Variational Autoencoders (VAEs)

Roko Čubrić

September 2025

1 Introduction to Standard Autoencoders

A standard **autoencoder** is a neural network model used for unsupervised learning of data representations. It consists of two main parts:

1. An **encoder**, which compresses a high-dimensional input x into a low-dimensional latent representation z.
2. A **decoder**, which takes the latent representation z and attempts to reconstruct the original input $\hat{x}$.

The objective of a standard autoencoder is to minimize the reconstruction loss, which is typically the mean squared error (MSE) or binary cross-entropy between the original input x and the reconstructed output $\hat{x}$. This forces the model to learn a compact representation that still retains enough information to reproduce the input.

However, standard autoencoders have a significant limitation: the learned latent space is not guaranteed to be **continuous** or **structured**. They may simply map each input to an isolated, non-overlapping point. When we try to generate new data by sampling a random point from this latent space, the decoder often produces nonsensical output because the sampled point falls into a "gap" where the model has learned nothing.

2 The Variational Autoencoder (VAE)

The **Variational Autoencoder (VAE)** addresses this limitation by taking a fundamentally different approach. It is a generative model that aims to learn not just a point representation, but a **probability distribution** for the latent space. This ensures the latent space is continuous and structured, allowing us to generate new, meaningful data.

The VAE's objective is to maximize the **log-likelihood** of the data, which means we want to maximize the probability $p_\theta(x)$ of our input data (something resembling it) being generated by our model. A higher probability indicates our model has learned to represent the data well. This probability is expressed as:

$$\log p_\theta(x) = \log \int p_\theta(x, z)\, dz$$

This integral is computationally intractable (not theoretically solvable). To solve this, the VAE uses a technique called **variational inference**, where we introduce a simple, tractable distribution $q_\phi(z|x)$ to approximate the true but intractable posterior $p_\theta(z|x)$. The core idea is that we can find a lower bound on our objective, which is easier to optimize. This lower bound is the **Evidence Lower Bound (ELBO)**.

Example: Why Use Log-Likelihood?

When working with probabilities in machine learning, we often multiply many very small numbers together. For example, the probability of a specific image appearing is a product of the probabilities of all its individual pixels. If an image has D pixels, the total probability is a product of D small numbers.

$$p(x) = p(x_1) \cdot p(x_2) \cdot \dots \cdot p(x_D)$$

Since these probabilities are between 0 and 1, multiplying thousands or millions of them together would quickly lead to a number so close to zero that a computer cannot represent it. This is called a **floating-point underflow**, which effectively makes the probability zero and breaks the training process.
To avoid this, we take the logarithm of the probability. A key property of logarithms is that they convert multiplication into summation:

$$\log(p(x)) = \log(p(x_1)) + \log(p(x_2)) + \dots + \log(p(x_D))$$

Summing up numbers is far more numerically stable for a computer than multiplying them. By maximizing the log-likelihood (the sum), we achieve the same goal as maximizing the original probability, but without the risk of numerical errors.

2.1 Deriving the ELBO

The **Evidence Lower Bound (ELBO)** is the lower bound on the log-likelihood that we aim to optimize. It can be derived as follows:

$$\log p_\theta(x) = \log \int p_\theta(x,z)\,dz = \log \int \frac{p_\theta(x,z)}{q_\phi(z|x)} q_\phi(z|x)\,dz$$

Lemma 1 (Jensen's Inequality)**.** *For a concave function f and a random variable X, the following inequality holds: $f(\mathbb{E}[X]) \geq \mathbb{E}[f(X)]$. Since the logarithm function is concave, we can use this inequality to find a lower bound.*

Applying Jensen's Inequality to the equation above, we get the lower bound:

$$\log p_\theta(x) \geq \int q_\phi(z|x) \log\left(\frac{p_\theta(x,z)}{q_\phi(z|x)}\right) dz$$

This expectation can be broken down using the properties of logarithms and probability distributions.

$$\int q_\phi(z|x) \left[\log p_\theta(x,z) - \log q_\phi(z|x)\right] dz$$

$$\int q_\phi(z|x) \left[\log p_\theta(x|z) + \log p(z) - \log q_\phi(z|x)\right] dz$$

Rearranging and grouping terms, we get the final form of the ELBO. First, we separate the integral into two terms:

$$\int q_\phi(z|x) \log p_\theta(x|z)\,dz - \int q_\phi(z|x) \left[\log q_\phi(z|x) - \log p(z)\right] dz$$

Recognizing the definitions of an expectation and the Kullback-Leibler (KL) divergence, we can rewrite the terms as:

$$\mathbb{E}_{z\sim q_\phi}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x) \,\|\, p(z))$$

This gives us the final form of the ELBO:

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

This formula is at the heart of the VAE. It is a sum of two terms that are optimized simultaneously. The loss function used for training is the negative of the ELBO.

2.2 The Two Terms of the VAE Loss Function

The ELBO's two terms have distinct roles and are calculated for each specific data point x during training.

2.2.1 The Reconstruction Term

$$\mathbb{E}_{z\sim q_\phi}[\log p_\theta(x|z)] = \int q_\phi(z|x) \log p_\theta(x|z)\, dz$$

This term measures how well the decoder can reconstruct the original data point x from the latent codes z provided by the encoder. It is a log-likelihood, and its maximization corresponds to minimizing a reconstruction loss (e.g., binary cross-entropy for images). This term trains both the **encoder** and the **decoder**. Unlike a standard autoencoder that predicts a single pixel value, the VAE's decoder outputs the parameters of a probability distribution for each pixel. The final image is created by taking the mean values, but the variance is a crucial component of the loss function. It allows the model to express uncertainty about its predictions; a larger variance will reduce the reconstruction penalty if a prediction is slightly off, which helps the model learn a more robust representation. This is the only part of the VAE that uses the variance of the pixel distributions.

Example: Log-Likelihood Formulas

The formula for $\log p_\theta(x|z)$ depends on the type of data.
For continuous data (e.g., grayscale images with pixel values from 0 to 1), the decoder outputs a Gaussian distribution for each pixel. The log-likelihood is:

$$\log p_\theta(x|z) = \sum_{i=1}^{D} \log \mathcal{N}(x_i; \mu_{\theta,i}(z), \sigma^2_{\theta,i}(z))$$

Here, D is the dimensionality of the data, which is the total number of pixels in the image. The total loss is the sum of the negative log-likelihoods for all pixels, where the loss for a single pixel is:

$$-\log \mathcal{N}(x_i; \mu_{\theta,i}, \sigma^2_{\theta,i}) = \frac{1}{2}\log(2\pi) + \frac{1}{2}\log(\sigma^2_{\theta,i}) + \frac{(x_i - \mu_{\theta,i})^2}{2\sigma^2_{\theta,i}}$$

For binary data (e.g., black and white images with pixel values of 0 or 1), the decoder outputs a Bernoulli distribution for each pixel. The log-likelihood is:

$$\log p_\theta(x|z) = \sum_{i=1}^{D} [x_i \log(\hat{x}_i) + (1 - x_i)\log(1 - \hat{x}_i)]$$

where $\hat{x}_i$ is the decoder's predicted probability for the i-th pixel, and this is equivalent to the negative of the binary cross-entropy loss.

2.2.2 The KL Divergence Term

$$D_{KL}(q_\phi(z|x) \,\|\, p(z)) = \int q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z)}\, dz$$

This is a regularization term that measures the difference between the encoder's output distribution, $q_\phi(z|x)$, and a predefined prior distribution, $p(z)$ (typically a standard normal distribution, $\mathcal{N}(0, I)$). By minimizing this term, we force the latent distributions for all our data points to be organized and centered around the origin, which creates a continuous and structured latent space. This term trains only the **encoder**. The analogy is that instead of isolated data points on a map, this term forces the data points to be on well-defined streets that can be smoothly navigated.

3 Backpropagation and the Reparameterization Trick

Since the VAE trains using backpropagation, we need a way to pass gradients from the loss function all the way back to the encoder's parameters. A critical problem arises when we try to sample a random latent variable z from our distribution $q_\phi(z|x)$. The act of random sampling is a non-differentiable operation, which would break the backpropagation chain.

The **reparameterization trick** solves this problem by separating the randomness from the model's parameters. Instead of sampling from $q_\phi(z|x)$, we sample a random variable ϵ from a simple standard normal distribution, $\mathcal{N}(0, I)$, and then compute z as a deterministic, differentiable function of μ and σ (the mean and standard deviation output by the encoder).

$$z = \mu + \sigma \cdot \epsilon$$

This allows us to backpropagate gradients from the reconstruction loss back through the encoder.

- The gradient with respect to μ is straightforward: $\frac{\partial \mathcal{L}}{\partial \mu} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial \mu} = \frac{\partial \mathcal{L}}{\partial z} \cdot 1$.
- The gradient with respect to σ is now also possible: $\frac{\partial \mathcal{L}}{\partial \sigma} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial \sigma} = \frac{\partial \mathcal{L}}{\partial z} \cdot \epsilon$.

The trick is essential for the variance part of the backpropagation as it makes the random sampling step differentiable.

The reparameterization trick is more accurately viewed not as a trick, but as the only logical method for computing the latent variable z in a way that allows for the necessary backpropagation and gradient calculation.

4 Amortized Inference and Latent Space Arithmetic

The VAE's design allows for other key technical advantages.

Amortized Inference

During training, we "amortize" the cost of finding the approximate posterior. Instead of running a separate optimization for each data point, we train a single neural network (the encoder) to learn a general mapping from any data point to its latent distribution parameters. This makes the VAE incredibly efficient at inference time, as it can instantly find a latent representation for any new input. This is analogous to doing all the prep work for a feast (e.g., chopping all vegetables) once, so that each dish can be cooked quickly.

Latent Space Arithmetic

Because the VAE learns a continuous and structured latent space, we can perform meaningful arithmetic on latent vectors. A famous example is manipulating the concept of "glasses." We can encode an image of a person with glasses ($z_{\text{with glasses}}$) and an image of the same person without glasses ($z_{\text{without glasses}}$), and then find a vector representing the "glasses" concept:

$$z_{\text{glasses}} = z_{\text{with glasses}} - z_{\text{without glasses}}$$

This "glasses" vector can then be added to the latent vector of a different person, and the decoder can produce an image of that person with glasses. This is a direct result of the structured latent space created by the KL divergence term.

5 Conclusion

The VAE is a powerful generative model that overcomes the limitations of a standard autoencoder by learning a continuous and structured latent space. This is achieved through the ELBO, a unique loss function that balances reconstruction quality with regularization. The reparameterization trick allows for efficient training via backpropagation, and concepts like amortized inference enable the model to be practical for a wide range of applications.