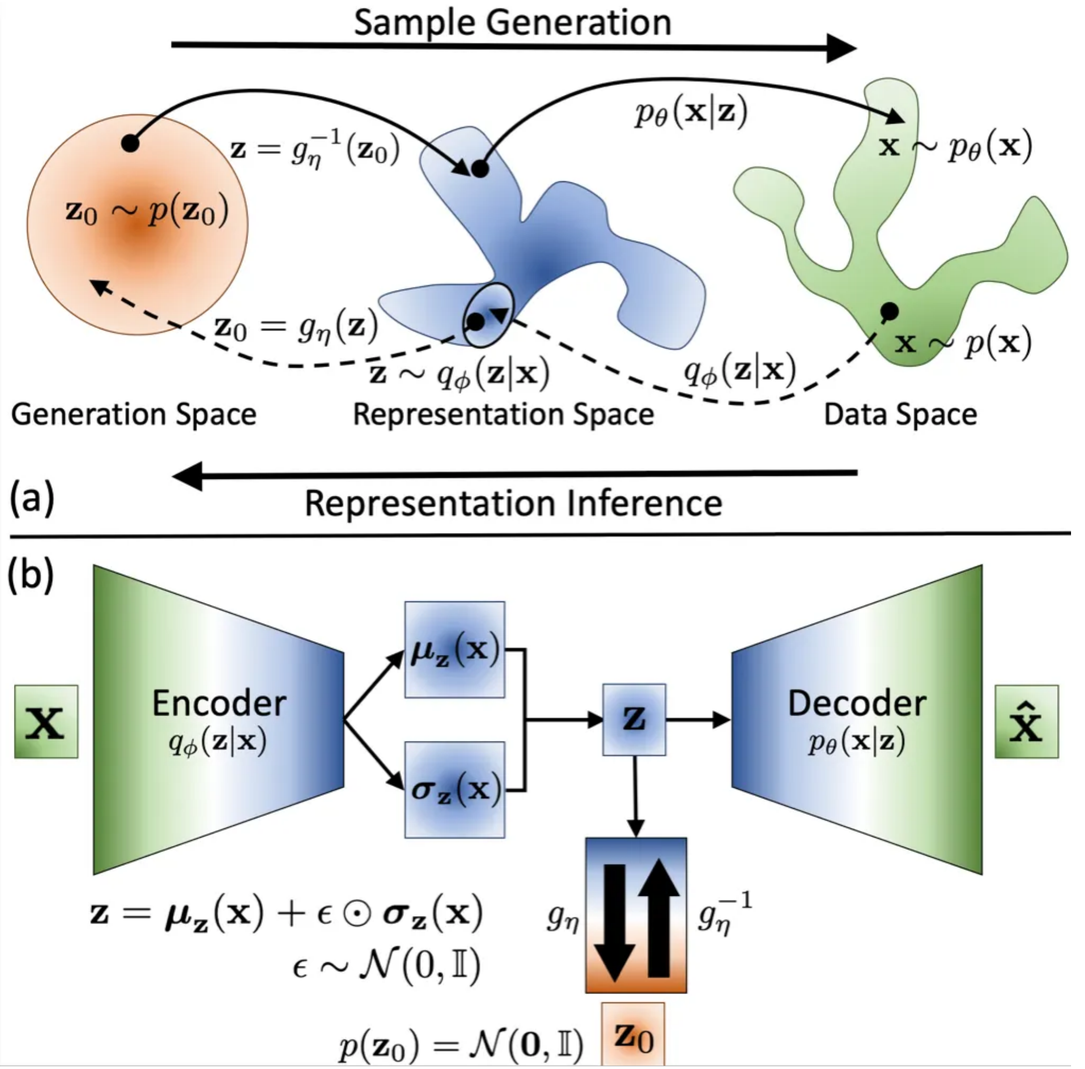
Lecture Outline

- Overview
- Vanilla Autoencoders
- Limitations of Vanilla Autoencoders
- The Latent Variable Framework
- VAE Loss function

Title & Overview

- **What is a VAE?** A generative model that learns to map high-dimensional data (images, sound, text) into a continuous, structured, and probabilistic **Latent Space**
- **The Core Mechanism:** A dual-network system (Encoder/Decoder) that doesn't just "copy" data, but learns the underlying statistical "DNA" of the dataset
- **Lecture Goal:** To move from **Sequence Modelling** (where we predict the next word) to **Representation Learning** (where we manipulate the fundamental features of data)

VAE



Credit: https://medium.com/@kapardhikannekanti/deep-generative-modeling-the-foundation-of-modern-generative-ai-b5a8c051cd73

Why VAEs Matter in the Age of Transformers

- **Representation vs. Prediction**:
  - Transformers like GPT are “Autoregressive", i.e., they predict the next token based on the past
  - VAEs are Holistic. They condense an entire object (like a human face) into a single vector
  - This allows us to perform "Latent Arithmetic" (e.g., adding a "smile" vector to a "neutral face" vector), which is much harder to do with pure text-based Transformers
- **Interpretability & Disentanglement**
  - In a BERT model, it's hard to find the "part" of the model that handles "sarcasm"
  - VAE, we can often find the specific neuron
- **Anomaly Detection**
  - VAEs learn the "Normal Distribution" of data, they are incredibly effective at spotting things that don't belong
  - More useful for specialized medical and industrial tasks

Recap & Comparison

Feature	BERT	GPT	VAE
Model Type	Encoder-Only	Decoder-Only	Encoder-Decoder
Primary Goal	Understanding: Contextual embeddings for NLU	Generation: Sequential, autoregressive completion	Representation: Learning a smooth latent manifold
Direction	Bidirectional (sees past and future)	Unidirectional (sees only the past)	Holistic (compresses the whole input)
Analogy	The Scholar: Reads a sentence 5 times to understand the nuance	The Storyteller: Writes one word at a time based on what came before	The Artist: Learns the "essence" of a face to draw infinite variations

The Generative Problem

- The fundamental goal of generative modelling is to learn the **true data distribution** $P_{data}(x)$
- If we can model this distribution, we can sample from it to **create new, realistic data points** that look like they belong to our original dataset
- **Problem: The Challenge of High Dimensionality**
  - Real-world data (like a 256×256 colour image) exists in a space of massive dimensionality (e.g., 196,608 dimensions)
  - Most of this space is empty "white noise”
  - Realistic data lives on a very thin, curved manifold within that massive space. Simple statistics cannot find it

The "Likelihood" Approach

- We want to find a model with parameters θ that **maximizes the probability** of our observed data:

$$P(x) = \int P(x \mid z) P(z) dz$$

- $P(z)$ (The Prior): Our assumption about how "latent" traits (like lighting or shape) are distributed. We usually choose a simple Gaussian
- $P(x \mid z)$ (The Decoder): The probability of generating data x given a specific latent trait z
- **The Problem**: In practice, this integral is intractable. We cannot check every possible value of z to see how likely it is to produce x

Discrete vs. Continuous Generation

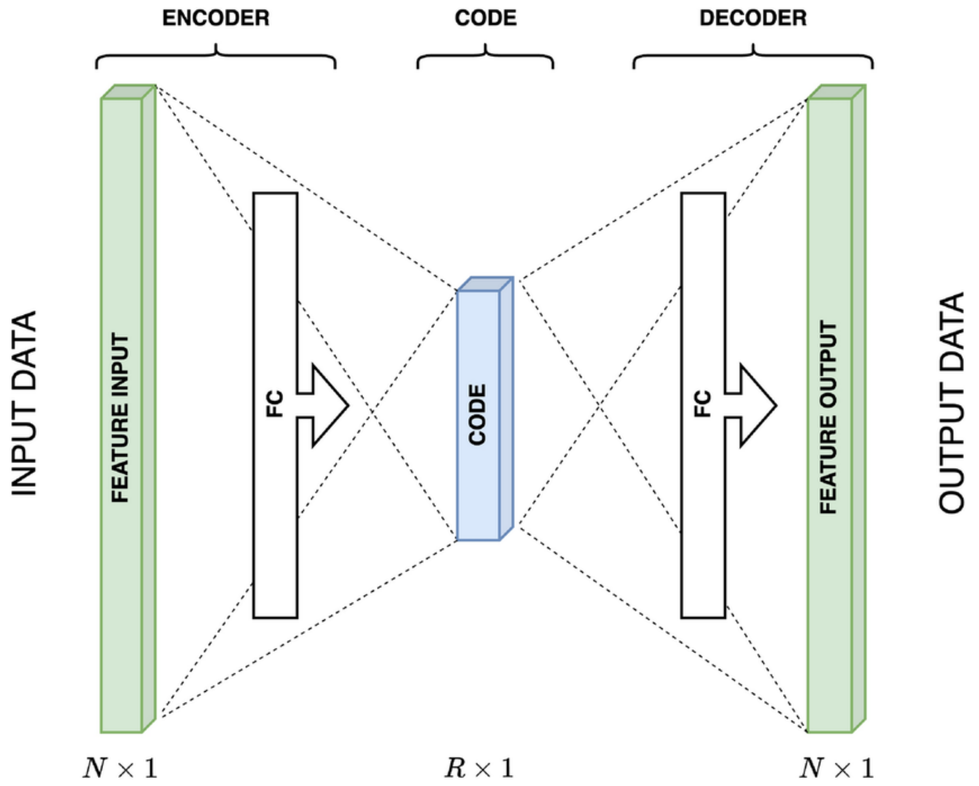
- **The GPT way (Discrete)**: GPT treats generation as a chain of classification tasks. It predicts a discrete "token" from a vocabulary
- **The VAE way (Continuous)**: VAEs treat generation as a regression task in a continuous space
  - VAE is not interested in "What is the next word?"
  - VAE asks: "What are the continuous coordinates in 'Idea Space' that represent this object?"

Requirements for a Solution

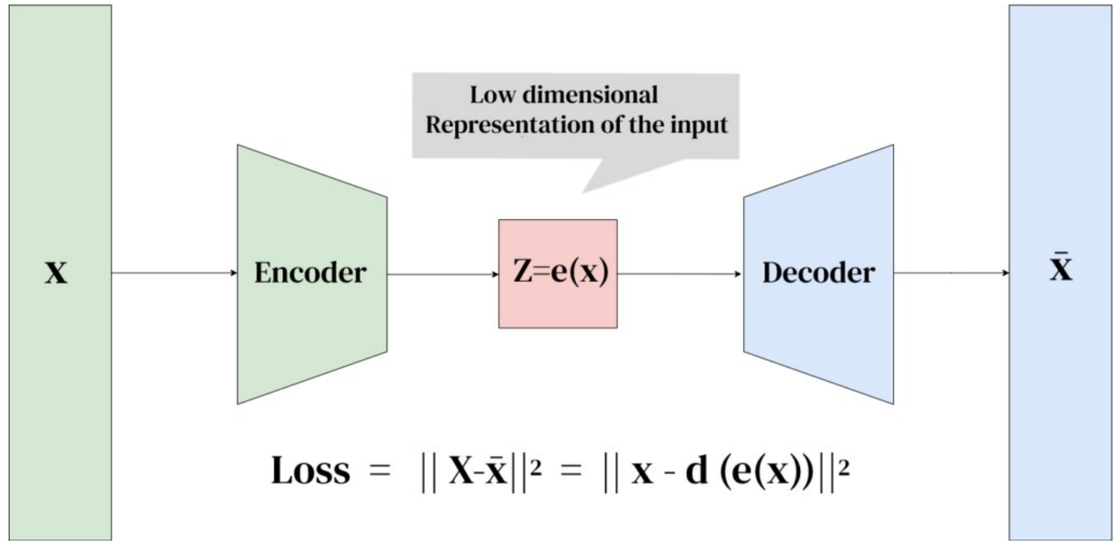
To model $P(x)$ effectively, our model needs:

- **Efficiency**: A way to approximate the integral without checking every z
- **Smoothness**: If two points in latent space are close, the generated images should look similar
- **Completeness**: We should be able to pick any point in the distribution and get a valid result

Vanilla Autoencoders (AE)



Credit: Andric et al. (2024) Deep learning assisted XRF spectra classification



Credit: https://mlarchive.com/deep-learning/variational-autoencoders-a-vanilla-implementation/

Vanilla Autoencoders (AE)

- A Vanilla Autoencoder is a neural network designed to learn a compressed, distributed representation of a dataset, typically for the purpose of dimensionality reduction or denoising
- The Three Core Components
  - **The Encoder (f_θ)**: A series of layers that gradually "squeeze" the input data into a smaller and smaller shape. It maps high-dimensional input x to a low-dimensional Latent Vector z
  - **The Bottleneck**: The narrowest point of the network. This layer contains the Code (latent representation). Because it has fewer dimensions than the input, the model is forced to discard "noise" and keep only the most salient features
  - **The Decoder (g_ϕ)**: A "mirror image" of the encoder that attempts to reconstruct the original input from the compressed code. It maps z back to a reconstruction $\hat{x}$

The Training Objective: Reconstruction Loss

- The model is self-supervised; the "target" is the input itself. We want the output $\hat{x}$ to be as close to the original x as possible
- **Formula**: $L(x, \hat{x}) = \|x - \hat{x}\|^2$ (typically Mean Squared Error)
- **The Logic**: If the model can rebuild the image of a cat from just 16 numbers in the bottleneck, then those 16 numbers must capture the "essence" of being a cat

Key Characteristics

- **Deterministic:** A specific input x will always map to the exact same point z in latent space
- **Lossy Compression:** Autoencoders are “lossy”, some fine detail is always lost during the squeeze
- **Feature Learning:** It automatically discovers features (like edges or textures) without needing human-labeled tags

Why "Squeeze" the Data?

- **Discovering the "Latent Manifold"**
  - The "meaningful" information (the curve of a '2' or the line of a '1') can actually be described by just a few variables like rotation, thickness, and scale
  - The code layer captures these Latent Variables. It learns the "Rules of the World" rather than just the "Values of the Pixels”
- **Feature Extraction (Automatic Engineering)**
  - An Autoencoder automatically discovers these features
  - The first layers of the encoder might learn "edges," the middle layers learn "loops," and the bottleneck learns "digits"
  - This "Code" can then be used for other tasks, like searching for similar images or classifying data, much more efficiently than raw pixels

Why "Squeeze" the Data?

- **Denoising & Pattern Completion**
  - Because the model learns the "essence" of a shape, it becomes robust to errors
  - If you give a trained Autoencoder a blurry or "noisy" image of a '4', the bottleneck (which only knows what a perfect '4' looks like) will ignore the noise
  - The decoder then "hallucinates" the missing parts to create a clean version. This is the foundation of Denoising Autoencoders
- **Dimensionality Reduction (Better than PCA)**
  - PCA can only find linear relationships
  - Autoencoders, thanks to their non-linear activation functions (like ReLU or Sigmoid), can find complex, non-linear correlations that PCA would miss

The Limitation of Vanilla Autoencoders

- **Discrete Latent Points**
  - In a Vanilla AE, the encoder is "too good" at its job. It maps each training image to a single, specific point in the latent space
  - The Result: The latent space becomes a "starry night", isolated points of data surrounded by vast, empty "voids"
  - The "Dead Zone" Problem: If you pick a coordinate in the space between two trained points (e.g., halfway between the code for "Digit 1" and "Digit 7"), the decoder has no idea what to do. It was never trained on those coordinates.

The Limitation of Vanilla Autoencoders

- **The "Gap" in Generation**
  - The Symptom: When you try to "sample" from these empty gaps, the decoder typically outputs "visual gibberish", unrealistic blurs or static
  - The Cause: The model **lacks Regularization**. There is no mathematical force ensuring that the area around a data point or the area between data points is meaningful
- **Discrete vs. Continuous Manifolds**
  - Vanilla AE (Discrete): The model learns a look-up table. Input A →Point A. It doesn't care about the relationship between points
  - VAE Goal (Continuous): We need a space where if you move 0.1 units to the left, the image changes only slightly. We want a Continuous Manifold where every single coordinate results in a realistic-looking output

Importance for “Generation”

To generate new data, we need to be able to:

- **Sample randomly**: Pick any point from a known distribution (like a Gaussian) and get a valid result
- **Interpolate**: Walk from one "concept" to another (e.g., from "Smiling" to "Frowning") and see a smooth transition

A Vanilla Autoencoder is a **Memorizer.** To get a Creator, we must force the latent space to be "full" and "smooth"

Why "Vanilla" AEs Fail to Generate

- **Lack of Regularization**
  - In machine learning, regularization forces a model to prefer "simpler" or "smoother" solutions
  - **Vanilla AE**: Has zero pressure to organize the latent space. The encoder can scatter data points anywhere as long as the decoder can find them again
  - **The Result**: The latent space is unconstrained. There is no incentive for the model to make sure that the area near a "Cat" point also looks like a “Cat"

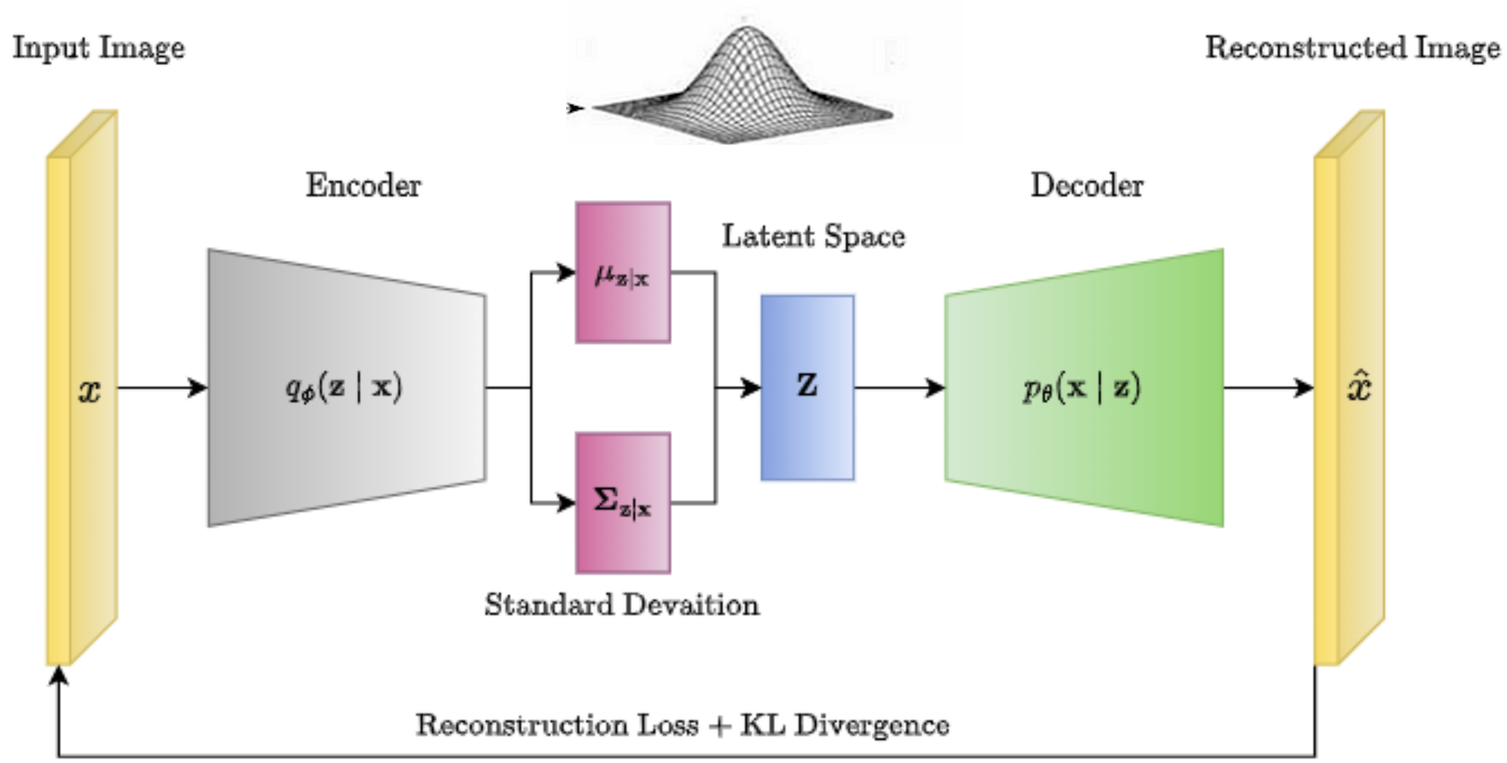
Why "Vanilla" AEs Fail to Generate

- **Overfitting to Specific Points**
  - Because the model is only graded on how well it reconstructs exact training examples, it "memorizes" the coordinates
- **The "Identity Mapping" Trap**
  - If the bottleneck isn't small enough, or if the model is powerful enough, it may stop learning features entirely
  - It finds a way to pass the input through the bottleneck without actually understanding the structure (essentially becoming a complex Identity function)
  - This makes the latent space mathematically correct but semantically useless for generation

Summary: The Generative Breakdown

Feature	Vanilla Autoencoder	What Generation Requires
Latent Structure	Fractured and Gappy	Continuous and Dense
Predictability	High (for known points)	High (for unseen points)
Sampling	Impossible (no known distribution)	Easy (Standard Normal Distribution)
Transitions	Sudden jumps / Blurs	Smooth semantic interpolation

VAE Architecture



Credit: https://medium.com/@rushikesh.shende/autoencoders-variational-autoencoders-vae-and-%CE%B2-vae-ceba9998773d

The VAE Intuition

- **The Probabilistic Bottleneck**
  - In a VAE, the Encoder does not output a fixed vector z. Instead, it outputs two parameters:
    - Mean (μ): The centre of where the data should live
    - Standard Deviation (σ): The "radius" or uncertainty around that centre
  - By doing this, we represent every input as a Gaussian Distribution (a probability cloud) in the latent space

The VAE Intuition

- **"Smearing" the Identity**
  - Because we sample a point from this distribution during training, the Decoder doesn't just see the "perfect" centre point. It sees a slightly different version of the data every time
  - The Result: The Decoder is forced to learn that the entire neighbourhood around μ represents the same object
  - This effectively "smears" the data across the latent space, ensuring that similar images start to overlap and fill the gaps

The VAE Intuition

- **The "Force" of Regularization**
  - While the Encoder tries to give each image a unique neighbourhood, we apply a second force (the KL Divergence) that tries to pull all these clouds toward the centre $(0,0)$
    - This creates a "crowded room" effect
    - Because the clouds are forced to be near each other and stay wide enough to cover their neighbourhoods, they must overlap in a meaningful, organized way
  - By doing this, we represent every input as a Gaussian Distribution (a probability cloud) in the latent space

The VAE Intuition

- **Why This Enables Generation**
  - No More Dead Zones: Since the clouds overlap and cover the entire space, there is no "empty" coordinate. Every point you pick will belong to at least one distribution the Decoder understands
  - Predictable Sampling: Because we force the whole "mess of clouds" to centre around $(0,0)$, we can simply sample from a Standard Normal Distribution to generate brand-new, realistic data

Latent Variables (z)

- A fundamental assumption in Generative Modelling is that the high-dimensional data we see (pixels) is actually controlled by a small number of unobserved, underlying factors
- What are Latent Variables?
  - These are variables that are not explicitly present in the dataset but are responsible for the variations we see
  - The Goal: The VAE's job is to "reverse engineer" the pixels to find these hidden sliders

Example: Generating a Human Face

- If you have a 256 X 256 image of a face, it has 65,536 pixels. However, the "essence" of that face can be described by a few latent factors:
  - z_1: Rotation of the head (Left ↔ Right).
  - z_2: Lighting direction (Sun on left ↔ Sun on right).
  - z_3: Emotional expression (Smile ↔ Frown).
  - z_4: Presence of accessories (Glasses ↔ No glasses)

The Manifold Hypothesis

- This theory suggests that high-dimensional data actually lies on a low-dimensional "manifold" (a curved surface) embedded within the high-dimensional space
  - The Encoder acts as a map-maker: it finds the coordinates on this surface
  - The Decoder acts as a translator: it turns those coordinates back into a full image
- **Discrete vs. Continuous Latent Factors**
  - **Discrete (BERT/GPT)**: Often deals with categories (Words, Parts of Speech)
  - **Continuous (VAE)**: Deals with degrees. You can have 10% of a "smile" or 85% of "head rotation."
  - This is why VAEs are so powerful for images and audio, the real world is rarely "all or nothing"

The Generative Process

- The Three-Step Pipeline - To generate a brand-new image, we follow this sequence:
  - Sampling ($z \sim p(z)$): We pick a random vector z from our Prior. Usually, this is the Standard Normal Distribution $\mathcal{N}(0,I)$
  - Decoding ($g_\phi(z)$): We pass this vector z through the trained Decoder network.
  - Realization ($\hat{x}$): The Decoder transforms the latent coordinates into the final output (pixels, audio samples, etc.)

The Role of the "Prior"

- Because we forced the model during training to organize everything into a unit Gaussian, we know exactly where to look for valid data
- The Center (0,0): Represents the "average" or "most typical" version of the data
- The Tails: Represent the rare, extreme variations

Probabilistic vs. Deterministic Decoding

- Technically, the Decoder doesn't just output pixels; it outputs the parameters of a distribution for pixels (e.g., the mean colour for each pixel)
  - In practice, we usually just take that mean as our final image
  - This is why VAE outputs are often “blurry", the model is essentially averaging out all the possible ways a "Cat" could look based on that specific latent vector z
- Why is this “Generative"?
  - Unlike an Autoencoder, which requires an input image to start the process, the Generative Process requires nothing but a random seed
  - By sampling different z values, we can produce an infinite variety of outputs that follow the "rules" of the training data

The Intractability Problem

The Bayesian Roadblock:

- To find the posterior, we use Bayes' Rule:

$$P(z \mid x) = \frac{P(x \mid z)P(z)}{P(x)}$$

- $P(x \mid z)P(z)$: Easy to calculate (it's our Decoder and our Prior).
- $P(x)$(The Evidence): This is the problem. To find it, we must solve the integral we saw earlier:
  $P(x) = \int P(x \mid z)P(z)dz$

- Why is this "Intractable"?
  - Infinite Search Space: The latent space z is continuous. To solve that integral, we would have to evaluate the Decoder for every single possible value of z
  - The "Needle in a Haystack": Most z values are useless for a specific x

Consequences

- Because we cannot calculate $P(x)$, we cannot calculate the true posterior $P(z \mid x)$
  - We can't "invert" the generator
  - We can't find the exact hidden factors that created our data
  - **The VAE Solution**: If we can't calculate the posterior, we will approximate it
- Enter: **Variational Inference**
  - Instead of solving the math, we use a Neural Network (The Encoder) to guess the shape of $P(z \mid x)$
  - We define a simpler, tractable distribution $q_\phi(z \mid x)$ (usually a Gaussian)
  - We train the network to make $q_\phi(z \mid x)$ as similar as possible to the true $P(z \mid x)$

Variational Inference

- We introduce a new player: $q_\phi(z \mid x)$
- This is our **Encoder** (the "Inference Network").
- It is a neural network with parameters ϕ that "guesses" the distribution of latent variables z for a given input x
- We choose q to be a simple, well-behaved family of distributions—typically a Multivariate Gaussian

Goal: Minimizing Divergence

- We want our "guess" (q) to be as close to the "truth" (p) as possible. To measure the distance between two probability distributions, we use the Kullback–Leibler (KL) Divergence:

$$\min_{\phi} \mathrm{KL}(q_\phi(z \mid x) \mid\mid p(z \mid x))$$

- Why this works
  - By using a neural network as q, we can:
    - Avoid the integral: We don't need to evaluate every possible z to find the “Evidence" $p(x)$
    - Amortize Inference: Once the encoder is trained, finding the latent code for a new image is a single "forward pass" (fast), rather than a iterative search (slow)

The Encoder’s New Output

- Because we chose q to be a Gaussian, the Encoder’s job is now very specific. It must look at an image x and output the two numbers that define a Gaussian "cloud":
  - μ (mu): The predicted centre point of the latent features
  - σ (sigma): The predicted uncertainty/spread of those features
- Inference vs. Generation
  - It helps to see them as two sides of the same coin:
    - Inference (The Encoder): Going from the Real World to the Idea Space ($x \rightarrow z$)
    - Generation (The Decoder): Going from the Idea Space to the Real World ($z \rightarrow x$)

The Encoder-Decoder Architecture in VAEs

- The Inference Network (Encoder)
  - The Encoder's goal is to map the high-dimensional input x(e.g., 784 pixels) into a compressed distribution
    - Input Layer: Receives the raw data x
    - Hidden Layers: Extract increasingly abstract features (edges → shapes → concepts)
    - The Distribution Heads: The final layers don't produce z directly. They produce:
      - μ (Mu): The mean of the latent distribution
      - σ (Sigma): The standard deviation (spread) of the distribution

The Stochastic Bottleneck (Sampling)

- This is where the "Variational" magic happens. We don't pass μ or σ to the decoder. Instead, we sample a random value z from the distribution defined by those parameters:

$$z \sim \mathcal{N}(\mu, \sigma^2)$$

- This means every time the same image passes through the network, the "Code" (z) is slightly different

The Generative Network (Decoder) & Summary

- The Decoder takes the sampled latent vector z and attempts to reconstruct the original input.
  - Input: A single vector z (sampled from the encoder's "cloud").
  - Hidden Layers: Mirror the encoder, gradually expanding the code back into the original dimensions.
  - Output ($\hat{x}$): The reconstructed data.
- Summary of the Data Flow
  - Encode: $x \rightarrow (\mu, \sigma)$
  - Sample: $z = \mu + \sigma \cdot \epsilon$ (where ϵ is random noise)
  - Decode: $z \rightarrow \hat{x}$

Gaussian Latent Space: Why the Normal Distribution $N(\mu, \sigma^2)$ is the "Gold Standard"

- **Mathematical Simplicity and Tractability**
  - The Gaussian distribution is well-behaved.
  - Closed-form solutions: Calculating the KL Divergence between two Gaussians can be done with a simple formula. We don't need expensive simulations; we just plug μ and σ into an equation
  - The Central Limit Theorem: It is the "natural" distribution that many real-world features (like height, weight, or even pixel intensities) tend to follow when many small factors are combined.
- **Smooth and Dense Geometry**
  - The Gaussian distribution has "infinite support," meaning it assigns a non-zero probability to every point in space.
  - No Hard Borders: Unlike a Uniform distribution (which just stops at a boundary), a Gaussian tapers off. This creates a smooth gradient for the model to learn.
  - Concentration of Mass: Most of the "action" happens near the mean, but the "tails" ensure that the model explores the transitions between different types of data.

Gaussian Latent Space: Why the Normal Distribution $N(\mu, \sigma^2)$ is the "Gold Standard"

- **Easy Sampling (The Unit Circle)**
  - By forcing our latent space toward a Standard Normal Prior $\mathcal{N}(0,I)$, we effectively pack our data into a compact, spherical ball centred at the origin.
  - Standardization: This ensures that no single dimension becomes "too powerful" or "too far away" for the decoder to understand.
  - Generating is Easy: To create something new, we don't have to guess where the data is. We just pick a point near $(0,0)$, and we are guaranteed to find something the model was trained on.
- **Independence of Dimensions: In a VAE, we often use a diagonal covariance matrix. This encourages the model to make each dimension of z represent an independent factor of variation.**
  - Dimension 1: Might learn "Rotation"
  - Dimension 2: Might learn "Brightness"
  - Because they are Gaussian, the model can adjust one without necessarily breaking the other.

Deriving the Objective

- By definition, the distance between our "guess" and the "truth" is:

$$D_{KL}(q_\phi(z\,|\,x)\,||\,p(z\,|\,x)) = \mathbb{E}_q[\log q_\phi(z\,|\,x) - \log p(z\,|\,x)]$$

- Remember that $p(z\,|\,x) = \frac{p(x\,|\,z)p(z)}{p(x)}$. Substitute this into the equation:

$$D_{KL} = \mathbb{E}_q\left[\log q_\phi(z\,|\,x) - \log\left(\frac{p(x\,|\,z)p(z)}{p(x)}\right)\right]$$

- Using log properties ($\log\frac{ab}{c} = \log a + \log b - \log c$):

$$D_{KL} = \mathbb{E}_q[\log q_\phi(z\,|\,x) - \log p(x\,|\,z) - \log p(z) + \log p(x)]$$

Deriving the Objective

- Since $\log p(x)$ (the evidence) does not depend on z, it is a constant relative to the expectation $\mathbb{E}_q$:

$$D_{KL}(q||p) = \mathbb{E}_q[\log q_\phi(z|x) - \log p(z) - \log p(x|z)] + \log p(x)$$

- Rearrange to isolate $\log p(x)$:

$$\log p(x) = D_{KL}(q_\phi(z|x)||p(z|x)) + \mathbb{E}_q[\log p(x|z)] - \underbrace{\mathbb{E}_q[\log q_\phi(z|x) - \log p(z)]}_{D_{KL}(q_\phi(z|x)||p(z))}$$

- The Final ELBO Form:

$$\log p(x) = \underbrace{D_{KL}(q_\phi(z|x)||p(z|x))}_{\text{The Error (Gap)}} + \underbrace{\mathbb{E}_q[\log p(x|z)] - D_{KL}(q_\phi(z|x)||p(z))}_{\text{The ELBO (Objective Function)}}$$

Defining the Objective - Evidence Lower Bound (ELBO)

- Why "Lower Bound"?
  - If you can’t reach the ceiling, you try to push the floor as high as possible $\log P(x) \geq$ ELBO
  - By making the ELBO as large as possible through training, we are guaranteed to also push up the actual likelihood of our model generating real-world data
- Why "Evidence"?
  - In Bayesian statistics, $P(x)$ is called the "Evidence." Therefore, the ELBO is literally the Evidence Lower Bound, i.e., the highest point we can reach without knowing the impossible-to-calculate truth

The Two-Part Equation

- The ELBO is composed of two competing terms that create a perfect "tug-of-war" for the model:

$$\text{ELBO} = \text{Reconstruction Reward} - \text{Complexity Penalty}$$

$$\text{ELBO} = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x)||p(z))$$

  - Term A: The Reconstruction Term $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$
    - **Role**: The "Autoencoder" part.
    - **Goal**: It measures how well the decoder can reconstruct the input x from the latent code z. It wants the output to be sharp and accurate
  - Term B: The Regularization Term. $\text{KL}(q_\phi(z|x)||p(z))$
    - **Role**: The "Variational" part.
    - **Goal**: It measures how much our predicted distribution q (from the encoder) deviates from our prior p (the Standard Normal). It wants to keep the latent space organized and centred

ELBO Component 1 — Reconstruction Loss

- The first part of the ELBO equation, $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]$, represents the Expected Log-Likelihood, it ensures that the model actually learns to represent the data accurately
- If the output $\hat{x}$ is a perfect match for the input x, this value is maximized.
- Why an "Expectation" ($\mathbb{E}$)?
  - We encode x to a distribution
  - The $\mathbb{E}$ (Expectation) symbol means we want the reconstruction to be good on average across all the points z that could be sampled from that distribution
  - This forces the decoder to be robust; it can't just memorize one coordinate; it has to understand the whole "cloud"

ELBO Component 1 — Reconstruction Loss

- Mathematical Implementation
  - Binary Data (e.g., black and white digits): We often use Binary Cross-Entropy.
  - Continuous Data (e.g., colour photos): We often use Mean Squared Error (MSE).

$$L_{recon} = \|x - \hat{x}\|^2$$

- The Effect on Latent Space
  - Without any other constraints, this term would cause the model to behave like a Vanilla Autoencoder:
    - It would make the distributions $q(z \mid x)$ as "thin" as possible (tiny σ) to avoid any uncertainty.
    - It would spread data points as far apart as possible to make them easy for the decoder to tell apart.

ELBO Component 2: KL Divergence

- The second part of the ELBO equation, $-\mathrm{KL}(q_\phi(z|x)||p(z))$, is the Regularization Term. While the Reconstruction Loss focuses on the output, the KL Divergence focuses entirely on the structure of the bottleneck
- The KL term measures how far the Encoder's predicted distribution $q(z|x) = \mathcal{N}(\mu, \sigma^2)$ is from the Standard Normal Prior $p(z) = \mathcal{N}(0,I)$
- The Goal: Forces of Symmetry and Centrality
  - This term acts as a "spring" that pulls the encoder's output toward the centre of the latent space. It enforces three critical properties:
    - Cantering: It pulls the Mean (μ) toward 0.
    - Standardizing: It forces the Standard Deviation (σ) toward 1.
    - Completeness: By forcing all data "clouds" toward the same centre, it ensures they overlap. This overlap is what eliminates the "Gaps" or "Dead Zones"

Properties

- In our loss function, we minimize the KL Divergence
  - If the Encoder tries to be "too specific" (making σ very tiny to perfectly memorize a pixel), the KL Divergence increases, penalizing the model
  - If the Encoder tries to hide data in a far-off corner of the map (making μ very large), the KL Divergence increases, penalizing the model

The "Tug-of-War" Dynamics

- **If we only had Reconstruction**: The model would act like a Vanilla AE, creating a messy, fragmented latent space to achieve perfect detail.
- **If we only had KL Regularization**: The model would collapse all data into a single circle at $(0,0)$. Everything would look like a generic, blurry average.
- **The ELBO Balance**: Together, they force the model to be accurate while remaining organized

Practical Implementation Note

- In your code, you will often see a weight added to the KL term, known as β (Beta):

$$\mathscr{L} = \text{Reconstruction} - \beta \cdot \text{KL}$$

- By adjusting β, researchers can decide if they want the model to prioritize high-quality images (low β) or a more "disentangled" and organized latent space (high β)

Summary of the VAE Loss Function

- The Combined Objective ($\mathscr{L}$)

$$\mathscr{L}(\theta, \phi; x) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{Reconstruction Accuracy}} - \underbrace{D_{KL}(q_\phi(z|x)||p(z))}_{\text{Latent Regularization}}$$

- **Too much Reconstruction:** You get a standard Autoencoder
- **Too much KL Divergence:** You get a model that only outputs the "average" of the dataset. Every generated image looks like a blurry, ghostly blob
- **The Sweet Spot**: You get a manifold. Points near each other in latent space look like similar objects, and moving between points creates a smooth transition (interpolation)