Variational Auto Encoders

1 Introduction

∗ Def. **1.0.1 Statistical Inference:** Determining properties of the underlying probability distribution based on observed data. This could include estimating parameters, latent variables, or making predictions. In Bayesian inference, the primary goal is to compute the posterior $p(z|x)$:

$$p(z|x) = \frac{p(x,z)}{p(x)} = \frac{p(x|z)p(z)}{\int_{\mathcal{Z}} p(x,z)\,dz}$$

Since the marginal likelihood $p(x)$ is intractable, we instead approximate the posterior $p(z|x)$ with a simpler, tractable distribution $q(z|\lambda)$, and we wish to minimize the KL divergence $KL\Big(q(z|\lambda)\,||\,p(z|x)\Big)$, or equivalently maximize the Evidence Lower Bound (ELBO):

∗ Def. **1.0.2 Density Estimation:** Approximating PDF $p(x)$ of a random variable X based on observed data $\{x_1, \ldots, x_n\}$ drawn i.i.d from the underlying distribution. Can be either parametric, e.g.

$$p(x|\theta) \sim \mathcal{N}(\mu_x, \sigma_x)$$

or nonparametric density estimation, for example Kernel Density Estimation:

$$p(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

∗ Def. **1.0.3 Modeling:** Modeling is unveiling the underlying ruling processes, by posing hypotheses and predictions, based on observations. For instance, physicists model how fluids flow, and biologists the structure of organisms.

Modeling often involves representation, where we describe a phenomena using specific qualities and quantities related to the process we are interested in. We describe an object by its shape, color, position, volume, etc. When looking at data - a large collection of samples - in some cases, it is reasonable to believe that those representations follow some distributions. For example, human height clearly follow some probability distribution. In such cases, we can think of the samples as being "generated" from those distributions. When such a hypothesis is true, we can generate new samples in the population, provided we estimated its probabilities. Alternatively, we may say that we have uncertainty about those variables, and we specify the degree and nature of this uncertainty in terms of probability distributions.

Remark (): A complete probabilistic model captures both the distributions of its components and the relations/dependencies between them. Usually the linear relations, given by the covariances.

∗ Def. **1.0.4 Probabilistic Model:** Assume the observed variables $\boldsymbol{x}$ are a random sample from an unknown underlying process, whose true probability distribution is $p^*(\boldsymbol{x})$. We approximate this underlying process with a chosen probabilistic model $p_\theta(\boldsymbol{x})$, with parameters $\boldsymbol{\theta}$:

$$\boldsymbol{x} \sim p_\theta(\boldsymbol{x})$$

Learning is the process of searching for a value of the parameters θ, such that the probability function given by the mode $p_\theta(\boldsymbol{x})$, approximates the true distribution of the data, denoted by $p^*(\boldsymbol{x})$, such that for any observed $\boldsymbol{x}$:

$$p_\theta(\boldsymbol{x}) \approx p^\star(\boldsymbol{x})$$

∗ Def. **1.0.5 Conditional Models:** Often, we are not interested in learning an unconditional model $p_\theta(\boldsymbol{x})$, but a conditional model $p_\theta(\boldsymbol{y}|\boldsymbol{x})$, that approximates the underlying conditional distribution $p^*(\boldsymbol{y}|\boldsymbol{x})$: A distribution over the values of variable $\boldsymbol{y}$, conditioned on the value of an observed variable $\boldsymbol{x}$. $\boldsymbol{x}$ is often called the *input* of the model.

A common example is image classification, where $\boldsymbol{x}$ is the image, and $\boldsymbol{y}$ is the label, and $p_\theta(\boldsymbol{y}|\boldsymbol{x})$ is chosen to be the categorical distribution, whose parameters are computed from $\boldsymbol{x}$.

Remark (): We can use neural networks to parameterize a distribution. For example for the categorical distribution Categorical$(y; \boldsymbol{p})$ over a class label y, we have:

$$\boldsymbol{p} = \text{NeuralNet}(\boldsymbol{x})$$
$$p_\theta(y|\boldsymbol{x}) = \text{Categorical}(y; \boldsymbol{p})$$

1.1 Directed PGMs and NNs

We work with *directed* PGMs, where all variables are organized into a directed acyclic graph, and the joint distribution over the variables factorizes as a product of prior and conditional probabilities:

$$p_\theta(\boldsymbol{x}_1, \dots \boldsymbol{x}_m) = \sum_{j=1}^{m} p_\theta(\boldsymbol{x}_j | Pa(\boldsymbol{x}_j))$$

where $Pa(\boldsymbol{x}_j)$ is the set of parent variables of node j in the directed graph.

Traditionally, each conditional probability distribution $p_\theta(\boldsymbol{x}_j|Pa(\boldsymbol{x}_j))$ is parameterized by a lookup table or a linear model. A more flexible way is to parametrize such conditional distributions using a neural network. In this case, the NN takes as input the parents of a variable, and produce the distributional parameters $\boldsymbol{\eta}$ over that variable:

$$\boldsymbol{\eta} = \text{NeuralNet}(Pa(\boldsymbol{x}))$$
$$p_\theta(\boldsymbol{x}|Pa(\boldsymbol{x})) = p_\theta(\boldsymbol{x}|\boldsymbol{\eta})$$

for example, in VAE, when learning $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, note that $Pa(\boldsymbol{x}) = \boldsymbol{z}$, and the recognition model learn the distributional parameters $\boldsymbol{\eta} = (\mu, \sigma)$ to parametrize $p_\theta(\boldsymbol{x}|\boldsymbol{\eta})$

1.2 Learning in Fully Observed Models with Neural Nets

If all variables in the directed model are observed in the data, then we can compute and differentiate the log-probability of the data under the model, leading to a relatively straightforward optimization.

Dataset: We collect a dataset $\mathcal{D}$ consisting of N datapoints:

$$\mathcal{D} = \left\{ \boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N)} \right\}$$

The datapoints are assumed to be independent samples from an unchanging underlying distribution, i.e. i.i.d. Then the probability of the datapoints given the parameters factorizes as a product of individual datapoint probabilities:

$$\log p_\theta(\mathcal{D}) = \sum_{\boldsymbol{x} \in \mathcal{D}} \log p_\theta(\boldsymbol{x}) \qquad \text{(Maximum Likelihood)}$$

A common criterion for probabilistic models is maximum log-likelihood (MLL). We attempt to find the parameters θ that maximize the above sum, or equivalently, the average, of the log-probabilities assigned to the data by the model. Since our model is limited by its expressiveness, and our data is noisy, this can not be trivially solved.

Using stochastic gradient descent, we draw a minibatch $\mathcal{M}$ of the data, and since:

$$\nabla_\theta \log p_\theta(\mathcal{D}) \simeq \nabla_\theta \log p_\theta(\mathcal{M}) = \sum_{\boldsymbol{x} \in \mathcal{M}} \nabla_\theta \log p_\theta(\boldsymbol{x})$$

where $\simeq$ indicates an unbiased estimator, we can use $\sum_{\boldsymbol{x} \in \mathcal{M}} \nabla_\theta \log p_\theta(\boldsymbol{x})$ to iteratively update θ and hill-climb to a local optimum of the Maximum Likelihood.

1.3 Learning and Inference in Deep Latent Variable Models

$*$ Def. **1.3.1 Latent Variables:** Variables which are part of the model, but which we do not observe.

For an observed variable $\boldsymbol{x}$ and an unobserved variable $\boldsymbol{z}$, the joint distribution is denoted $p_\theta(\boldsymbol{x}, \boldsymbol{z})$ over both variables. The marginal distribution over the observed variables $p_\theta(\boldsymbol{x})$ is given by:

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) \, d\boldsymbol{z} \tag{1}$$

This is also called the *evidence*.

$*$ Def. **1.3.2 Deep Latent Variable Model (DLVM):** latent variable model $p_\theta(\boldsymbol{x}, \boldsymbol{z})$ whose distributions are parameterized by neural nets. Then, even when each factor (prior or conditional distribution) in the directed model is relatively simple (such as a conditional Gaussian), the marginal distribution $p_\theta(\boldsymbol{x})$ can be very complex. This expressivity makes DLVM attractive for approximating complicated underlying distributions $p^\star(\boldsymbol{x})$.

@ Example: A simple DLVM is:

$$p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p_\theta(\boldsymbol{z})p_\theta(\boldsymbol{x}|\boldsymbol{z})$$

we call $p_\theta(\boldsymbol{z})$ the *prior distribution* over $\boldsymbol{z}$.

For example, assume, for binary data $\boldsymbol{x}$ (such as 0-1 MNIST images), a spherical Gaussian latent space, and a factorized Bernoulli observation model. Then:

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; 0, I)$$
$$\boldsymbol{p} = \text{DecoderNeuralNet}_\theta(\boldsymbol{z})$$
$$\log p(\boldsymbol{x}|\boldsymbol{z}) = \sum_{j=1}^{D} \log p(x_j|\boldsymbol{z}) = \sum_{j=1}^{D} \log \text{Bernoulli}(x_j; p_j)$$

where D is the dimensionality of $\boldsymbol{x}$.

Remark (): As $\boldsymbol{z}$ is not observed, computing Eq. (1) is intractable, and hence we cannot directly optimize it.

For a Fully observed model, like $p_\theta(\boldsymbol{x}) \sim \mathcal{N}(\mu, \sigma^2)$, we can easily compute $\log p_\theta(\boldsymbol{x})$, and optimize θ using the gradients of Maximum Likelihood.

For latent variable models, we have:

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}, \boldsymbol{z})\, d\boldsymbol{z}$$

The integral over $\boldsymbol{z}$ often lacks a closed-form solution, due to the complexity of taking its gradient, and the dimensionality of $\boldsymbol{z}$.

Note - we often do know $p_\theta(\boldsymbol{x}, \boldsymbol{z})$, the joint distribution, since we have $p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p_\theta(\boldsymbol{x}|z)p_\theta(\boldsymbol{z})$, and we often assume $p_\theta(\boldsymbol{z})$, and $p_\theta(\boldsymbol{x}|z)$, the *likelihood*

The intractability of $p_\theta(\boldsymbol{x})$ is related to the intractability of the *posterior* distribution $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, via the identity:

$$p_\theta(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{p_\theta(\boldsymbol{x})}$$

since $p_\theta(\boldsymbol{x}, \boldsymbol{z})$ is tractable to compute. We approximate $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ using variational inference, by a similar distribution $q\theta(\boldsymbol{z}|\boldsymbol{x})$, called the **variational distribution**.

2 Variational Auto Encoders

Remark (Motivation): Suppose we have observed data $\mathbf{x}$, for instance, pixels of dog images. These data points are drawn from some true but unknown distribution $p^*(\mathbf{x})$. Our goal is to approximate $p^*(\boldsymbol{x})$ using a parameterized family $p_\theta(\mathbf{x})$, enabling tasks like generation (sampling new images) or downstream inference (classification, etc.).

Directly modeling $p_\theta(\mathbf{x})$ can be extremely challenging, especially for complex, high-dimensional data. One common approach is to introduce latent variables $\mathbf{z}$ that describe underlying factors or structures, and then express the data distribution through marginalization:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x} \mid \mathbf{z})p_\theta(\mathbf{z})\, d\mathbf{z}.$$

Here, $p_\theta(\mathbf{x} \mid \mathbf{z})$ is often simpler to model than $p_\theta(\mathbf{x})$ itself. For example, $\mathbf{z}$ might represent attributes like a dog's breed, pose, or lighting, making it easier to build a model that generates images $\mathbf{x}$ given $\mathbf{z}$. We also choose a prior $p_\theta(\mathbf{z})$ that is tractable, such as a Gaussian.

The challenge is that computing $p_\theta(\mathbf{x})$ requires integrating over $\mathbf{z}$, which is typically intractable for complex models. This makes it hard to directly optimize parameters θ by maximum likelihood.

Role of the Posterior and Variational Inference: Remember our goal is to find parameters θ that will optimize $\log p_\theta(\boldsymbol{x})$. Using mathematical tricks, we use posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ - in fact an approximation $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ - and by choosing ϕ such that $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is close to $p_\theta(\boldsymbol{z})$, we increase $\log p_\theta(\boldsymbol{x})$.

Specifically, note that:

$$\log p_\theta(\boldsymbol{x}) = \log \int p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})\, d\boldsymbol{z}$$

We multiply and divide by $q_\phi(\boldsymbol{z}|\boldsymbol{x})$:

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

Recall Jensen's inequality for concave function states that $f(E[X]) \geq E[f(X)]$, and since log is concave, we get that $\log E[X] \geq E[\log X]$. Define $X = \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}$, and we get:

$$\log p_\theta(\boldsymbol{x}) \geq \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \, d\boldsymbol{z}$$

Rewrite the integrand using $p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})$ and we get:

$$\log p_\theta(\boldsymbol{x}) \geq \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \, d\boldsymbol{z}$$

This gives us:

$$\begin{aligned}
\log p_\theta(\boldsymbol{x}) &\geq E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[p_\theta(\boldsymbol{x}|\boldsymbol{z})\big] + E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_\theta(\boldsymbol{z})\big] - E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log q_\phi(\boldsymbol{z}|\boldsymbol{x})\big] \\
&= E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[p_\theta(\boldsymbol{x}|\boldsymbol{z})\big] - \Big[E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log q_\phi(\boldsymbol{z}|\boldsymbol{x})\big] - E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_\theta(\boldsymbol{z})\big]\Big] \\
&= E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[p_\theta(\boldsymbol{x}|\boldsymbol{z})\big] - KL\Big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})\Big)
\end{aligned}$$

with:

$$\log p_\theta(\boldsymbol{x}) = E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left[\frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right]\right] + E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left[\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\right]\right]$$

where we denote the ELBO by:

$$\mathcal{L}_{\theta,\phi}(\boldsymbol{x}) = E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})]$$

and:

$$D_{KL}\big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})\big) = E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left[\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\right]\right]$$

so:

$$\log p_\theta(\boldsymbol{x}) = \mathcal{L}_{\theta,\phi}(\boldsymbol{x}) + D_{KL}\big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})\big)$$

This is the Evidence Lower BOund (ELBO):

$$\log p_\theta(\mathbf{x}) \geq \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z}))}_{\mathrm{ELBO}(\theta,\phi)}.$$

If we fix θ and adjust ϕ, making $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ closer to $p_\theta(\boldsymbol{z})$ we increase $\log p_\theta(\boldsymbol{x})$.

In summary, latent variable models and the introduction of a variational posterior allow us to deal with intractable marginal likelihoods by converting the problem into one of optimizing a tractable lower bound, thereby enabling effective parameter learning even for very complex data like images.

$*$ Def. **2.0.1 VAE:** The VAE can be viewed as two coupled, but independently parameterized models: The encoder or the recognition model,and the generation model (aka decoder). The recognition model provides (an approximation to) the posterior for latent variables conditioned on observed data $p_\theta(z|x)$.

You've hit upon a crucial point of understanding the Variational Autoencoder (VAE) and its optimization objective, the Evidence Lower Bound (ELBO). Your intuition is partially correct, highlighting a subtle but important detail. Let's break it down step-by-step:

1. The Ultimate Goal: Maximizing the Log-Likelihood

Yes, the overarching goal in training a generative model like a VAE is to find the parameters θ of the generative model $p_\theta(\mathbf{x})$ that maximize the likelihood of the observed data $\mathbf{x}$. This means we want to find parameters such that the model assigns a high probability to the data we've seen. Mathematically, we want to maximize $\log p_\theta(\mathbf{x})$.

Why is Directly Maximizing $\log p_\theta(\mathbf{x})$ Difficult?

The problem is that calculating $p_\theta(\mathbf{x})$ directly often involves marginalizing over the latent variable $\mathbf{z}$:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}$$

This integral is often intractable, meaning we can't compute it analytically or efficiently. This intractability is the core motivation behind using the ELBO.

2. The Role of the ELBO: A Tractable Lower Bound

The ELBO, $\mathcal{L}_{\theta,\phi}(\boldsymbol{x})$, is introduced as a *lower bound* on the log-likelihood $\log p_\theta(\boldsymbol{x})$. This is evident from the equation you provided:

$$\log p_\theta(\boldsymbol{x}) = \mathcal{L}_{\theta,\phi}(\boldsymbol{x}) + D_{KL}\big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})\big)$$

Since the KL divergence $D_{KL}\big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})\big)$ is always non-negative, we have:

$$\log p_\theta(\boldsymbol{x}) \geq \mathcal{L}_{\theta,\phi}(\boldsymbol{x})$$

This confirms that the ELBO is indeed a lower bound.

3. What Happens When We Maximize the ELBO?

Now, let's consider what happens when we try to maximize the ELBO with respect to both the generative parameters θ and the inference network parameters ϕ.

* **Maximizing $\mathcal{L}_{\theta,\phi}(\boldsymbol{x})$ pushes the lower bound upwards.** Since $\log p_\theta(\boldsymbol{x}) = \mathcal{L}_{\theta,\phi}(\boldsymbol{x}) + D_{KL}\big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})\big)$, increasing the lower bound means we are either increasing the log-likelihood itself or decreasing the KL divergence (or both).

* **The interplay between the terms:** It's crucial to understand that we are optimizing the ELBO with respect to *both* θ and ϕ. * **Optimizing with respect to ϕ (Encoder):** By optimizing ϕ, we are trying to find an approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ that is close to the true (but intractable) posterior $p_\theta(\mathbf{z}|\mathbf{x})$. This directly minimizes the KL divergence term $D_{KL}\big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})\big)$. * **Optimizing with respect to θ (Decoder):** By optimizing θ, we are trying to improve the generative model $p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})$. This directly affects the term $E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}, \boldsymbol{z})]$ in the ELBO.

Your Intuition and the Key Insight

You are correct that if we were *only* minimizing the KL divergence without considering the other term in the ELBO, we could potentially achieve a small KL divergence without actually improving the generative model $p_\theta(\mathbf{x})$.

However, the optimization process of the ELBO simultaneously tries to:

1. **Reconstruct the data well:** The term $E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}, \boldsymbol{z})] = E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\mathbf{z}) + \log p_\theta(\mathbf{z})]$ encourages the decoder $p_\theta(\mathbf{x}|\mathbf{z})$ to reconstruct the input $\mathbf{x}$ accurately from the latent representation $\mathbf{z}$ sampled from the encoder $q_\phi(\mathbf{z}|\mathbf{x})$.

2. **Make the approximate posterior close to the prior:** The term $-D_{KL}\big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})\big)$ (after expanding the ELBO) encourages the encoder's distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to be close to the prior distribution $p_\theta(\mathbf{z})$ over the latent

space. This regularizes the latent space and prevents overfitting. *Note: The KL divergence in your equation involves the true posterior, which is intractable. In practice, the ELBO is often rewritten using the prior $p_\theta(\mathbf{z})$ to make it tractable.*

Why Maximizing the ELBO Helps Maximize the Log-Likelihood

By maximizing the ELBO with respect to both θ and ϕ, we are effectively "tightening" the lower bound. As we increase the ELBO, since $\log p_\theta(\boldsymbol{x}) = \mathcal{L}_{\theta,\phi}(\boldsymbol{x}) + D_{KL}\big(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})\big)$, and the KL divergence is always non-negative, the log-likelihood itself must also increase (or at least not decrease).

Analogy: Imagine you're trying to reach the ceiling of a room. The ELBO is like a platform you can stand on. By maximizing the ELBO, you're raising the platform. As you raise the platform, you get closer to the ceiling (the log-likelihood). You can't go higher than the ceiling, and the distance between your platform and the ceiling is the KL divergence.

In summary:

* **Our goal is to maximize $\log p_\theta(\mathbf{x})$.** * **Directly maximizing $\log p_\theta(\mathbf{x})$ is often intractable.** * **The ELBO, $\mathcal{L}_{\theta,\phi}(\boldsymbol{x})$, is a tractable lower bound on $\log p_\theta(\mathbf{x})$.** * **Maximizing the ELBO with respect to both θ and ϕ simultaneously encourages good data reconstruction and makes the approximate posterior close to the prior (or true posterior).** * **By maximizing the ELBO, we are effectively pushing the lower bound upwards, which in turn encourages the log-likelihood to increase.** We are not just minimizing the KL divergence in isolation; the optimization process is intertwined.

The Task and How ELBO Helps:

The task is to find the parameters θ of a generative model that can explain the observed data. Maximizing the log-likelihood $\log p_\theta(\mathbf{x})$ directly achieves this. Since this is intractable, we resort to maximizing the ELBO.

Maximizing the ELBO gives us:

1. **A way to approximate the intractable log-likelihood:** While we don't get the exact value of $\log p_\theta(\mathbf{x})$, the ELBO serves as a useful surrogate objective. 2. **A framework for learning meaningful latent representations:** The encoder $q_\phi(\mathbf{z}|\mathbf{x})$ learns to map data points to a latent space, and the KL divergence term encourages this latent space to have desirable properties (e.g., being smooth and continuous). 3. **A balance between reconstruction and regularization:** The two terms in the ELBO work in concert. The reconstruction term ensures the model can generate data similar to the training data, while the KL divergence term prevents overfitting and encourages a well-structured latent space.

Therefore, maximizing the ELBO is not just about minimizing the KL divergence. It's about finding a sweet spot where the generative model explains the data well (high reconstruction likelihood) and the approximate posterior is a reasonable approximation of the true posterior (low KL divergence). This indirectly leads to finding parameters θ that result in a higher (though we can't directly compute it) $\log p_\theta(\mathbf{x})$.

Okay, let's break down how to compute $\log p_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i)})$ during the decoder's forward pass in a VAE.

Mathematical Formulation

Recall the basic factorization of the joint probability:

$$p_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}) = p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})p_\theta(\mathbf{z}^{(i)})$$

Taking the logarithm of both sides, we get:

$$\log p_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}) = \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)}) + \log p_\theta(\mathbf{z}^{(i)})$$

So, to compute the log-joint probability, we need to compute two terms:

1. **The log-likelihood of the data given the latent variable:** $\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})$ 2. **The log-prior probability of the latent variable:** $\log p_\theta(\mathbf{z}^{(i)})$

Let's discuss each term in detail:

1. Computing $\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})$

* **The Decoder's Role:** The decoder network, parameterized by θ, takes the latent variable $\mathbf{z}^{(i)}$ as input and outputs the parameters of a distribution over the data space. The choice of this distribution depends on the nature of your data. Common choices include:

* **Gaussian Distribution:** If your data $\mathbf{x}$ is continuous, you often model $p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})$ as a multivariate Gaussian distribution. The decoder will output the mean $\boldsymbol{\mu}_\theta(\mathbf{z}^{(i)})$ and the covariance matrix $\boldsymbol{\Sigma}_\theta(\mathbf{z}^{(i)})$ (or a related parameterization like the log-standard deviation for a diagonal covariance). Assuming a diagonal covariance for simplicity, where $\boldsymbol{\Sigma}_\theta(\mathbf{z}^{(i)}) = \text{diag}(\sigma^2_{\theta,1}(\mathbf{z}^{(i)}), \ldots, \sigma^2_{\theta,D_x}(\mathbf{z}^{(i)}))$, the log-likelihood is:

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

Expanding the Gaussian log-likelihood:

$$\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)}) = -\frac{D_x}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{D_x}\left(\log \sigma^2_{\theta,j}(\mathbf{z}^{(i)}) + \frac{(x_j^{(i)} - \mu_{\theta,j}(\mathbf{z}^{(i)}))^2}{\sigma^2_{\theta,j}(\mathbf{z}^{(i)})}\right)$$

* **Bernoulli Distribution:** If your data $\mathbf{x}$ consists of binary values (e.g., pixel intensities are 0 or 1), you might model $p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})$ as a product of independent Bernoulli distributions. The decoder will output the probabilities $\mathbf{p}_\theta(\mathbf{z}^{(i)})$ of each dimension being 1. The log-likelihood is:

$$\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)}) = \sum_{j=1}^{D_x}\left[x_j^{(i)}\log(p_{\theta,j}(\mathbf{z}^{(i)})) + (1 - x_j^{(i)})\log(1 - p_{\theta,j}(\mathbf{z}^{(i)}))\right]$$

* **Implementation:** In your code, the decoder network (a neural network) will take $\mathbf{z}^{(i)}$ as input and output the parameters needed for the chosen distribution (e.g., means and log-variances for a Gaussian, or probabilities for a Bernoulli). You then use these parameters and the formula for the log-likelihood of that distribution to compute $\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})$.

2. Computing $\log p_\theta(\mathbf{z}^{(i)})$

* **The Prior Distribution:** The prior distribution over the latent space, $p_\theta(\mathbf{z})$, is typically chosen to be a simple distribution, independent of the data and often fixed. The most common choice is the standard normal distribution:

$$p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$$

where $\mathbf{0}$ is a vector of zeros and $\mathbf{I}$ is the identity matrix. The log-prior probability for a single latent sample $\mathbf{z}^{(i)}$ is:

$$\log p_\theta(\mathbf{z}^{(i)}) = \log \mathcal{N}(\mathbf{z}^{(i)}; \mathbf{0}, \mathbf{I}) = -\frac{D_z}{2} \log(2\pi) - \frac{1}{2} \sum_{k=1}^{D_z} (z_k^{(i)})^2$$

where D_z is the dimensionality of the latent space.

* **Implementation:** Since the prior is fixed (typically a standard normal), this calculation is straightforward. You simply apply the log-likelihood formula of the chosen prior distribution to the sampled latent variable $\mathbf{z}^{(i)}$.

Putting it Together

Once you have computed both $\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})$ and $\log p_\theta(\mathbf{z}^{(i)})$, you can add them together to obtain the log-joint probability:

$$\log p_\theta(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}) = \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)}) + \log p_\theta(\mathbf{z}^{(i)})$$

3 Tutorial on VAE / Doresch

Assume we have a dataset D, containing images X coming from some high-dimensional space $\mathcal{X}$. We wish to find $P(X)$. However, in practice, $P(X)$ is both intractable, too large to represent, and too large to sample. Even if we knew $P(X)$, we could not generate new samples from it. We make the correct assumption that the generation process is actually the following: first we choose a *latent* vector $\boldsymbol{z}$, and then a deterministic, parametric function $f : \mathcal{Z} \times \Theta \to \mathcal{X}$ generates X. We wish to optimize θ, such that when we sample $\boldsymbol{z}$ from $P(\boldsymbol{z})$, and with high probability, $f(\boldsymbol{z}; \theta)$ will be like the X in our dataset. To make this notion precise mathematically, we are aiming at maximizing the probability of each X in the training set under the entire generative process, according to:

$$P(X) = \int_{\mathcal{Z}} P(X|\boldsymbol{z})P(\boldsymbol{z})\, d\boldsymbol{z}$$

We replace the notation $f(z; \theta)$ by $P(X|z)$, to make the dependence of X on z explicit. We often assume:

$$P(X|\boldsymbol{z}; \theta) = \mathcal{N}(X|f(\boldsymbol{z}; \theta), \sigma^2 * I) \tag{1}$$

This formalizes the notion some $\boldsymbol{z}$ needs to result in samples that are merely *like* X, instead of setting $X = f(\boldsymbol{z}; \theta)$.

Early in training, our model will not produce output that are close to X. By using gradient descent on θ, we can increase P(X) by making $f(\boldsymbol{z}; \theta)$ approach X for some $\boldsymbol{z}$.

There are two problems when trying to optimize Eq. (1):

1. How to choose define the latent variables $\boldsymbol{z}$, i.e. decide what information they decode? The answer is: (multi-variable) Gaussian distribution. This is because, a sufficiently complicated function f can turn a Gaussian into every desired probability distribution, as is demonstrated by Inverse transform sampling. For f that is a NN, we can imagine the first few layers turning the normally distributed $\boldsymbol{z}$'s to the latent vectors (like digit identity, stroke weight, angle, etc.), and then using later layers to map those latent values to a fully-rendered digit.

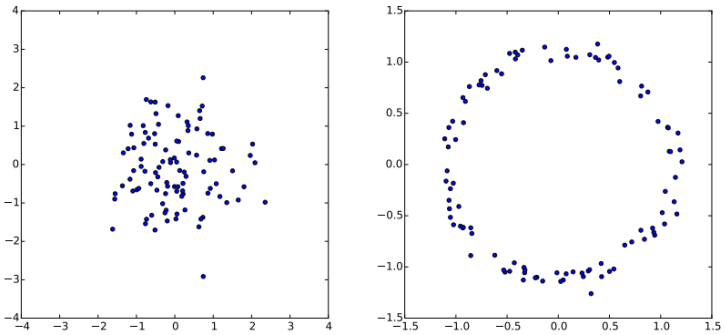

Figure 2: Given a random variable z with one distribution, we can create another random variable $X = g(z)$ with a completely different distribution. Left: samples from a gaussian distribution. Right: those same samples mapped through the function $g(z) = z/10 + z/||z||$ to form a ring. This is the strategy that VAEs use to create arbitrary distributions: the deterministic function g is learned from data.

2. Conceptually, it easy to approximate $P(X)$ - we sample a large value of z values $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$, and compute $P(X) \approx \frac{1}{n} \sum_i P(X|\boldsymbol{z}_i)$. However, for high dimensional spaces, the number of samples of $\boldsymbol{z}$ might need to be very large before we have an accurate estimate of $P(X)$. They include a reason why this is so:

(a) (b) (c)

Figure 3: It's hard to measure the likelihood of images under a model using only sampling. Given an image X (a), the middle sample (b) is much closer in Euclidean distance than the one on the right (c). Because pixel distance is so different from perceptual distance, a sample needs to be extremely close in pixel distance to a datapoint X before it can be considered evidence that X is likely under the model.

So instead, we use the correct assumption, that for most $\boldsymbol{z}$, $P(X|\boldsymbol{z})$ will be nearly zero, and hence contribute almost nothing to our estimate of $P(X)$. Therefore, we attempt to sample values of $\boldsymbol{z}$ that are likely to have produced X, and compute $P(X)$ just from those. This means we need a new function $Q(\boldsymbol{z}|X)$, the "recognition" model, which can take a value of X and give a distribution over $\boldsymbol{z}$ values that are likely to produce X. By assuming that this space of $\boldsymbol{z}$ values that are likely under Q will be much smaller than the space of all $\boldsymbol{z}$ that are likely under the prior $P(\boldsymbol{z})$. This lets us compute $E_{\boldsymbol{z}\sim Q}P(X|\boldsymbol{z})$ relatively easy.

We now formulate our new objective. Having moved to computing $E_{z\sim Q}P(X|\boldsymbol{z})$, we need to relate $P(X)$ to $E_{\boldsymbol{z}\sim Q}P(X|\boldsymbol{z})$. We use some math tricks:

$$\mathrm{KL}\Big[Q(\boldsymbol{z}|X)\,||\,P(\boldsymbol{z}|X)\Big] = E_{\boldsymbol{z}\sim Q}\Big[\log Q(\boldsymbol{z}|X) - \log P(\boldsymbol{z}|X)\Big]$$

Using Bayes rule, we get:

$$\mathrm{KL}\Big[Q(\boldsymbol{z}|X)\,||\,P(\boldsymbol{z}|X)\Big] = E_{\boldsymbol{z}\sim Q}\Big[\log Q(\boldsymbol{z}|X) - \log P(X|\boldsymbol{z}) - \log P(\boldsymbol{z})\Big] + \log P(X)$$

We negate and move sides, and get:

$$\log P(X) - \mathrm{KL}\Big[Q(\boldsymbol{z}|X)\,||\,P(\boldsymbol{z}|X)\Big] = E_{\boldsymbol{z}\sim Q}\Big[\log P(X|\boldsymbol{z})\Big] - \mathrm{KL}\Big[Q(\boldsymbol{z}|X)\,||\,P(\boldsymbol{z})\Big] \tag{2}$$

The LHS is the quantity we wish to optimize, plus an error term for how well Q produces $\boldsymbol{z}$ the can reproduce a given X; This is small for the high-capacity Q. The RHS is something we can optimize via SGD. Recall that we assume that:

$$Q(\boldsymbol{z}|X) = \mathcal{N}\big(\boldsymbol{z}|\mu(X;\phi),\Sigma(X;\phi)\big)$$

Since we also assume Gaussian prior $P(\boldsymbol{z})$, we get the value of $\mathrm{KL}\Big[Q(\boldsymbol{z}|X)\,||\,P(\boldsymbol{z})\Big]$. As For the first term, we take one sample of $\boldsymbol{z}$, and treat $\log P(X|z)$ (which we compute via $f(\boldsymbol{z};\theta)$ as the approximation of $E_{\boldsymbol{z}\sim Q}\big[\log P(X|\boldsymbol{z})\big]$. Recall we are already doing stochastic gradient descent over different values of X from a dataset D. The full equation we want to optimize is:

$$\begin{aligned} E_{X\sim D}\Big[\log P(X) - \mathrm{KL}\Big[Q(\boldsymbol{z}|X)\,||\,P(\boldsymbol{z}|X)\Big]\Big] = \\ E_{X\sim D}\Big[E_{\boldsymbol{z}\sim Q}\big[\log P(X|\boldsymbol{z})\big] - \mathrm{KL}\big[Q(\boldsymbol{z}|X)\,||\,P(\boldsymbol{z})\big]\Big] \end{aligned} \tag{3}$$

Hence we can compute:

$$P(X|\boldsymbol{z}) - \mathrm{KL}\Big[Q(\boldsymbol{z}|X)\,||\,P(\boldsymbol{z})\Big]$$

And we can take its gradient. By averaging the gradient of this function over aribtrarily many samples of X and $\boldsymbol{z}$, the result converges to the gradient of Eq. (3).

However there is still one problem.In Eq. (3), $E_{\boldsymbol{z}\sim Q}\big[\log P(X|\boldsymbol{z})\big]$ depends both on the parameters of P (θ) and Q (ϕ). Doing back propagation on the (left) network below is impossible, since sampling is a non-continuous operation, and has no gradient.

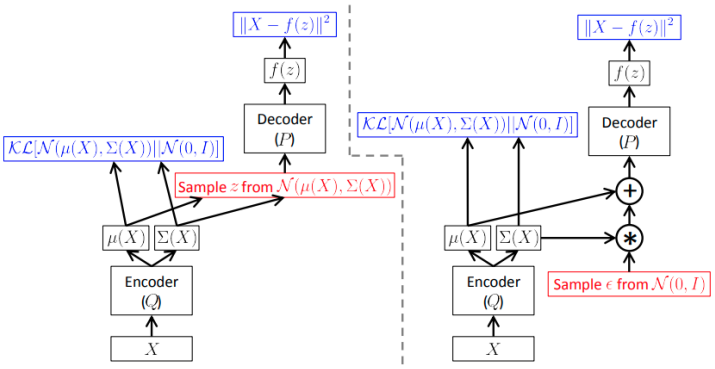

Figure 4: A training-time variational autoencoder implemented as a feed-forward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network.

The solution is called the reparametrization trick. We move the sampling to an input layer, $\epsilon \sim \mathcal{N}(0, I)$, and then compute $z = \mu(X) + \Sigma^{1/2}(X) * \epsilon$, where $\mu(X), \Sigma(X)$ were computed by Q.

Thus, the equation we actually take the gradient of is:

$$E_{X \sim D}\Big[E_{\epsilon \sim \mathcal{N}(0,I)}\big[\log P(X|\boldsymbol{z} = \mu(X) + \Sigma^{1/2}(X) * \epsilon)\big] - \mathrm{KL}\big[Q(\boldsymbol{z}|X) \,||\, P(\boldsymbol{z})\big]\Big]$$

And since this sampling is independent of θ and ϕ, we don't need to do backpropagation over the sampling.

3.1 Test Time

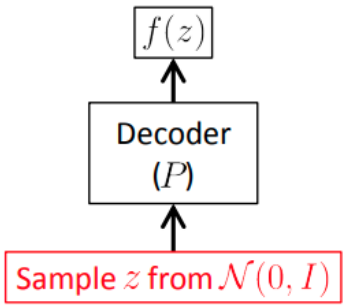

Figure 5: The testing-time variational "autoencoder," which allows us to generate new samples. The "encoder" pathway is simply discarded.

4 Diagnosing and Enhancing VAE Models

Bin Dai, David Wipf Microsoft / Amazon / Tsinghua / 2019

We wish to learn a probabilistic generative model of observale variables $\boldsymbol{x} \in \mathcal{X}$, where $\mathbb{X}$ is an r-dimensional manifold embedded in $\mathbb{R}^d$. Moreover, we assume that $\mathcal{X}$ is a simple Riemannian manifold, which means there exists a diffeomorphism $\phi : \mathcal{X} \to \mathbb{R}^r$. Denote the ground-truth probability measure on $\mathcal{X}$ as μ_{gt} such that the probability mass of an infinitesimal $d\boldsymbol{x}$ on the manifold is $\mu_{gt}(d\boldsymbol{x})$, and $\int_{\mathcal{X}} \mu_{gt}(d\boldsymbol{x}) = 1$.

The VAE attempts to approximate this ground truth measure, using a parameterized density $p_\theta(\boldsymbol{x})$ defined across all of $\mathbb{R}^d$. The density is further assumed to admit the latent decomposition $p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{z})\, d\boldsymbol{z}$, where $z \in \mathbb{R}^k$ with $k \approx r$, and prior $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, \boldsymbol{I})$.

Ideally, we might like to minimize the negative log likelihood $-\log p_\theta(\boldsymbol{x})$ averaged across the ground-truth measure μ_{gt}. i.e.:

$$\min_\theta \int_{\mathcal{X}} -\log p_\theta(\boldsymbol{x}) \mu_{gt}\,(d\boldsymbol{x})$$

Unfortunately, the required marginalization over $\boldsymbol{z}$ is generally infeasible. Instead, the VAE model relies on tractable *encoder* $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and *decoder* $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ distributions, where ϕ represents additional trainable parameters.

The canonical VAE can be written as:

$$\mathcal{L}(\theta, \phi) := \int_{\mathcal{X}} \Big\{ -E_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] + KL[q_\phi(\boldsymbol{z}|\boldsymbol{x})] \Big\} \mu_{gt}(d\boldsymbol{x})$$

5 Summary

Assume we have n data vectors, $\boldsymbol{x}_1, \ldots \boldsymbol{x}_n$, each $\boldsymbol{x}_i$ a d-vector (for example images). We assume the data was generated via a latent variable model (VLM): each image $\boldsymbol{x}_i$ was generated There is a latent space $\mathcal{Z} \subseteq \mathbb{R}^k$, and $\boldsymbol{z} \sim p_{\boldsymbol{z}}$.