Lecture Script

Latent Variable Models and VAEs

Based on Alex Alemi's blog posts:

- *KL is All You Need* (2024)
- *A Path to the Variational Diffusion Loss* (2022)

KL divergence: the right ruler [0:05–0:20]

Weight of evidence

Posterior odds = likelihood ratio × prior odds:

$$\frac{\Pr(P \mid X)}{\Pr(Q \mid X)} = \frac{\Pr(X \mid P)}{\Pr(X \mid Q)} \cdot \frac{\Pr(P)}{\Pr(Q)}$$

Take log → additive:

$$\underbrace{\log \frac{\Pr(P \mid X)}{\Pr(Q \mid X)}}_{\text{posterior log-odds}} = \underbrace{\log \frac{\Pr(X \mid P)}{\Pr(X \mid Q)}}_{\text{weight of evidence}} + \underbrace{\log \frac{\Pr(P)}{\Pr(Q)}}_{\text{prior log-odds}}$$

The **weight of evidence** tells you how much to update your beliefs. Positive → evidence for P. Negative → evidence for Q.

KL = expected weight of evidence

Now suppose our two hypotheses *are* probability distributions: P says data comes from density $p(x)$, Q says it comes from $q(x)$. For a single observation x, the weight of evidence is just the log density ratio $\log \frac{p(x)}{q(x)}$.

If P is actually true, the *expected* weight of evidence is:

$$\mathrm{KL}[p;q] \equiv \left\langle \log \frac{p(x)}{q(x)} \right\rangle_{p(x)} = \int p(x) \log \frac{p(x)}{q(x)}\, dx$$

This is the KL divergence. It measures: **how quickly could we tell p and q apart, using samples from p?**

Asymmetry is natural: if P is a fair coin and Q is double-headed, then $\mathrm{KL}[p;q] = \infty$ (we'll eventually see tails), but $\mathrm{KL}[q;p]$ is finite (we only see heads, can never rule out fair coin).

Two properties we need

1. Non-negativity.

$$\mathrm{KL}[p;q] \geq 0$$

Equality iff $p = q$. The world can't systematically lie to us: on average, evidence points toward the truth.

2. Monotonicity (data processing inequality).

$$\mathrm{KL}[p(x,y);\, q(x,y)] \;\geq\; \mathrm{KL}[p(x);\, q(x)]$$

Observing (x, y) jointly gives you at least as much power to distinguish two hypotheses as observing x alone. You can't do better with less information.

This is the key move for VAEs: if the joint KL (over data + latents) is small, the marginal KL (over data alone) must be small too.

▹ These two properties—non-negativity and monotonicity—are the only math we need. Everything else follows.

The universal recipe

[0:20–0:25]

The Recipe

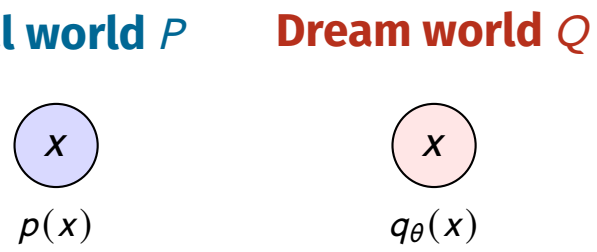
1. Write down the **real world** P: the joint distribution over all your random variables, as they actually occur.
2. **Augment** it with anything you want to add (representations, parameters, …).
3. Write down the **dream world** Q—what success looks like.
4. **Minimize** $\mathrm{KL}[P; Q]$.

Make reality indistinguishable from the dream. The smaller $\mathrm{KL}[P; Q]$, the harder it is—for us or anyone—to tell reality from our model.

▹ Let's see this recipe in action, starting simple.

Warmup: density estimation

[0:25–0:30]



Real world: black box generates samples $x \sim p(x)$. We can push the button but don't know p.

Dream world: we *wish* those samples came from our model $q_\theta(x)$ instead.

Apply the recipe—minimize $\mathrm{KL}[p; q_\theta]$:

$$\mathrm{KL}[p; q_\theta] = \left\langle \log \frac{p(x)}{q_\theta(x)} \right\rangle_p = \underbrace{-H[p]}_{\text{constant}} + \underbrace{\langle -\log q_\theta(x) \rangle_p}_{\text{cross-entropy}}$$

Drop the constant → **maximize likelihood**. Maximum likelihood = make the dream indistinguishable from reality.

Why this is hard in practice

In high dimensions, probability mass ≠ density. A d-dimensional Gaussian has its mode at the origin, but samples live on a thin shell at radius $\approx \sqrt{d}$. Density peaks at the center; volume grows exponentially with radius. Mass = density × volume, so it concentrates on the shell.

Real data lives on a low-dimensional manifold inside a huge ambient space. Modeling $q_\theta(x)$ directly means placing mass on a needle in a haystack. We need a coordinate system for the manifold.

▸ Solution: introduce latent variables.

Augmenting the real world: representations

[0:30–0:40]

We're going to **add something new** to the real world.

The setup

We have images $x \sim p(x)$ (real world, outside our control).

We **augment** the real world with a new random variable z—a **representation**. We build an encoder $p_\phi(z \mid x)$ that maps images to codes.

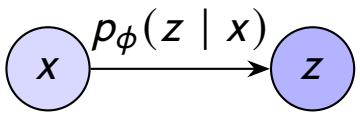


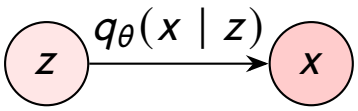
image representation

The real world joint is $p(x, z) = p(x)\, p_\phi(z \mid x)$. This is a whole family of possible real worlds, one for each setting of the encoder parameters ϕ.

What does success look like?

We *wish* the representations were actually **latents that generate images**. Success = there exists a simple story:

1. Draw z from a simple prior $q(z)$
2. Decode to an image $q_\theta(x \mid z)$



$q(z) = \mathcal{N}(0, I)$ generated image

The dream world joint is $q(x, z) = q(z)\, q_\theta(x \mid z)$.

▹ Now we just apply the recipe.

The ELBO from KL

[0:40–0:55]

The objective

$$\mathrm{KL}[p; q] = \left\langle \log \frac{p(x, z)}{q(x, z)} \right\rangle_p = \left\langle \log \frac{p(x)\, p_\phi(z \mid x)}{q(z)\, q_\theta(x \mid z)} \right\rangle_p \geq 0$$

This is a joint KL over (x, z). We sample from the forward process: draw $x \sim p(x)$, encode $z \sim p_\phi(z|x)$, then ask how surprised the reverse process would be.

Why training the joint improves the generative model

Here's where monotonicity pays off. Our objective is a *joint* KL over (x, z)—but what we actually care about is the *marginal* $q(x) = \int q(z)q_\theta(x|z)\,dz$, the distribution of images our generative model produces. Monotonicity guarantees one controls the other:

$$\underbrace{\left\langle \log \frac{p(x,z)}{q(x,z)} \right\rangle_p}_{\text{what we optimize (joint KL)}} \geq \underbrace{\left\langle \log \frac{p(x)}{q(x)} \right\rangle_p}_{\text{what we care about (marginal KL)}} \geq 0$$

Push down the left side $\rightarrow$ the right side *must* also go down. Training the encoder-decoder pair (joint) guarantees we are also building a good generative model (marginal).

Splitting into terms

$$\begin{aligned} \mathrm{KL}[p;q] &= \langle \log p(x) \rangle + \left\langle \log \frac{p_\phi(z|x)}{q(z)} \right\rangle - \langle \log q_\theta(x|z) \rangle \\ &= \underbrace{-H[p]}_{\text{constant}} + \underbrace{\langle -\log q_\theta(x|z) \rangle}_{D\ \text{(distortion)}} + \underbrace{\left\langle \log \frac{p_\phi(z|x)}{q(z)} \right\rangle}_{R\ \text{(rate)}} \end{aligned}$$

Drop the constant entropy $H[p]$ (we can't control it). We're left with:

$D = \langle -\log q_\theta(x \mid z) \rangle$ **Distortion** (reconstruction)
$R = \langle \mathrm{KL}(p_\phi(z|x) \| q(z)) \rangle$ **Rate** (encoding cost)

$D + R$ is the **ELBO** (up to sign)—derived from "make forward and reverse indistinguishable" rather than "lower-bound the log-likelihood."

Reading the terms

Distortion D: Given an encoded z, how well does the decoder reconstruct x? Minimizing D alone → a regular autoencoder.

Rate R: How far is the encoding distribution $p_\phi(z|x)$ from the prior $q(z) = \mathcal{N}(0, I)$? This is the information cost of the code. Minimizing R alone → ignore the data, just match the prior (trivial encoder).

Together: we want a code that's cheap (low R) but informative enough to reconstruct (low D). This tension structures the latent space.

▹ Let's make this concrete.

The VAE in practice [0:55–1:05]

Architecture

Encoder $p_\phi(z|x) = \mathcal{N}\big(\mu_\phi(x),\ \sigma^2_\phi(x)\, I\big)$

Neural net takes x, outputs μ and $\log \sigma^2$. Two heads.

Decoder $q_\theta(x|z)$: neural net takes z, outputs the reconstructed x.

For images: Gaussian → MSE loss. For binary: Bernoulli → BCE loss.

Prior $q(z) = \mathcal{N}(0, I)$.

Closed-form KL

For two Gaussians, the rate R has a closed form:

$$\mathrm{KL}\big(\mathcal{N}(\mu, \sigma^2 I) \,\|\, \mathcal{N}(0, I)\big) = \frac{1}{2} \sum_{j=1}^{d} \left(\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1\right)$$

No sampling needed for this term. Only the reconstruction term D requires a Monte Carlo sample (one sample of z per data point usually suffices).

Training loop

For each mini-batch of data $\{x_i\}$:

1. Encode: compute $\mu_\phi(x_i)$, $\sigma_\phi(x_i)$
2. Sample: $z_i = \mu_\phi(x_i) + \sigma_\phi(x_i) \odot \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, I)$
3. Decode: compute $\log q_\theta(x_i \mid z_i)$
4. Loss: $\mathcal{L} = \underbrace{-\log q_\theta(x_i \mid z_i)}_{D} + \underbrace{\mathrm{KL}(p_\phi(z|x_i) \| q(z))}_{R}$
5. Backprop and update ϕ, θ

Generation

After training, **discard the encoder**. To generate:

$$z \sim \mathcal{N}(0, I) \quad \longrightarrow \quad x = g_\theta(z)$$

The KL regularization ensured the decoder is trained everywhere the prior has mass. No holes.

▸ The D vs R tradeoff is the soul of the VAE.

The β-VAE and the rate-distortion tradeoff

[1:05–1:10]

$$\mathcal{L}_\beta = D + \beta \cdot R = \langle -\log q_\theta(x|z) \rangle + \beta \cdot \mathrm{KL}(p_\phi(z|x) \| q(z))$$
