Course Materials for GEN-AI

Northeastern University

These materials have been prepared and sourced for the course **GEN-AI** at Northeastern University. Every effort has been made to provide proper citations and credit for all referenced works.

If you believe any material has been inadequately cited or requires correction, please contact me at:

Instructor: Ramin Mohammadi
r.mohammadi@northeastern.edu

Thank you for your understanding and collaboration.

Content sourced from multiple references - see Reference section for details

Latent variable models

Figure 1: CelebA

There is significant variability in images $\mathbf{x}$ due to factors such as gender, eye color, hair color, pose, and more. However, unless the images are annotated, these factors of variation are not explicitly available and remain latent.

Idea: Explicitly model these factors using latent variables $\mathbf{z}$.

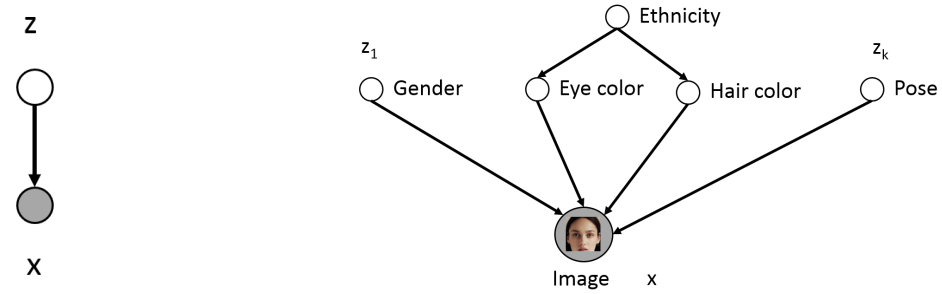


Figure 2: on the right we have a conceptually we can see this as Bayesian network - Not exactly valid

Observed data $\mathbf{x}$ (e.g., image pixel values) are modeled as being influenced by these latent variables.

1. Only shaded variables $\mathbf{x}$ are observed in the data (pixel values).
2. Latent variables $\mathbf{z}$ correspond to high-level features:
    - If $\mathbf{z}$ is chosen properly, we can have several :

- Learning $p(\mathbf{x} \mid \mathbf{z})$ could be much simpler than $p(\mathbf{x})$.
- $p(\mathbf{x} \mid \mathbf{z})$ Could provide us with a clustering over data.

- If the model is trained, we can identify features via $p(\mathbf{z} \mid \mathbf{x})$, e.g., $p(\text{EyeColor} = \text{Blue} \mid \mathbf{x})$.

3. **Challenge:** It is very difficult to specify these conditional probabilities manually.

Deep Latent Variabel Models

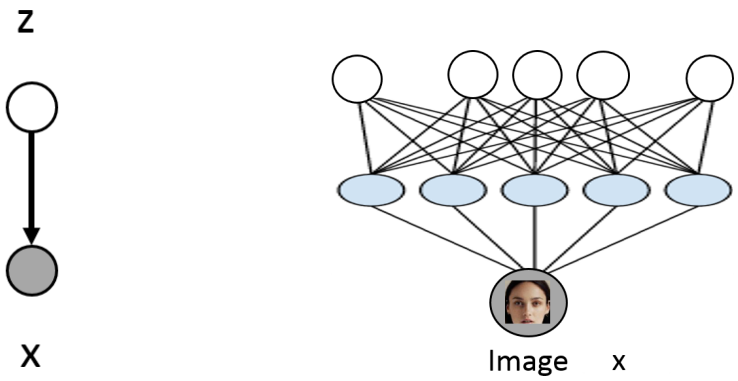


Figure 3: Deep latent variable models

Use neural networks to model conditionals:

- $$z \sim \mathcal{N}(0, I)$$
- $$p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}\left(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z})\right)$$

  where μ_θ and Σ_θ are neural networks.

Hope that after training, $\mathbf{z}$ *will correspond to meaningful latent factors of variation (features).* This represents unsupervised representation learning. As before, features can be computed via $p(\mathbf{z} \mid \mathbf{x})$.

Generative Process

1. Sample $\mathbf{z}$ from $z \sim \mathcal{N}(0, I)$.
2. Feed z to your NNs and get the μ and Σ.
3. Generate a data point by sampling from the corresponding Gaussian with the the μ and Σ.

Challenge: Evaluation of $P(x|z)$: The main challenge is that the z variables are not observed during training, making it unclear how to update the parameters of the neural networks (θ) in a way that relates z to the observed x.

Intuitively, we estimate the value of z for a given x and then use specific techniques to address this problem (e.g., the EM algorithm).

1 Types of Latent Variable Models

Mixture of Gaussians - a Shallow Latent Variable Model

In a one-dimensional mixture of Gaussians, the latent variable z is discrete, and the prior $P_r(z)$ is modeled as a categorical distribution with probabilities λ_n for each possible value of z. The likelihood $P_r(\mathbf{x} \mid z = n)$ of the data $\mathbf{x}$, given that the latent variable z takes on the value n, is modeled as a normal distribution with mean μ_n and variance σ_n^2:

$$\mathbf{z} \sim \text{Categorical}(1, \ldots, N), \quad P_r(z = n) = \lambda_n, \quad P_r(\mathbf{x} \mid z = n) = \text{Norm}[\mu_n, \sigma_n^2].$$

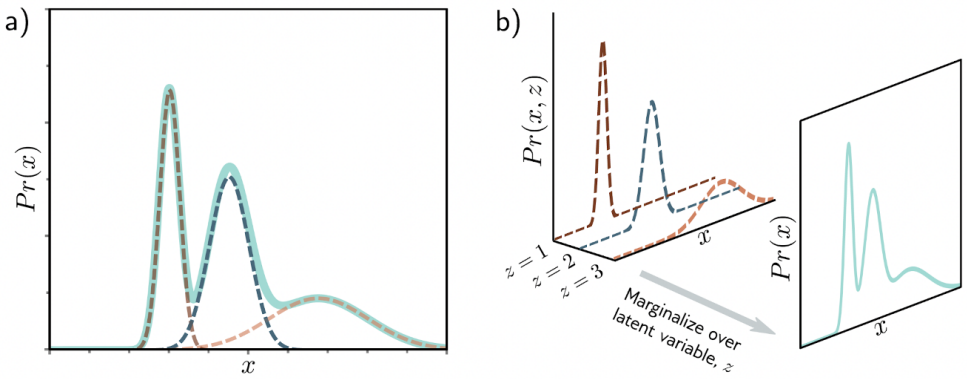


Figure 4: The Mixture of Gaussians (MoG) models a complex probability distribution as a weighted sum of Gaussian components, which is obtained by marginalizing the joint density $P_r(x, z)$ that relates the continuous observed data x and a discrete latent variable z.

The probability $P_r(\mathbf{x})$ is obtained by marginalizing over the latent variable z. Since z is discrete in this case, the marginalization is expressed as a summation over all possible values of z:

$$P_r(\mathbf{x}) = \sum_{n=1}^{N} P_r(\mathbf{x}, z = n)$$

$$P_r(\mathbf{x}) = \sum_{n=1}^{N} P_r(\mathbf{x} \mid z = n) \cdot P_r(z = n)$$

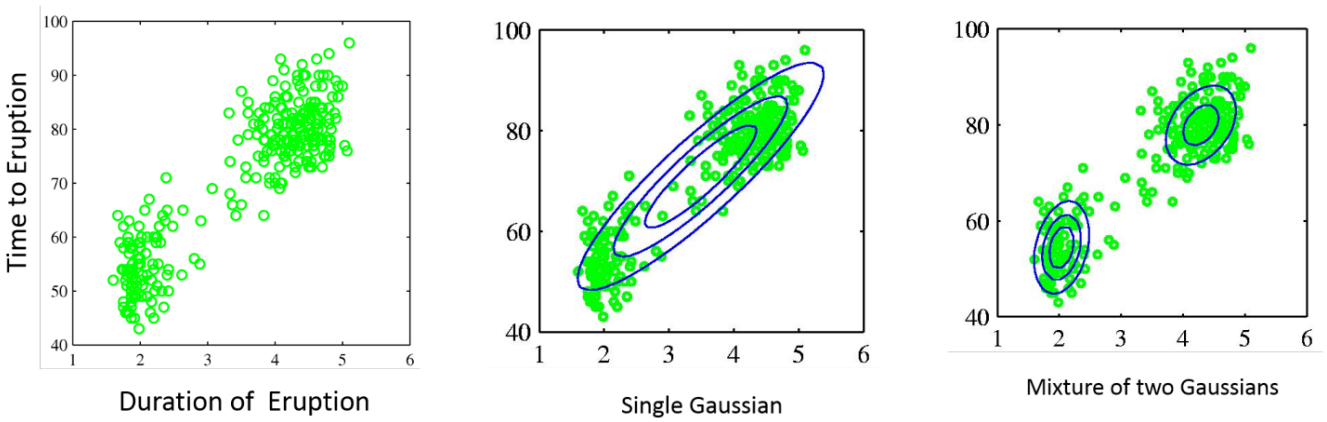
$$P_r(\mathbf{x}) = \sum_{n=1}^{N} \lambda_n \cdot \text{Norm}[\mu_n, \sigma_n^2].$$

Using simple expressions for the likelihood and prior, we can describe a complex multi-modal probability distribution.

Generative Process

1. Pick a mixture component k by sampling $\mathbf{z}$.
2. Generate a data point by sampling from the corresponding Gaussian.

Example:



- **Clustering:** The posterior $p(\mathbf{z} \mid \mathbf{x})$ identifies the mixture component.
- **Unsupervised Learning:** We aim to learn from unlabeled data, (ill-posed problem - the definition of good learning is not obvious).
- most likely will fail when applied to a large image dataset (might work if we select k also large)

Variational AutoEncoder - Nonlinear and continuous latent variable model

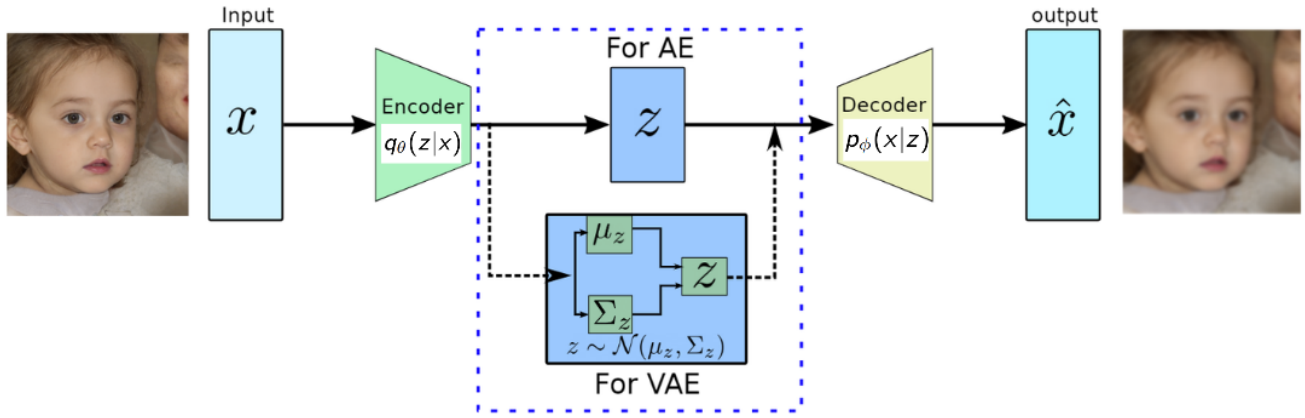


Figure 5: Variational AE

In the nonlinear latent variable model, both the observed data $\mathbf{x}$ and the latent variable $\mathbf{z}$ are continuous and multivariate. The prior $P_r(\mathbf{z})$ is modeled as a standard multivariate normal:

$$P_r(\mathbf{z}) = \mathcal{N}[0, I].$$

The likelihood $P_r(\mathbf{x} \mid \mathbf{z})$ is also modeled as a normal distribution, where its mean is a nonlinear function $f_\phi(\mathbf{z})$ of the latent variable and its covariance $\sigma^2 I$ is spherical:

$$P_r(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}[\mathbf{f}_\phi(\mathbf{z}), \sigma^2 I].$$

The function $f_\phi(\mathbf{z})$ is parameterized by a deep neural network, where the latent variable $\mathbf{z}$ captures lower-dimensional aspects of the data, while the remaining unexplained aspects are attributed to the noise $\sigma^2 I$.

The data probability $P_r(\mathbf{x})$ is derived by marginalizing over the latent variable $\mathbf{z}$:

$$P_r(\mathbf{x}) = \int P_r(\mathbf{x} \mid \mathbf{z}) P_r(\mathbf{z}) d\mathbf{z}.$$

This approach can be interpreted as an infinite weighted sum of spherical Gaussian distributions, where the weights are $P_r(\mathbf{z})$ and the means are the network outputs $\mathbf{f}_\phi(\mathbf{z})$.

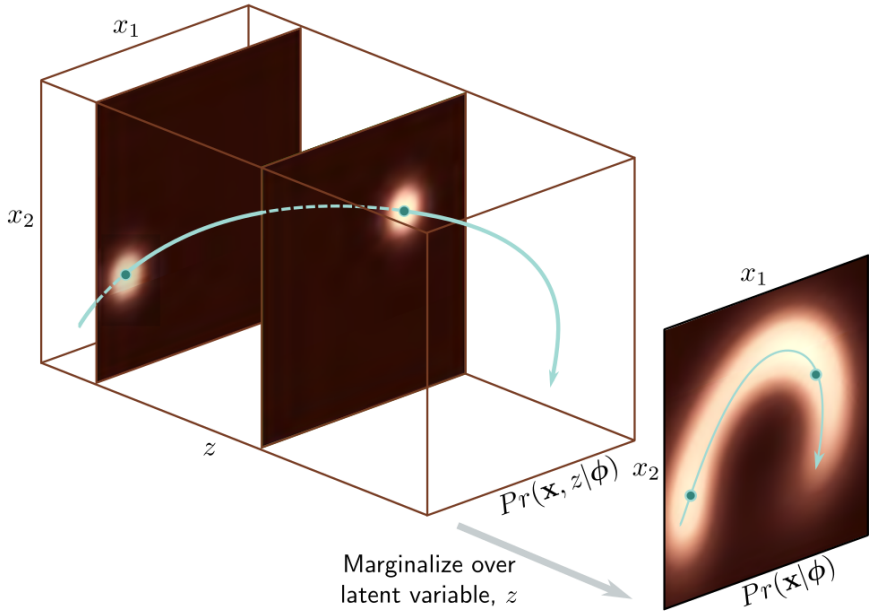


Figure 6: The nonlinear latent variable model generates a complex 2D density $P_r(\mathbf{x})$ by marginalizing the joint distribution $P_r(\mathbf{x}, \mathbf{z})$ over the latent variable $\mathbf{z}$; this is achieved by integrating the 3D volume over the dimension $\mathbf{z}$. For each $\mathbf{z}$, the distribution over $\mathbf{x}$ is a spherical Gaussian with a mean $f[\mathbf{z}, \phi]$, which is a nonlinear function of $\mathbf{z}$ and depends on parameters ϕ. The resulting distribution $P_r(\mathbf{x})$ is a weighted sum of these Gaussians.

Generative Process

1. Sample $\mathbf{z}^*$ from the prior $P_r(\mathbf{z})$.
2. Pass $\mathbf{z}^*$ through the network $f[\mathbf{z}^*, \phi]$ to compute the mean of the likelihood $P_r(\mathbf{x} \mid \mathbf{z}^*, \phi)$.
3. Generate $\mathbf{x}^*$ by sampling from the likelihood distribution.

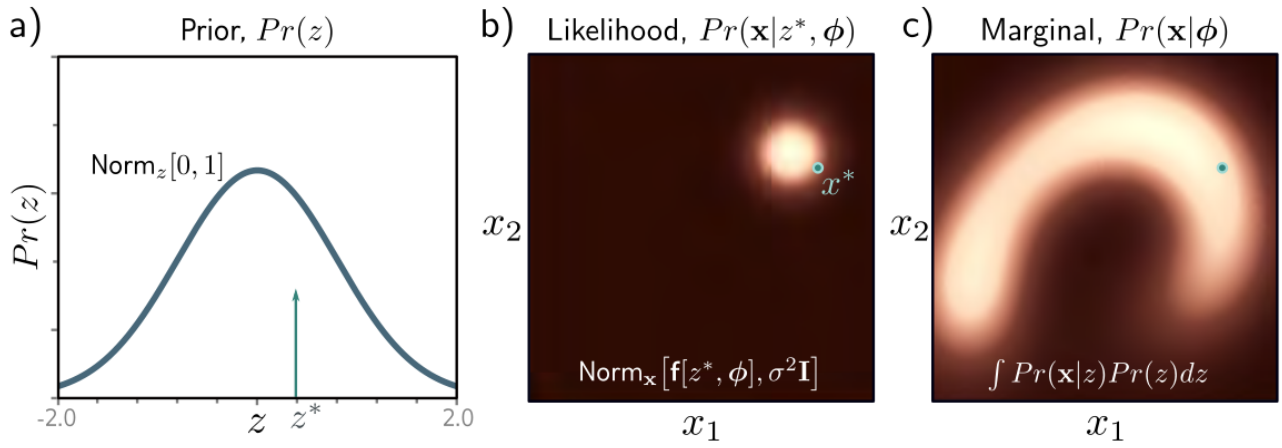


Figure 7: Generation from the nonlinear latent variable model involves sampling $\mathbf{z}^*$ from the prior $P_r(\mathbf{z})$, then sampling $\mathbf{x}^*$ from $P_r(\mathbf{x} \mid \mathbf{z}^*, \phi)$, which is a spherical Gaussian with mean $f[\mathbf{z}^*, \phi]$ (a nonlinear function of $\mathbf{z}^*$) and fixed variance $\sigma^2 I$. Repeating this process multiple times recovers the density $P_r(\mathbf{x} \mid \phi)$.

Marginal Likelihood

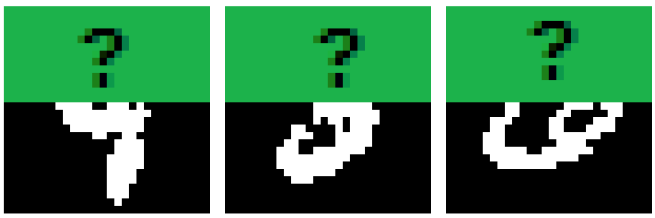


Figure 8: Missing pixels

Discrete Space - Autoregressive

Let's assume we are developing an Autoregressive model to generate image. Suppose some pixel values are missing at train time (e.g., the top half of an image).

- Let $\mathbf{X}$ denote the observed random variables and $\mathbf{Z}$ the unobserved (hidden or latent) variables.
- Assume we have a model for the joint distribution (e.g., PixelCNN):

$$p(\mathbf{X}, \mathbf{Z}; \phi)$$

- We have a dataset $\mathcal{D}$, where for each datapoint the $\mathbf{X}$ variables are observed (e.g., pixel values), and the $\mathbf{Z}$ variables are never observed (e.g., cluster or class ID). The dataset is defined as:

$$\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}\}.$$

- What is the probability $p(\mathbf{X} = \bar{\mathbf{x}}; \phi)$ of observing a training data point $\bar{\mathbf{x}}$?
  - This can be computed as:

$$p(\mathbf{X} = \bar{\mathbf{x}}; \phi) = \sum_{\mathbf{z}} p(\mathbf{X} = \bar{\mathbf{x}}, \mathbf{Z} = \mathbf{z}; \phi) = \sum_{\mathbf{z}} p(\bar{\mathbf{x}}, \mathbf{z}; \phi).$$

  - To compute this, we need to consider all possible ways to complete the image (fill in the missing parts, e.g., the green part).

- Maximum likelihood learning:
  - The parameters are optimized by maximizing the log-likelihood:

$$\ell(\phi; \mathcal{D}) = \log \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \phi) = \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \phi),$$

$$\ell(\phi; \mathcal{D}) = \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \phi).$$

  - Evaluating $\log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \phi)$ can be intractable. For example:
    - If we have 30 binary latent features, $\mathbf{z} \in \{0, 1\}^{30}$, evaluating $\sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \phi)$ involves a sum with 2^{30} terms.

VAE Marginal Likelihood - continuous space

Similarly, to train the VAE model, we maximize the log-likelihood over a training dataset $\{\mathbf{x}_i\}_{i=1}^{I}$ with respect to the model parameters. For simplicity, the variance term σ^2 in the likelihood expression is assumed to be known, and we focus on learning ϕ:

$$\hat{\phi} = \arg\max_{\phi} \sum_{i=1}^{I} \log \left[P_r(\mathbf{x}_i \mid \phi) \right],$$

where:

$$P_r(\mathbf{x}_i \mid \phi) = \int \text{Norm}_{\mathbf{x}_i} \left[\mathbf{f}[\mathbf{z}, \phi], \sigma^2 I \right] \cdot \text{Norm}_{\mathbf{z}}[0, I] \, d\mathbf{z}.$$

- z are unobserved during training.
- To compute $P_r(\mathbf{x}_i \mid \phi)$, we must consider all possible values of $\mathbf{z}$, combining their likelihood of generating $\mathbf{x}_i$ (through $\mathbf{f}[\mathbf{z}, \phi]$) and their own probability, effectively summing over every possible $\mathbf{z}$.

Unfortunately, this optimization problem is intractable because there is no closed-form expression for the integral, and evaluating it for a specific value of $\mathbf{x}$ is not straightforward.

Why is parameter learning in presence of Partially Observed Data challenging?

- Is not decomposable (by variable and parent assignment).
- Is not unimodal as a function of ϕ.
- Could still be optimized using gradient descent.
- Is hard to compute and take gradients ∇_ϕ due to too many possible completions.

Solution? We need some kind of approximations, and it needs to be a cheap approach as we need to iterate over the data so many times.

Approximation approaches:

- **Monte Carlo - MC**
- **Importance Sampling**
- **Estimate Lower Bound - ELBO**

Naive Monte Carlo

The likelihood function for partially observed data is challenging to compute. Suppose that $\mathbf{z}$ is sampled from a **uniform distribution** over $\mathcal{Z}$, which can either be a discrete set or a continuous space. The uniform distribution ensures that all values of $\mathbf{z}$ are equally likely, with the probability or density given by $\frac{1}{|\mathcal{Z}|}$.

Likelihood Function for Partially Observed Data

For **discrete latent variables** $\mathbf{z}$, the likelihood is computed via summation:

$$\sum_{\mathbf{z}\in\mathcal{Z}} p_\phi(\mathbf{x},\mathbf{z}) = |\mathcal{Z}| \sum_{\mathbf{z}\in\mathcal{Z}} \frac{1}{|\mathcal{Z}|} p_\phi(\mathbf{x},\mathbf{z}) = |\mathcal{Z}| \mathbb{E}_{\mathbf{z}\sim\text{Uniform}(\mathcal{Z})}[p_\phi(\mathbf{x},\mathbf{z})].$$

For **continuous latent variables** $\mathbf{z}$, the likelihood involves integration:

$$\int_{\mathcal{Z}} p_\phi(\mathbf{x},\mathbf{z})\, d\mathbf{z} = |\mathcal{Z}| \int_{\mathcal{Z}} \frac{1}{|\mathcal{Z}|} p_\phi(\mathbf{x},\mathbf{z})\, d\mathbf{z} = |\mathcal{Z}| \mathbb{E}_{\mathbf{z}\sim\text{Uniform}(\mathcal{Z})}[p_\phi(\mathbf{x},\mathbf{z})].$$

Explanation of Terms:

- $\frac{1}{|\mathcal{Z}|}$: Ensures that $\sum_{\mathbf{z}\in\mathcal{Z}} \frac{1}{|\mathcal{Z}|} p_\phi(\mathbf{x},\mathbf{z})$ (or $\int_{\mathcal{Z}} \frac{1}{|\mathcal{Z}|} p_\phi(\mathbf{x},\mathbf{z})\, d\mathbf{z}$) represents an **expectation** under a uniform distribution over $\mathcal{Z}$.
- $|\mathcal{Z}|$: Restores the original scale of the summation or integration after transforming it into an expectation. Without this factor, the result would approximate the average rather than the total likelihood.

Monte Carlo Estimation

To compute the likelihood approximately, we use Monte Carlo sampling.

Procedure:

1. Sample $\mathbf{z}^{(1)},\ldots,\mathbf{z}^{(k)}$ uniformly from $\mathcal{Z}$.
2. For **discrete** $\mathcal{Z}$, approximate the likelihood as:

$$\sum_{\mathbf{z}\in\mathcal{Z}} p_\phi(\mathbf{x},\mathbf{z}) \approx |\mathcal{Z}| \frac{1}{k} \sum_{j=1}^{k} p_\phi(\mathbf{x},\mathbf{z}^{(j)}).$$

3. For **continuous** $\mathcal{Z}$, approximate the likelihood as:

$$\int_{\mathcal{Z}} p_\phi(\mathbf{x},\mathbf{z})\, d\mathbf{z} \approx |\mathcal{Z}| \frac{1}{k} \sum_{j=1}^{k} p_\phi(\mathbf{x},\mathbf{z}^{(j)}).$$

Challenges:

- The *curse of dimensionality* implies that most sampled $\mathbf{z}$ values will have very low probabilities, requiring a large number of samples for reliable estimates.
- Rarely sampled regions may dominate $p_\phi(\mathbf{x},\mathbf{z})$, leading to high variance in the estimator.
- Naive uniform sampling may be inefficient, motivating the need for alternative sampling methods (e.g., importance sampling or variational inference) to reduce the variance of the estimate.

Importance Sampling

A better approach is to use importance sampling. Here, we sample $\mathbf{z}$ from an auxiliary distribution $q(z)$, evaluate $\Pr(x \mid z_n)$, and rescale the resulting values by the probability $q(z)$ under the new distribution:

$$\begin{aligned}\Pr(x) &= \int \Pr(x|z)\Pr(z)dz \\ &= \int \frac{\Pr(x|z)\Pr(z)}{q(z)}q(z)dz \\ &= \mathbb{E}_{q(z)}\left[\frac{\Pr(x|z)\Pr(z)}{q(z)}\right] \\ &\approx \frac{1}{N}\sum_{n=1}^{N}\frac{\Pr(x|z_n)\Pr(z_n)}{q(z_n)},\end{aligned}$$

We now draw samples from $q(z)$. If $q(z)$ is close to the region of z where $\Pr(x|z)$ has high likelihood, the sampling will focus on the relevant area of space, allowing us to estimate $\Pr(x)$ much more efficiently.

Motivation for Choosing $q(z)$

The product $\Pr(x|z)\Pr(z)$ that we aim to integrate is proportional to the posterior distribution $\Pr(z|x)$, as stated by Bayes' rule:

$$\Pr(z|x) = \frac{\Pr(x|z)\Pr(z)}{\Pr(x)}.$$

This implies:

$$\Pr(x|z)\Pr(z) \propto \Pr(z|x).$$

Therefore, a sensible choice of the auxiliary distribution $q(z)$ is the variational posterior $q(z|x)$, which approximates $\Pr(z|x)$. This posterior is often computed by the encoder in variational inference frameworks.

Why Use $q(z|x)$:

- $q(z|x)$ is designed to focus on regions of z that are relevant to the observed data x, i.e., where the posterior $\Pr(z|x)$ is large.
- By sampling $\mathbf{z}$ from $q(z|x)$, we increase the efficiency of importance sampling, as the samples are concentrated in regions where $\Pr(x|z)\Pr(z)$ contributes significantly to the integral.

Applications

Using this approach, we can:

- Approximate the probability of new samples with better accuracy compared to naive methods.
- Evaluate the model's quality by computing the log-likelihood of test data.
- Detect anomalies by determining whether new examples belong to the learned distribution.

Challenges of Using Importance Sampling in VAEs

- **High Variance of Estimates:**

  - Variability in Weights: If the auxiliary distribution $q(z)$ does not closely approximate the target distribution $p(z|x)$, the weights $\frac{p(x|z)p(z)}{q(z)}$ can vary greatly, leading to high variance.
  - Rare Events: Crucial contributions from rare but important regions under the target distribution might be missed if they are undersampled by $q(z)$.

- **Computational Efficiency:**
  - Complexity in Normalizing Constant: Estimating or knowing the normalizing constants involved in the distributions $p(z)$ and $p(x|z)$ can be computationally intensive, making the process inefficient.

- **Sample Efficiency:**
  - Scale of Sampling: A very large number of samples may be required to achieve a stable and accurate estimate, which is computationally expensive and slow.

- **Practical Implementation:**
  - Difficulty in Optimal Distribution Selection: Selecting an optimal $q(z)$ that minimizes variance while being computationally feasible is challenging in practice.

Evidence Lower bound - ELBO

Reminder, our goal is to maximize:

$$\ell(\phi; \mathcal{D}) = \sum_{\mathbf{x} \in \mathcal{D}} \log \int \text{Norm}_{\mathbf{x}_i} \left[\mathbf{f}[\mathbf{z}, \phi], \sigma^2 I \right] \cdot \text{Norm}_{\mathbf{z}}[0, I] \, d\mathbf{z}.$$

where:

- $\Pr(x, z \mid \phi) = \text{Norm}_{\mathbf{x}_i} \left[\mathbf{f}[\mathbf{z}, \phi], \sigma^2 I \right]$
- $\Pr(\mathbf{z}) = \text{Norm}_{\mathbf{z}}[0, I]$

As we saw earlier, Log-likelihood function for partially observed data is hard to compute.

$$\log \int \text{Norm}_{\mathbf{x}_i} \left[\mathbf{f}[\mathbf{z}, \phi], \sigma^2 I \right] \cdot \text{Norm}_{\mathbf{z}}[0, I] \, d\mathbf{z} = \log \mathbb{E}_{q(z|x)} \left[\Pr(x|z, \phi) \right].$$

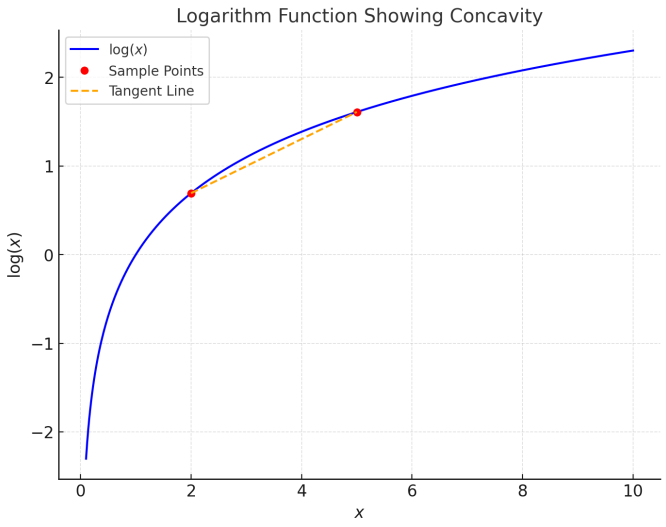


Figure 9: Logarithm Function Showing Concavity

We know that log is a concave function $\log\left(px + (1-p)x'\right) \geq p\log(x) + (1-p)\log(x')$.

To make progress, we define a lower bound on the log-likelihood. This is a function that is always less than or equal to the log-likelihood for a given value of ϕ and will also depend on some other parameters θ. Eventually, we will build a network to compute this lower bound and optimize it. To define this lower bound, we need Jensen's inequality.

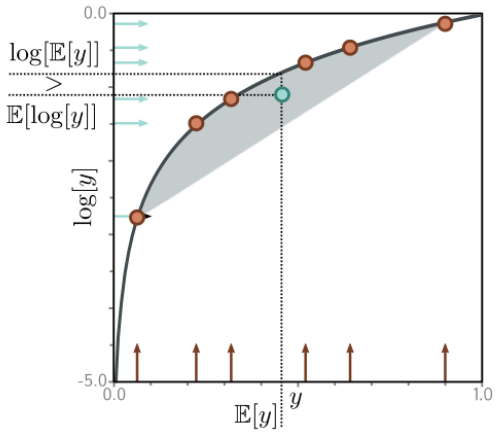


Figure 10: Jensen's inequality (discrete case): The logarithm (black curve) is a concave function; you can draw a straight line between any two points on the curve, and this line will always lie underneath it. It follows that any convex combination (weighted sum with positive weights that sum to one) of the six points on the log function must lie in the gray region under the curve. Here, we have weighted the points equally (i.e., taken the mean) to yield the cyan point. Since this point lies below the curve, $\log[\mathbb{E}[y]] > \mathbb{E}[\log[y]]$.

Jensen's Inequality

Jensen's inequality says that a concave function $g(y)$ (e.g log is concave) of the expectation of data y is greater than or equal to the expectation of the function of the data:

$$g(\mathbb{E}[y]) \geq \mathbb{E}[g(y)].$$

In this case, the concave function is the logarithm, so we have:

$$\log \mathbb{E}[y] \geq \mathbb{E}[\log(y)].$$

Or, writing out the expression for the expectation in full, we have:

$$\log \int p(y) y dy \geq \int p(y) \log(y) dy.$$

This is explored further. In fact, the slightly more general statement is true:

$$\log \int \frac{p(y)}{q(y)} q(y) dy \geq \int q(y) \log \frac{p(y)}{q(y)} dy,$$

where $q(y)$ is a function of y. This follows because $q(y)$ is another random variable with a new distribution with some unknown parameters. Since we never specified $p(y)$, the relation remains true.

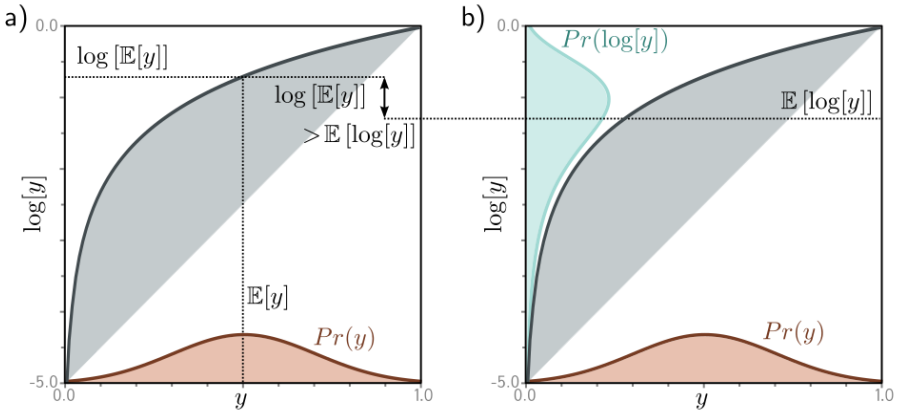


Figure 11: Jensen's inequality (continuous case): For a concave function, $\log[\mathbb{E}[y]] \geq \mathbb{E}[\log[y]]$, as applying the function to the expectation gives a result greater than or equal to the expectation of the function applied to the variable, compressing high values of y relative to low values and lowering the expected value.

Variational Approximation/Inference

To circumvent this issue, variational approximation is employed. This approach involves selecting a simple parametric form for $q(z|\theta)$ to approximate the true posterior ($\Pr(z|x,\phi)$). Typically, a multivariate normal distribution with mean μ and diagonal covariance Σ is used. While this approximation may not perfectly match the posterior, it is effective under certain conditions.

We now use Jensen's inequality to derive the lower bound for the log-likelihood. We start by multiplying and dividing the log-likelihood by an arbitrary parametric probability distribution $q(z|\theta)$ over the latent variables

$$\begin{aligned}
\log \mathbb{E}_{q(z|x,\theta)}[\Pr(x|z,\phi)] &= \log\left[\int \Pr(x,z \mid \phi)\, dz\right] \\
&= \log\left[\int \frac{\Pr(x,z \mid \phi)}{q(z \mid \theta)} q(z \mid \theta)\, dz\right], \\
\log\left[\int \frac{\Pr(x,z \mid \phi)}{q(z \mid \theta)} q(z \mid \theta)\, dz\right] &\geq \int q(z \mid \theta) \log\left[\frac{\Pr(x,z \mid \phi)}{q(z \mid \theta)}\right] dz
\end{aligned}$$

On the left, we have the logarithm of the marginal probability of a data point. On the right, we refer to it as the evidence lower bound, or ELBO. This term derives from the fact that $Pr(x|\phi)$ is known as the evidence within the framework of Bayes' rule. Our objective is to compute the ELBO.

This means if we optimize the ELBO (right hand side) the left hand side value will also go up, which is our goal.

General/First definition:

$$\text{ELBO}(\theta,\phi) = \int q(z \mid \theta) \log\left[\frac{\Pr(x,z \mid \phi)}{q(z \mid \theta)}\right] dz.$$

The ELBO is defined as:

For continuous z:

$$\log \Pr(x;\phi) \geq \int q(z|\theta) \log\left[\frac{\Pr(x,z|\phi)}{q(z|\theta)}\right] dz$$

For discrete z:

$$\log \Pr(x;\phi) \geq \sum_z q(z|\theta) \log \left[\frac{\Pr(x,z|\phi)}{q(z|\theta)} \right].$$

$$\log \Pr(x;\phi) \geq \sum_z q(z;\theta) \log \Pr(z,x;\phi) - \sum_z q(z;\theta) \log q(z;\theta)$$

where:

$$\mathcal{L}(x^i;\phi,\theta^i) = \sum_z q(z;\theta^i) \log \Pr(z,x^i;\phi) + H(q(z;\theta^i)),$$

$$= \mathbb{E}_{q(z;\theta^i)}[\log \Pr(z,x^i;\phi) - \log q(z;\theta^i)].$$

This is similar to what we do in EM algorithm, however EM is not scalable.

Overall, any value of ϕ will gives us a lower bound estimate, our goal is to find a ϕ that gives us the highest lower bound estimate.

Here are two ways to represent $\Pr(x, z \mid \phi)$, each leading to a different expression of the ELBO:

- Using the marginal probability of x and the conditional probability of z given x:

$$\Pr(x, z \mid \phi) = \Pr(z \mid x, \phi) \Pr(x \mid \phi).$$

- Using the marginal probability of z and the conditional probability of x given z:

$$\Pr(x, z \mid \phi) = \Pr(x \mid z, \phi) \Pr(z).$$

Discussion Both formulations of ELBO incorporate different aspects of the joint distribution:

- **ELBO_1** emphasizes the role of conditional dependencies and the overall likelihood of the data.
- **ELBO_2** focuses on the conditional dependency of x on z and the influence of a general prior on z.

Each definition suits different scenarios and assumptions about the data and underlying model, providing flexibility in how the ELBO is applied to optimize the variational inference process.

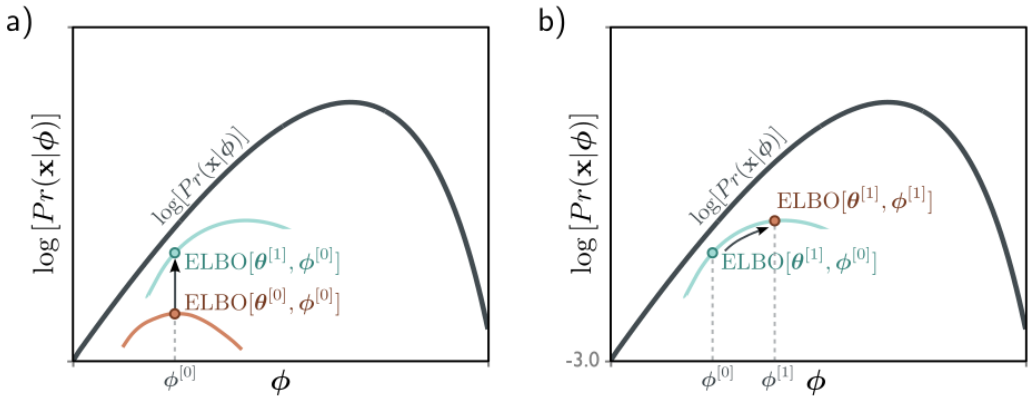


Figure 12: Evidence lower bound (ELBO): The goal is to maximize the log-likelihood log[Pr(x | ϕ)] (black curve) with respect to the parameters ϕ. The ELBO is a function that lies everywhere below the log-likelihood and depends on both ϕ and a second set of parameters θ. For fixed θ, we obtain a function of ϕ (two colored curves for different values of θ). Consequently, the log-likelihood can be increased by either improving the ELBO with respect to (a) the parameters θ (moving from one colored curve to another) or (b) the parameters ϕ (moving along the current colored curve).

Tightness of Bound

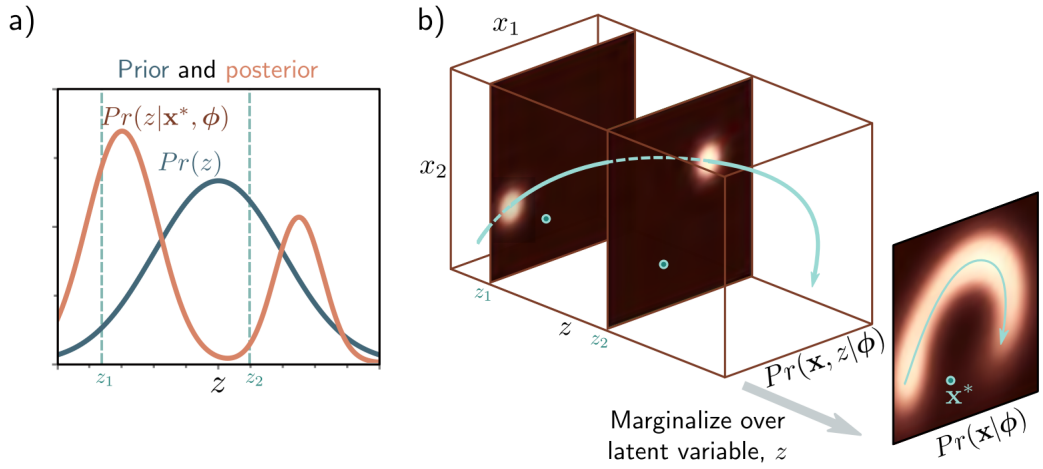


Figure 13: Posterior distribution over latent variable: (a) The posterior distribution $\Pr(z|x^*, \phi)$ is the distribution over the values of the latent variable z responsible for a data point x^*. Using Bayes' rule, $\Pr(z|x^*, \phi) \propto \Pr(x^*|z, \phi)\Pr(z)$. (b) The first term on the right-hand side (the likelihood) is computed by assessing the probability of x^* against the symmetric Gaussian associated with each value of z, showing z_1 is more likely than z_2. The second term is the prior probability $\Pr(z)$. Combining these factors and normalizing so the distribution sums to one gives the posterior $\Pr(z|x^*, \phi)$.

The ELBO (Evidence Lower Bound) is *tight* when, for a fixed value of ϕ, the ELBO equals the likelihood function. To optimize the distribution $q(z|\theta)$, the numerator of the log term in the ELBO is rewritten using conditional probability.

$$
\begin{aligned}
\text{ELBO}(\theta, \phi) &= \int q(z|\theta) \log \left[\frac{\Pr(x, z|\phi)}{q(z|\theta)} \right] dz \\
&= \int q(z|\theta) \log \left[\frac{\Pr(z|x, \phi)\Pr(x|\phi)}{q(z|\theta)} \right] dz \\
&= \int q(z|\theta) \log \Pr(x|\phi) dz + \int q(z|\theta) \log \left[\frac{\Pr(z|x, \phi)}{q(z|\theta)} \right] dz \\
&= \log \Pr(x|\phi) + \int q(z|\theta) \log \left[\frac{\Pr(z|x, \phi)}{q(z|\theta)} \right] dz \quad \textbf{Reciprocal Rule of Logarithms} \\
&= \log \Pr(x|\phi) - D_{\text{KL}} \left[q(z|\theta) \| \Pr(z|x, \phi) \right] \quad \textbf{Second definition:}
\end{aligned}
$$

- The first integral simplifies between the third and fourth lines because $\log \Pr(x|\phi)$ does not depend on z, and the integral of the probability distribution $q(z|\theta)$ equals 1.
- The final line uses the Kullback-Leibler (KL) divergence definition to express the ELBO as the log likelihood minus the KL divergence between $q(z|\theta)$ and $\Pr(z|x, \phi)$.

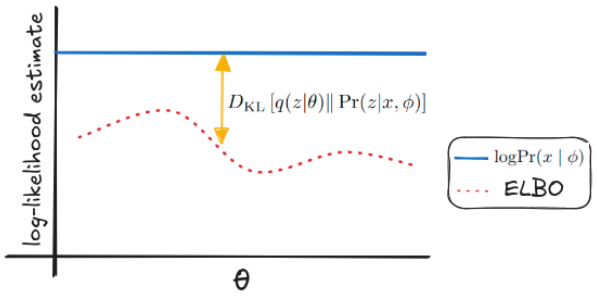


Figure 14: Tightness of bound

This demonstrates that the ELBO is a lower bound on $\log\Pr(x|\phi)$. The KL divergence measures the "distance" between distributions, is always non-negative, and becomes zero when $q(z|\theta) = \Pr(z|x,\phi)$. When the KL divergence is zero, the bound is tight, and the ELBO matches the log likelihood.

$$\mathrm{ELBO}(\theta,\phi) = \log\Pr(x|\phi)$$

$\Pr(z|x,\phi)$ is the posterior distribution over the latent variables z given observed data x; it indicates which values of the latent variable could have been responsible for the data point.

ELBO as reconstruction loss minus KL distance to prior

A third way to present ELBO is to consider the bound as reconstruction error minus the distance to the prior:

$$\begin{aligned}
\mathrm{ELBO}(\theta,\phi) &= \int q(z|\theta)\log\left[\frac{\Pr(x,z|\phi)}{q(z|\theta)}\right]dz \\
&= \int q(z|\theta)\log\left[\frac{\Pr(x|\phi)\Pr(z)}{q(z|\theta)}\right]dz \\
&= \int q(z|\theta)\log\Pr(x|\phi)dz + \int q(z|\theta)\log\left[\frac{\Pr(z)}{q(z|\theta)}\right]dz \\
&= \int q(z|\theta)\log\Pr(x|\phi)dz - D_{\mathrm{KL}}\left[q(z|\theta)\|\Pr(z)\right] \\
&= \int q(z|\theta)\log\Pr(x|\phi)dz - D_{\mathrm{KL}}\left[q(z|\theta)\|\Pr(z)\right] \quad \textbf{Third definition:}
\end{aligned}$$

where the joint distribution $\Pr(x,z|\phi)$ has been factored into conditional probability $\Pr(x|z,\phi)\Pr(z)$ between the first and second lines, and the definition of KL divergence is used again in the last line. In this formulation, the first term measures the average agreement $\Pr(x|z,\phi)$ of the latent variable and the data. This measures the reconstruction accuracy. The second term measures the degree to which the auxiliary distribution $q(z|\theta)$ matches the prior. This formulation is the one that is used in the variational autoencoder.

Missing Pixels Example

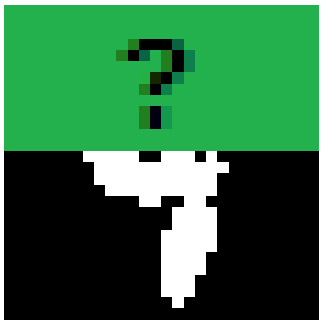


Figure 15: Image with missing pixels.

As we saw earlier, estimating the probability $p(\mathbf{X} = \bar{\mathbf{x}}; \phi)$ of observing a training data point $\bar{\mathbf{x}}$ involves considering all possible ways to complete the image, such as filling in the missing parts (e.g., the green part). Due to the vastness of the search space, it is impractical to compute this directly.

The ELBO becomes tight when $q(z|\theta)$ equals the posterior $\Pr(z|x, \phi)$ (as per the **Second definition**). The posterior $\Pr(z|x, \phi)$ is ideally computed using Bayes' rule:

$$\Pr(z|x, \phi) = \frac{\Pr(x, z|\phi)\Pr(z)}{\Pr(x|\phi)}.$$

However, this computation is often intractable due to the difficulty in evaluating the evidence term $\Pr(x|\phi)$ in the denominator.

To circumvent this issue, variational approximation is employed. This approach involves selecting a simple parametric form for $q(z|\theta)$ to approximate the true posterior ($\Pr(z|x, \phi)$). Typically, a multivariate normal distribution with mean μ and diagonal covariance Σ is used. While this approximation may not perfectly match the posterior, it is effective under certain conditions.

During training, the goal is to minimize the KL divergence between this approximate distribution and the true posterior $\Pr(z|x, \phi)$. This is achieved by optimizing the variational parameters θ, which depend on the observed data x. The variational approximation is defined as:

$$q(z|x, \theta) = \text{Norm}_z\left[g_\mu[x, \theta], g_\Sigma[x, \theta]\right],$$

where $g_\mu[x, \theta]$ and $g_\Sigma[x, \theta]$ are the outputs of a neural network predicting the mean and variance of the normal distribution.

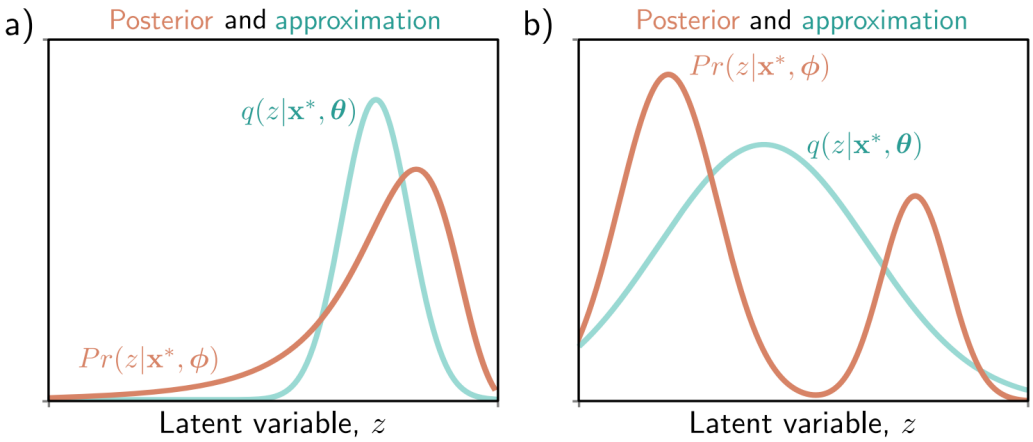


Figure 16: Variational approximation: The posterior $\Pr(z|x^*, \phi)$ cannot be computed in closed form, so a variational approximation selects a family of distributions $q(z|x, \theta)$ (e.g., Gaussians) and finds the closest member of this family to the true posterior; (a) the approximation (cyan curve) may closely match the true posterior (orange curve), but (b) if the posterior is multi-modal, the Gaussian approximation may perform poorly.

In the missing pixels example where pixel values are assumed to be binary (0 or 1), we can simplify the model for illustrative purposes:

$$q(z;\theta) = \prod_{\text{missing pixels } z_i} \theta_i^{z_i}(1-\theta_i)^{(1-z_i)}.$$

This approach involves determining the best θ_i for each missing pixel based on observed evidence and prior knowledge of image structures.

- Is $\theta_i = 0.5\,\forall i$ a good approximation to the posterior $p(z|x;\phi)$? **No** - This is just a random flip.
- Is $\theta_i = 1\,\forall i$ a good approximation to the posterior $p(z|x;\phi)$? **No**.
- Is $\theta_i \approx 1$ for pixels i corresponding to the top part of digit 9 a good approximation? **Yes**.

Note: The quality of this approximation can vary significantly based on the observed data x. If the data representation $p(z, x;\phi)$ deviates substantially from the true data distribution $p_{\text{data}}(z, x)$, such as at the initial stages of learning, the effectiveness of the approximation may be limited.

Reformatting VAE

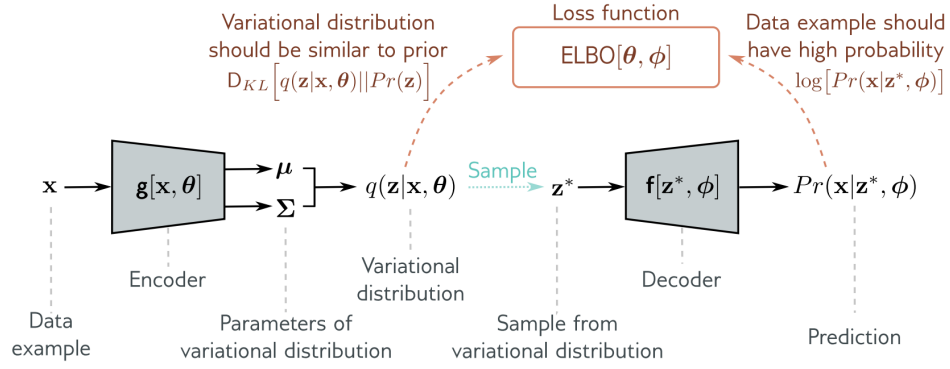


Figure 17: Variational autoencoder: The encoder $g[x, \theta]$ takes a training example x and predicts the parameters μ, Σ of the variational distribution $q(z|x, \theta)$; we sample from this distribution and use the decoder $f[z, \phi]$ to predict x, with the loss function being the negative ELBO, which measures the accuracy of this prediction and the similarity of $q(z|x, \theta)$ to the prior $\Pr(z)$.

Finally, we can describe the VAE. We build a network that computes the ELBO:

$$\text{ELBO}(\theta, \phi) = \int q(z|x, \theta) \log \Pr(x|z, \phi) dz - D_{\text{KL}} \left[q(z|x, \theta) \| \Pr(z) \right],$$

The first term in the ELBO involves an integral that is difficult to compute directly. However, since this term is an expectation over the distribution $q(z|x, \theta)$, we can approximate it using sampling.

$$\mathbb{E}_{q(z|x,\theta)} \left[\log \Pr(x|z, \phi) \right].$$

To avoid computing the integral, we approximate it using samples drawn from $q(z|x, \theta)$. Specifically, we replace the integral with the average of N samples:

$$\mathbb{E}_{q(z|x,\theta)}[a[z]] \approx \frac{1}{N} \sum_{n=1}^{N} \log \Pr(x|z_n^*, \phi).$$

where z_n^* is a sample drawn from $q(z|x, \theta)$.

The second term is the KL divergence between the variational distribution $q(z|x, \theta) = \text{Norm}_z[\mu, \Sigma]$ and the prior $\Pr(z) = \text{Norm}_z[0, I]$. The KL divergence between two normal distributions can be calculated in closed form. For the special case where one distribution has parameters μ, Σ and the other is a standard normal, it is given by:

$$D_{\text{KL}} \left[q(z|x, \theta) \| \Pr(z) \right] = \frac{1}{2} \left(\text{Tr}[\Sigma] + \mu^T \mu - D_z - \log \det[\Sigma] \right),$$

where D_z is the dimensionality of the latent space.

Training

To compute the ELBO, we:

- compute the mean μ and variance Σ of the variational posterior distribution $q(z|\theta, x)$ for this data point x using the network $g[x, \theta]$,
- draw a sample z^* from this distribution, and
- compute the ELBO.
- Back-propagate and update variables ϕ and θ

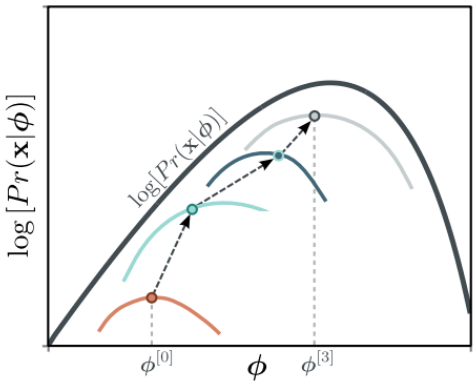


Figure 18: The VAE updates both factors that determine the lower bound at each iteration. Both the parameters ϕ of the decoder and the parameters θ of the encoder are manipulated to increase this lower bound.

Learning process in VAE

To optimize the model, we define the evidence lower bound (ELBO), which is a lower bound on the log-likelihood of the observed data $\log \Pr(x;\phi)$. The ELBO helps to compute this log-likelihood in a tractable way, as directly evaluating $p(z|x;\phi)$ is intractable due to the latent variables z.

The ELBO is defined as:

$$\log \Pr(x;\phi) \geq \int q(z|\theta) \log \left[\frac{\Pr(x,z|\phi)}{q(z|\theta)} \right] dz$$

$$\log \Pr(x;\phi) \geq \sum_z q(z;\theta) \log \Pr(z,x;\phi) - \sum_z q(z;\theta) \log q(z;\theta) = \mathcal{L}(x;\phi,\theta),$$

where:

$$\mathcal{L}(x^i;\phi,\theta^i) = \sum_z q(z;\theta^i) \log \Pr(z,x^i;\phi) + H(q(z;\theta^i)),$$
$$= \mathbb{E}_{q(z;\theta^i)}[\log \Pr(z,x^i;\phi) - \log q(z;\theta^i)].$$

We seek to maximize the expected log-likelihood over the dataset $\mathcal{D}$ using the ELBO as a surrogate function:

$$\ell(\phi;\mathcal{D}) = \sum_{x^i \in \mathcal{D}} \log \Pr(x^i;\phi) \geq \sum_{x^i \in \mathcal{D}} \mathcal{L}(x^i;\phi,\theta^i).$$

Thus, the optimization problem becomes:

$$\max_\phi L(\phi, D) \geq \max_{\phi,\theta^1,\ldots,\theta^M} \sum_{x^i \in \mathcal{D}} \mathcal{L}(x^i;\phi,\theta^i),$$

where M is the number of data points in $\mathcal{D}$ - We have one θ^i per data points which is not scalable in reality.

Stochastic variational inference (SVI):

Stochastic variational inference (SVI) optimizes the ELBO for each data point x^i.

Optimize $\sum_{x^i \in \mathcal{D}} \mathcal{L}(x^i;\phi,\theta^i)$ as a function of $\phi, \theta^1, \ldots, \theta^M$ using (stochastic) gradient descent.

$$\mathcal{L}(x^i;\phi,\theta^i) = \sum_z q(z;\theta^i) \log p(z,x^i;\phi) + H(q(z;\theta^i))$$

$$= \mathbb{E}_{q(z;\theta^i)} \left[\log p(z,x^i;\phi) - \log q(z;\theta^i) \right]$$

The steps are:

1. Initialize $\phi, \theta^1, \ldots, \theta^M$.
2. Randomly sample a data point x^i from $\mathcal{D}$.
3. To *evaluate* the bound, sample $\mathbf{z}^1, \ldots, \mathbf{z}^K$ from $q(z;\theta)$ and estimate:

$$\mathbb{E}_{q(z;\theta^i)} \left[\log p(z,\mathbf{x}^i;\phi) - \log q(z;\theta^i) \right] \approx \frac{1}{K} \sum_k \left(\log p(\mathbf{z}^k,\mathbf{x}^i;\phi) - \log q(\mathbf{z}^k;\theta^i) \right).$$

3. Optimize $\mathcal{L}(x^i;\phi,\theta^i)$ as a function of θ^i:

1. Repeat $\theta^i = \theta^i + \eta \nabla_{\theta^i} \mathcal{L}(x^i; \phi, \theta^i)$.
2. Until convergence to $\theta^{i,*} \approx \arg\max_\theta \mathcal{L}(x^i; \phi, \theta^i)$.

4. Compute $\nabla_\phi \mathcal{L}(x^i; \phi, \theta^{i,*})$.
5. **Update** ϕ using the gradient:

$$\begin{aligned}
&\nabla_\phi \mathcal{L}(x^i; \phi, \theta^{i,*}) \\
&= \nabla_\phi \mathbb{E}_{q(z,\theta)}[\log p(z, x; \phi) - \log q(z; \theta^i)] \\
&= \mathbb{E}_{q(z,\theta^i)}[\nabla_\phi \log p(z, x; \phi)] \\
&\approx \frac{1}{k} \sum_k \nabla_\phi \log p(z^k, x; \phi)
\end{aligned}$$

However, calculation of the gradient of θ can indeed be challenging due to the dependency of the distribution $q(z; \theta)$ on the parameters θ. The difficulty arises because you often need to differentiate through a stochastic node, which the standard backpropagation algorithm does not directly allow.

The reparameterization trick - continuous z

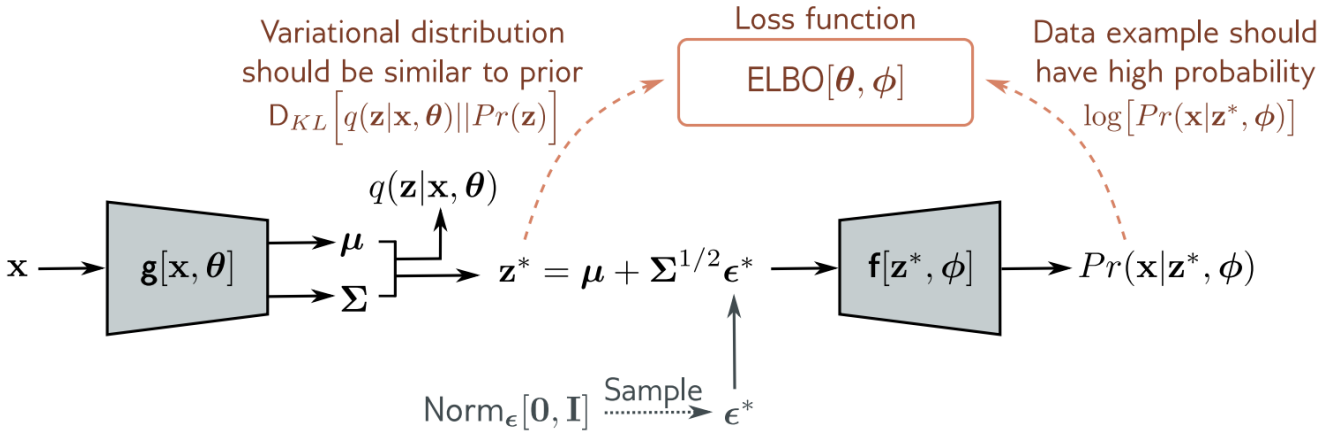


Figure 19: Reparameterization trick. With the original architecture, we cannot easily backpropagate through the sampling step. The reparameterization trick removes the sampling step from the main pipeline; we draw from a standard normal and combine this with the predicted mean and covariance to get a sample from the variational distribution.

The reparameterization trick enables backpropagation through the sampling step by re-expressing the latent variable z as:

$$z^* = \mu + \Sigma^{1/2}\epsilon^*, \quad \epsilon^* \sim \mathcal{N}[0, I]$$

This separates the stochastic sampling from the deterministic network, enabling gradient computation.

Goal: Gradient with Respect to Parameters θ

In generative models, we often need to compute the expectation of a function $r(z)$ of a random variable z sampled from a probability distribution $q(z; \theta)$, and then differentiate it with respect to the model parameters θ. The goal is to compute the gradient of the expected value with respect to θ, which is:

$$E_{q(z;\theta)}[\log p(z, x; \phi) - \log q(z; \theta)] = \int q(z; \theta) \log p(z, x; \phi) - q(z; \theta) \log q(z; \theta) dz$$

$$E_{q(z;\theta)}[r(z)] = \int q(z;\theta) r(z) dz$$

To handle the gradient computation efficiently, we reparameterize z as:

- Sample $\epsilon \sim \mathcal{N}(0, I)$
- Shift and rescale $z = g(\epsilon; \theta) = \mu + \sigma\epsilon$
- In general r is an aribitrary function but in VAE we set: $r(z, \theta, \phi) = \log p(z, x; \phi) - \log q(z; \theta)$

In this case, ϵ is sampled from a standard normal distribution, and the function $g(\epsilon; \theta)$ is a deterministic function independent of θ. This transformation allows us to rewrite the expectation in two ways:

Direct Sampling

$$E_{z\sim q(z;\theta)}[r(z)] = \int r(z) q(z;\theta) dz$$

Reparameterized Sampling

$$E_{z\sim q(z;\theta)}[r(z)] = E_{\epsilon\sim\mathcal{N}(0,I)}[r(g(\epsilon;\theta),\theta,\phi)] = \int r(\mu + \sigma\epsilon, \theta, \phi) \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\epsilon^2}{2}\right) d\epsilon$$

Gradient Computation with Reparameterization

The gradient computation is reformulated as:

$$\underbrace{\nabla_\theta \mathbb{E}_{q(z;\theta)}[r(z,\theta,\phi)]}_{\text{direct sampling}} = \underbrace{\nabla_\theta \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[r(g(\epsilon;\theta),\theta,\phi)]}_{\text{Reparametrized sampling}} = \mathbb{E}_\epsilon[\nabla_\theta r(g(\epsilon;\theta),\theta,\phi)]$$

This formulation allows the gradient to be computed by backpropagation through the deterministic function $g(\epsilon; \theta)$, which represents the transformed random variable z.

Monte Carlo Estimation

For the differentiable functions r and g with respect to θ, Monte Carlo estimation can be applied to approximate the expected value of the reparameterized function. During model training, the gradient of the expectation is estimated as follows:

$$\mathbb{E}_q[\nabla_\theta r(g(\epsilon;\theta),\theta,\phi)] \approx \frac{1}{k}\sum_{i=1}^{k} \nabla_\theta r(g(\epsilon^{(i)};\theta),\theta,\phi),$$

where $\epsilon^{(i)} \sim \mathcal{N}(0, I)$ are independent noise samples. This approach significantly reduces the variance of the gradient estimates compared to direct sampling methods like REINFORCE, enhancing the stability and efficiency of learning.

Furthermore, the expectation $\mathbb{E}_{q(z;\theta)}[r(z, \theta, \phi)]$ for the reparameterized distribution is approximated by:

$$\mathbb{E}_{q(z;\theta)}[r(z,\theta,\phi)] = \mathbb{E}_{\epsilon}[r(g(\epsilon;\theta),\theta,\phi)] \approx \frac{1}{k}\sum_{i=1}^{k} r(g(\epsilon^{(i)};\theta),\theta,\phi)$$

which is:

$$\mathcal{L}(x^i;\phi,\theta^i) = \underbrace{\mathbb{E}_{q(z;\theta)}[\log p(z,x;\phi) - \log q(z;\theta)]}_{\text{direct sampling}} = \underbrace{\mathbb{E}_{\epsilon}[r(g(\epsilon;\theta),\theta,\phi)]}_{\text{Reparametrized sampling}}$$

Now we can solve for:

$$\max_{\phi} \ell(\phi;\mathcal{D}) = \max_{\phi,\theta^1,\ldots,\theta^M} \sum_{x^i\in\mathcal{D}} \mathcal{L}(x^i;\phi,\theta^i),$$

where M represents the number of data points in $\mathcal{D}$. This maximization ensures the best approximation of the model parameters to the data distribution.

The challenge here is that we have a set of variational parameters θ_i for ech data point.

Amortization in Variational Autoencoders (VAE)

In the context of Variational Autoencoders (VAE), *amortization* refers to the process of using a single parametric function, typically a neural network, to approximate the posterior distribution of latent variables across all data points.

Key Concepts

- **Latent Variables and Posterior Inference:** The goal is to approximate the posterior distribution $q(z|x)$ of the latent variable z given an observation x. Direct inference can be computationally expensive.
- **Amortization Process:** Instead of computing the posterior for each data point individually, we learn a single neural network f_λ (Encoder) that maps each input x_i to a set of variational parameters. This is like performing regression, where the input x_i is mapped to variational parameters $\theta^{i,*}$.
  - **Example:** If $q(z|x_i)$ is a Gaussian with different means $\mu_1, \mu_2, \ldots, \mu_m$, we learn a single network f_λ maps x_i to μ_i, thus providing a probabilistic mapping from x to the mean of the approximate posterior distribution.
- We approximate the posterior $q(z|x^i)$ using this distribution $q_\lambda(z|x)$ (Encoder).

Benefits

Amortization significantly reduces the cost of posterior inference by allowing a single forward pass through the encoder network to obtain the variational parameters for all data points, making the VAE model more efficient and scalable for large datasets.

A Variational Approximation to the Posterior

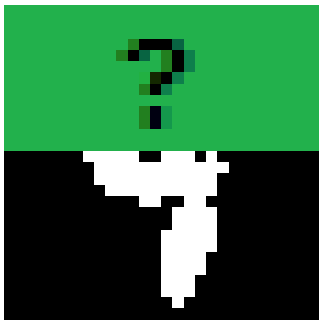


Figure 20: Image with missing pixels.

- Assume $p(z, x^i; \theta)$ is close to $p_{\text{data}}(z, x^i)$. Suppose z captures information such as the digit identity (label), style, etc.
- Suppose $q(z; \theta^i)$ is a (tractable) probability distribution over the hidden variables z parameterized by θ^i.
- For each x^i, we need to find a good $\theta^{i,*}$ (via optimization, which can be expensive).
- **Amortized inference**: learn how to map x^i to a good set of parameters θ^i via $q(z; f_\lambda(x^i))$.

Learning with Amortized Inference

- **Optimize** $\sum_{x^i \in \mathcal{D}} \mathcal{L}(x^i; \phi, \theta)$ as a function of ϕ, θ using (stochastic) gradient descent:

$$\mathcal{L}(x;\, \phi, \theta) \;=\; \sum_z \underbrace{q(z; f_\lambda(x^i))}_{encoder} \underbrace{\log p(z, x;\, \phi)}_{decoder} + H\Big(\underbrace{q(z; f_\lambda(x^i))}_{encoder}\Big)$$

$$= \mathbb{E}_{q(z; f_\lambda(x^i))}\Big[\log p(z, x;\, \phi) \;-\; \log q(z; f_\lambda(x^i))\Big].$$

1. Initialize $\phi^{(0)}, \theta^{(0)}$.
2. Randomly sample a data point x^i from $\mathcal{D}$.
3. Compute $\nabla_\phi \mathcal{L}(x^i; \phi, \theta)$ and $\nabla_\theta \mathcal{L}(x^i; \phi, \theta)$.
4. Update ϕ, θ in the gradient direction. - This is what we use in practice, jointly optimize for parameters

References

[1] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

[2] Stefano Ermon. *VAE*. CS236.