Machine Learning and Optimization

Lecture 21 : Generative Models — the Evidence Lower Bound and Variational Autoencoders

CSCI 4140/6960 — Spring 2026

Course website: `cs.rpi.edu/.../mlandopt.html`

Learning Objectives

By the end of this lecture, you should be able to:

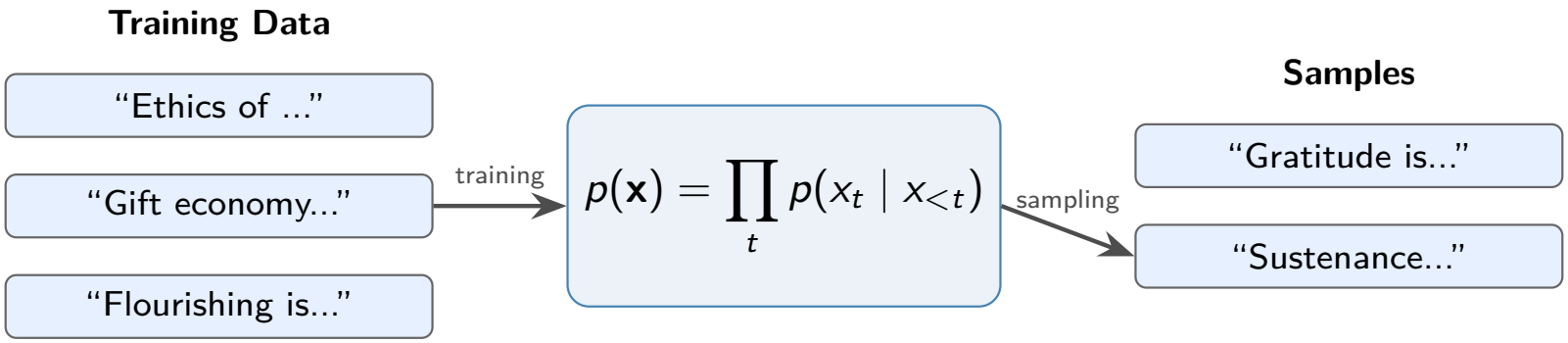
1. **Explain** what we can do with generative models and why latent variable structure helps
2. **Explain** why maximum likelihood is intractable for latent variable models
3. **Derive** the ELBO and interpret its two terms as competing objectives
4. **Analyze** the tension between expressivity and regularization in the ELBO
5. **Compare** score-function and reparameterization gradient estimators
6. **Apply** reparameterization to compute gradients for Gaussian VAEs
7. **Implement** a complete VAE training loop with proper gradient handling
8. **Evaluate** the limitations of basic VAEs and identify extension directions

Lecture Roadmap

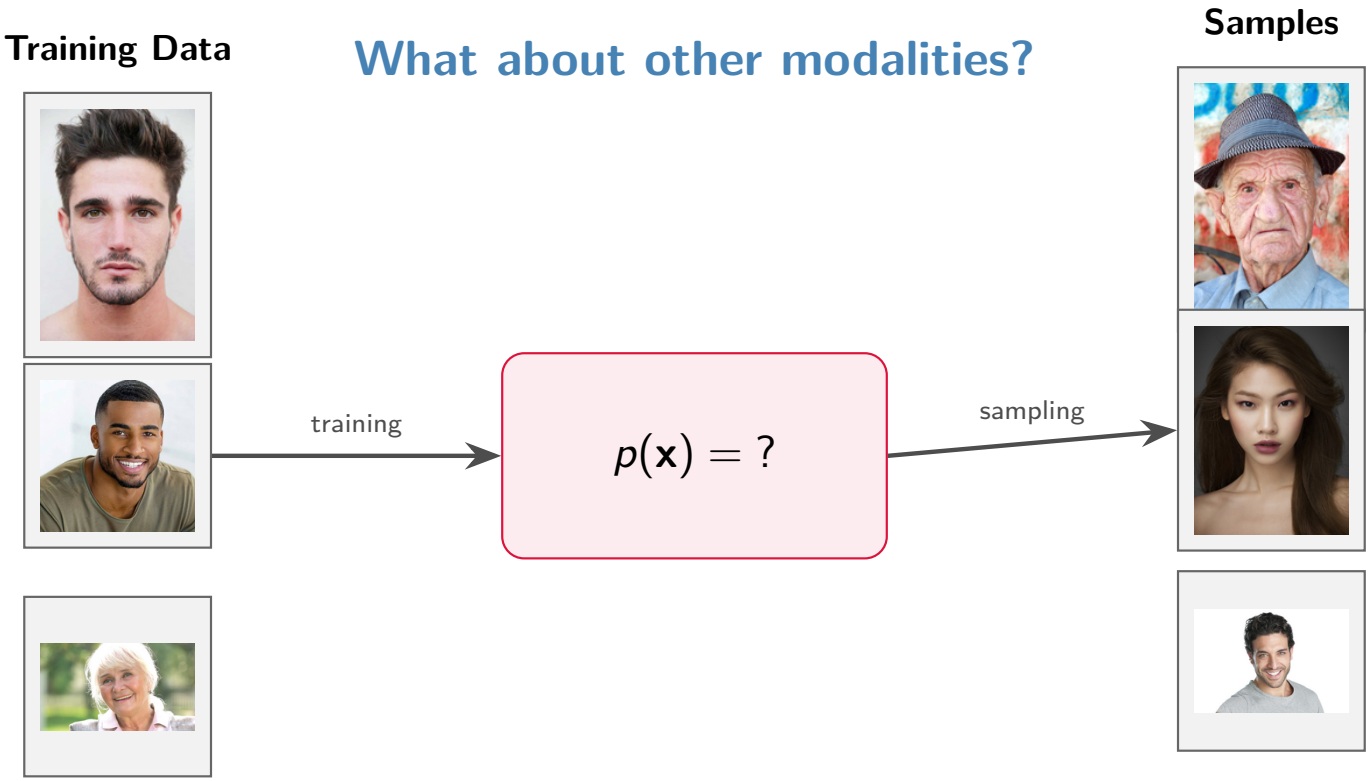
1. Latent Variable Models
2. The Intractability Problem and Variational Inference
3. The Evidence Lower Bound
4. Training Algorithms and Gradient Computation
5. Gaussian VAEs in Practice
6. Wrap-up and Future Directions

Latent Variable Models

Sampling from other modalities



Works for text — natural sequential order.



From Autoregressive to Latent Variable Models

LLMs exploit *natural sequential order* to model

$$p_\theta(\mathbf{x}) = \prod_t p_\theta(x_t | x_{<t}).$$

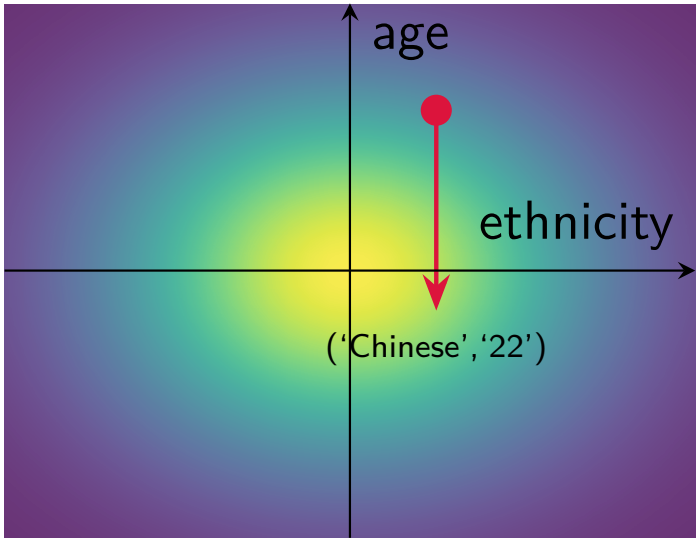
Other modalities like image, molecules, proteins, audio spectrograms have no canonical ordering. But they often can be assumed to have *latent variable* structure.

Sampling with latent variable structure:

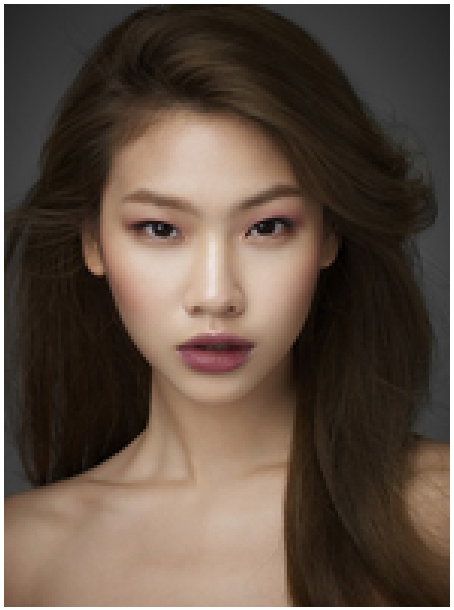
1. Choose a simple *prior* $p(\mathbf{z})$ over latent codes $\mathbf{z}$ — typically $\mathcal{N}(\mathbf{0}, \mathbf{I})$
2. Sample latent variable $\mathbf{z} \sim p(\mathbf{z})$ — a low-dimensional code encoding hidden factors like *gender*, *age*, *race*, *pose*, *lighting*
3. Sample $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$ — the *decoder* maps the latent code to observations

Why "latent"? We never observe latent codes $\mathbf{z}$ — only $\mathbf{x}$.
Why "decoder"? $p_\theta(\mathbf{x}|\mathbf{z})$ decodes the latent code to an observed output.

Sampling from Latent Variable Models



Sample a latent code $\mathbf{z}^* \sim p(\mathbf{z})$



Sampled from $p_\theta(\mathbf{x}|\mathbf{z}^*)$

What Can We Do with a Generative Model?

Generative models learn $p_\theta(\mathbf{x})$ to enable several downstream tasks:

1. **Sampling:** Generate new data $\mathbf{x} \sim p_\theta(\mathbf{x})$ that resembles training data
2. **Likelihood computation:** Evaluate how probable an observation is under the model
   - Useful for anomaly detection, out-of-distribution detection
3. **Representation learning:** Extract latent codes $\mathbf{z}$ that capture meaningful semantic factors
   - Use as features for downstream classification, clustering, visualization
4. **Conditional generation:** Sample from $p_\theta(\mathbf{x} \mid \text{attributes})$ by conditioning on $\mathbf{z}$

Challenge: All of these require computing or approximating integrals over the latent space.

Learning the decoder

As with all our other models up to this point, we want to learn θ by maximizing the likelihood of the model under the observed data

$$\theta^\star = \operatorname*{argmax}_\theta \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(\mathbf{x}_i)$$

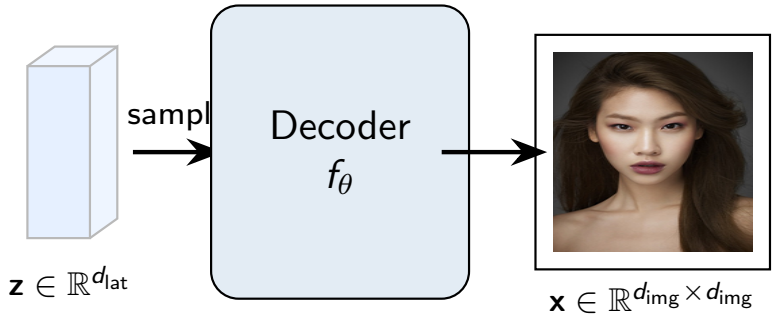
The marginal likelihood of the model θ for a given observation is:

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

Expanding the joint distribution:

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

Concrete Example: Decoder for Images



Decoder maps z to a Gaussian distribution over images:

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}\big(\mu(\mathbf{z}), \mathrm{Diag}(\sigma^2(\mathbf{z}))\big)$$

Simple network architecture:

$$h = \mathrm{ReLU}\big(W_1\mathbf{z} + b_1\big) \quad \text{(latent} \rightarrow \text{hidden)}$$

$$\mu(\mathbf{z}), \log\sigma^2(\mathbf{z}) = \mathrm{DeConvNet}_\theta(h) \quad \text{(hidden} \rightarrow \text{pixel means and variances)}$$

with model parameters: $\theta = W_1, b_1, \theta_{\text{DeConvNet}}$

Diagonal covariance makes pixels conditionally independent given z.

Pause and Check: Latent Variable Models

Question:
Consider a face generation model with latent codes $\mathbf{z} \in \mathbb{R}^{10}$ where each dimension represents: [gender, age, skin tone, pose angle, lighting direction, glasses, facial hair, emotion, ...].

Suppose we sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and decode to get a face image. If we increase the value of the "age" dimension while keeping others fixed, what do we expect to happen? What would happen if we sampled age from $\mathcal{N}(0, 100)$ instead of $\mathcal{N}(0, 1)$?

Think about:

- How do latent dimensions acquire semantic meaning during training?
- Why might extreme latent values produce unrealistic images?

The Intractability Problem and Variational Inference

Terminology

Let's pause to introduce important terminology formally:

- **Full-data likelihood**: $p_\theta(\mathbf{x}, \mathbf{z})$ — joint probability of observation and latent variable
- **Evidence** (marginal likelihood): $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z})\, d\mathbf{z}$ — probability of observation alone
- **Likelihood** (conditional): $p_\theta(\mathbf{x}|\mathbf{z})$ — probability of observation given specific latent code
- **Prior:** $p(\mathbf{z})$ — distribution over latents before seeing data
- **Posterior:** $p_\theta(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{x}|\mathbf{z})\, p(\mathbf{z}) / p_\theta(\mathbf{x})$ — distribution over latents after observing $\mathbf{x}$

Note: The marginal and posterior both involve integration over the latent space. We will examine whether this presents computational challenges.

The Intractability of MLE

For a neural-network decoder $p_\theta(\mathbf{x}|\mathbf{z})$ with $\mathbf{z} \in \mathbb{R}^d$ where d is large (e.g., $d = 128$ or more):

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

is a **high-dimensional integral of a neural network** with respect to its inputs ($\mathbf{z}$).

This integral has **no closed form**: the neural network's nonlinearity prevents analytic integration, and the high dimensionality makes numerical integration infeasible.

Therefore we cannot evaluate $\log p_\theta(\mathbf{x})$, let alone optimize it! The MLE objective

$$\theta^\star = \operatorname*{argmax}_\theta \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(\mathbf{x}_i)$$

is computationally intractable.

Monte Carlo Methods Don't Scale

A naive approach is to estimate the integral by sampling $\mathbf{z}^{(s)} \sim p(\mathbf{z})$ and average:

$$p_\theta(\mathbf{x}) \approx \frac{1}{S} \sum_{s=1}^{S} p_\theta(\mathbf{x}|\mathbf{z}^{(s)})$$

This estimator has **exponential variance in the latent dimension**.

- In high dimensions ($d = 128$), random samples $\mathbf{z}^{(s)}$ rarely land where $p_\theta(\mathbf{x}|\mathbf{z})$ is large
- Most samples contribute essentially zero to the sum
- To get a non-negligible estimate, need $S \propto e^d$ samples — **computationally infeasible**

Random samples from the prior rarely land where $p_\theta(\mathbf{x}|\mathbf{z})$ is large. We should instead sample from regions that are actually relevant for explaining $\mathbf{x}$.

Strategy: Tractable Lower Bound

Since $p_\theta(\mathbf{x})$ is intractable, we cannot directly maximize $\log p_\theta(\mathbf{x})$.

Instead of maximizing the likelihood itself, we will develop and maximize a *tractable approximation* that also has a nice interpretation.

Our starting point is the identity

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{\mathbf{z} \sim p_\theta(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}) = \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{z} \mid \mathbf{x})} \right]$$

Unfortunately, the right-hand side is also intractable: we can compute $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})$, but **computing $p_\theta(\mathbf{z} \mid \mathbf{x})$ is as intractable as $p_\theta(\mathbf{x})$** since $p_\theta(\mathbf{z} \mid \mathbf{x}) = p_\theta(\mathbf{x}, \mathbf{z}) / p_\theta(\mathbf{x})$.

Approximating the Posterior

Interpretation: Given our decoder $p_\theta(\mathbf{x}|\mathbf{z})$ and prior $p(\mathbf{z})$, the posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is the corresponding *encoder* — it maps observations $\mathbf{x}$ to latent codes $\mathbf{z}$.

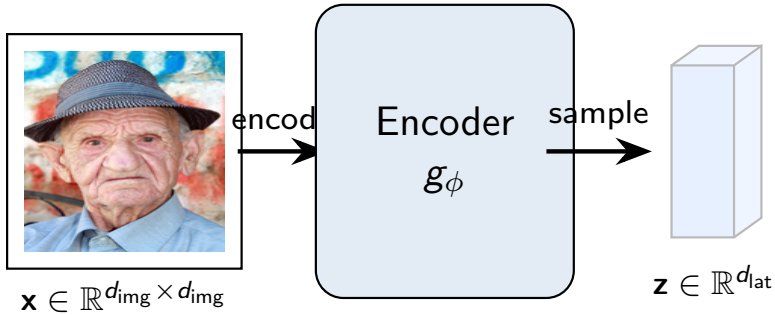
The true posterior is *completely determined* by our modeling choices, and is intractable. **If we had a tractable approximate posterior** $q(\mathbf{z} \mid \mathbf{x})$, then

$$\log p_\theta(\mathbf{x}) \approx \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q(\mathbf{z} \mid \mathbf{x})} \right]$$

and the right-hand side can be optimized w.r.t θ!

Solution (VAEs): learn an *approximate encoder* $q_\phi(\mathbf{z}|\mathbf{x})$ — a neural network parameterized by ϕ that maps observations to latent codes. Unlike the true posterior, q_ϕ is tractable (we can sample and evaluate it) and differentiable (we can optimize it).

Concrete Example: Encoder for Images



Encoder maps x to a Gaussian distribution over latent codes:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mu_\phi(\mathbf{x}), \mathrm{Diag}(\sigma^2_\phi(\mathbf{x}))\right)$$

Simple network architecture:

$$h = \mathrm{ConvNet}_\phi(\mathbf{x}) \quad \text{(image} \rightarrow \text{hidden)}$$

$$\mu_\phi(\mathbf{x}), \log \sigma^2_\phi(\mathbf{x}) = \mathrm{MLP}_\phi(h) \quad \text{(hidden} \rightarrow \text{latent means and variances)}$$

with encoder parameters: $\phi = \phi_{\text{ConvNet}}, \phi_{\text{MLP}}$

Diagonal covariance makes latent dimensions conditionally independent given x.

Pause and Check: Why Approximate the Posterior?

Question:
Suppose we tried to use the **true posterior** $p_\theta(\mathbf{z}|\mathbf{x})$ directly in the ELBO instead of an approximation $q_\phi(\mathbf{z}|\mathbf{x})$.
Recall that $p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p_\theta(\mathbf{x})}$.

Which of the following operations would be problematic, and why?

1. Sampling $\mathbf{z} \sim p_\theta(\mathbf{z}|\mathbf{x})$ to compute Monte Carlo estimates
2. Computing the density $p_\theta(\mathbf{z}|\mathbf{x})$ for a given $\mathbf{z}$
3. Computing gradients of $p_\theta(\mathbf{z}|\mathbf{x})$ with respect to model parameters θ

Answer: Using q_ϕ is not merely convenient—it makes gradient-based learning tractable, which the true posterior does not.

The Evidence Lower Bound

Bounding the Approximation Error

We want to estimate $\log p_\theta(\mathbf{x})$ by approximating the intractable true posterior with the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$. **Is this even a useful approximation?**

Let's compute the error in estimating the evidence:

$$\log p_\theta(\mathbf{x}) - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} \right]$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}) - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} \right]$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})} \right] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})} \right]$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})} \cdot \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{x}, \mathbf{z})} \right]$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log \left[\frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} \right] = D_{\mathrm{KL}}\big(q_\phi(\mathbf{z}|\mathbf{x}) \,\big\|\, p_\theta(\mathbf{z}|\mathbf{x})\big)$$

The Evidence Lower Bound (ELBO)

We have shown that

$$\log p_\theta(\mathbf{x}) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]}_{\mathcal{L}_{\mathrm{ELBO}}(\phi,\theta;\mathbf{x})} + \underbrace{D_{\mathrm{KL}}\big(q_\phi(\mathbf{z}|\mathbf{x}) \,\big\|\, p_\theta(\mathbf{z}|\mathbf{x})\big)}_{\geq 0}$$

Our approximation of the evidence using the learned encoder is called the **Evidence Lower Bound (ELBO):**

$$\mathcal{L}_{\mathrm{ELBO}}(\phi,\theta;\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$$

Key properties:

- Since $D_{\mathrm{KL}} \geq 0$: $\log p_\theta(\mathbf{x}) \geq \mathcal{L}_{\mathrm{ELBO}}(\phi,\theta;\mathbf{x})$
- Gap $= D_{\mathrm{KL}}\big(q_\phi(\mathbf{z}|\mathbf{x}) \,\big\|\, p_\theta(\mathbf{z}|\mathbf{x})\big)$ measures approximation error
- Maximizing $\mathcal{L}_{\mathrm{ELBO}}$ w.r.t. ϕ pushes q_ϕ toward p_θ

ELBO: Two Terms

Starting from the ELBO definition:

$$\mathcal{L}_{\mathrm{ELBO}}(\phi, \theta; \mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})\, p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$$

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

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

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})}\right]$$

$$= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x}|\mathbf{z})\big]}_{\text{reconstruction}} - \underbrace{D_{\mathrm{KL}}\big(q_\phi(\mathbf{z}|\mathbf{x}) \,\big\|\, p(\mathbf{z})\big)}_{\text{prior matching}}$$

The Expressivity–Regularization Tension

Recall the two-term decomposition:

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{reconstruction}} - \underbrace{\mathrm{D}_{\mathrm{KL}}(q_\phi \| p)}_{\text{prior matching}}$$

These terms create a **tension**:

Reconstruction wants:

- $q_\phi(\mathbf{z}|\mathbf{x})$ to be *deterministic*
- Always encode $\mathbf{x}$ to the single "best" $\mathbf{z}^*$
- High confidence, low entropy

Prior matching wants:

- $q_\phi(\mathbf{z}|\mathbf{x})$ to be *stochastic*
- Match the prior spread (e.g., $\mathcal{N}(\mathbf{0}, \mathbf{I})$)
- Non-zero entropy, uncertainty

Resolution: The VAE learns to encode with just enough uncertainty to satisfy the prior while still enabling good reconstructions. This tradeoff shapes the learned representation.

What is a Variational Autoencoder?

A **Variational Autoencoder (VAE)** learns the encoder q_ϕ and decoder p_θ simultaneously by maximizing the ELBO.

Why "variational"? The term comes from *variational inference*, where we approximate an intractable optimization problem (finding the true posterior) with a tractable one (optimizing over a family of distributions).

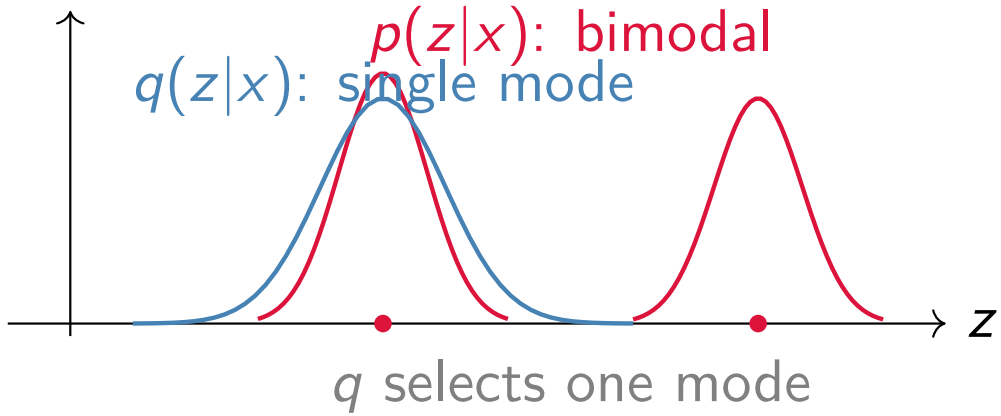
Why "autoencoder"? The architecture mirrors an autoencoder: an *encoder* compresses data $\mathbf{x}$ to latent codes $\mathbf{z}$, and a *decoder* reconstructs $\mathbf{x}$ from $\mathbf{z}$.

Optimization: We optimize both ϕ and θ jointly using stochastic gradient ascent. This is non-convex (local minima exist), but the reparameterization trick we see later provides stable gradients in practice.

Mode-Seeking Behavior

The KL divergence $\mathrm{D}_{\mathrm{KL}}(q\|p)$ is **asymmetric**. Optimizing w.r.t q has a *mode-seeking* behavior.

q is penalized only when it has mass where $p = 0$. Ignoring a mode of p incurs no penalty. Optimization drives q to concentrate on a single high-probability region of p.



Implication for VAEs: The ELBO gap equals $\mathrm{D}_{\mathrm{KL}}(q_\phi\|p_\theta)$—this is mode-seeking. A Gaussian encoder $q_\phi(z|x)$ approximates a multimodal true posterior $p_\theta(z|x)$ by concentrating on a single mode. The approximation is good locally but misses alternative explanations.

Posterior Collapse

Posterior collapse: the decoder reconstructs $\mathbf{x}$ without using $\mathbf{z}$, so $q_\phi(\mathbf{z}|\mathbf{x}) = p(\mathbf{z})$ with KL $= 0$. The latent code becomes uninformative.

Why bad: No representation learning, no controllable generation, wasted architecture.

Fixes:

- β-**VAE:** Weight KL by $\beta < 1$ (weaker regularization)
- **KL annealing:** Gradually increase KL weight during training
- **Constraints:** Limit decoder capacity or use skip connections

Latent Space Geometry

The KL regularizer $\mathrm{D}_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$ pulls encoded distributions toward the prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Effects:

- **Structured organization:** The KL encourages similar data points to have similar latent codes (to minimize KL while maintaining reconstruction)
- **Smooth interpolation:** Because similar $\mathbf{x}$ map to nearby $\mathbf{z}$, interpolating between two latent codes produces gradual changes in the decoded output
- **Limitation:** The standard Gaussian prior may be too simple for complex data distributions

Result: latent codes cluster near origin with roughly unit variance, enabling controlled generation via latent space traversal.

Pause and Check: Understanding the ELBO Tradeoffs

Scenario:
You're training a VAE and observe two phenomena:

A The KL term $\mathrm{D}_{\mathrm{KL}}(q_\phi \| p)$ rapidly goes to near-zero

B Generated samples are blurry and lack diversity

Question: Are these observations consistent with each other? Explain the connection using the expressivity-regularization tension and KL asymmetry concepts.

Additionally: if you wanted to fix this by modifying the objective, would you increase or decrease the weight on the KL term? What would be the tradeoff?

Hint: Think about what happens when the encoder has "nothing to do" because the decoder ignores $\mathbf{z}$.

Training Algorithms and Gradient Computation

Training VAEs

To train VAEs, we simultaneously learn both parameter sets to maximize the average ELBO:

$$(\phi^\star, \theta^\star) = \underset{\phi,\theta}{\text{argmax}} \; \frac{1}{n} \sum_{i=1}^{n} \Big[\underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)} \big[\log p_\theta(\mathbf{x}_i|\mathbf{z}) \big]}_{\text{reconstruction}} - \underbrace{\mathrm{D}_{\mathrm{KL}} \big(q_\phi(\mathbf{z}|\mathbf{x}_i) \,\big\|\, p(\mathbf{z}) \big)}_{\text{prior matching}} \Big]$$

Algorithm: VAE Training (Stochastic Gradient Ascent on the ELBO)

Input: Data $\{\mathbf{x}_i\}_{i=1}^n$, prior $p(\mathbf{z})$, encoder q_ϕ, decoder p_θ, step size η

Initialize ϕ, θ

Repeat:

1. Sample mini-batch $B \subset \{1, \ldots, n\}$
2. Compute $\widehat{\mathcal{L}}_B(\phi, \theta) = \frac{1}{|B|} \sum_{i \in B} \mathcal{L}_{\mathrm{ELBO}}(\phi, \theta; \mathbf{x}_i)$
3. Update $\phi \leftarrow \phi + \eta \nabla_\phi \widehat{\mathcal{L}}_B$ (**hard**), $\theta \leftarrow \theta + \eta \nabla_\theta \widehat{\mathcal{L}}_B$ (**easy**)

Return ϕ, θ

The ELBO Gradients: Easy vs. Hard

Recall the ELBO decomposition:

$$\mathcal{L}_{\text{ELBO}}(\phi, \theta; \mathbf{x}) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{reconstruction}} - \underbrace{\mathrm{D}_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))}_{\text{prior matching}}$$

Easy gradients (standard autodiff):

- $\nabla_\theta \mathrm{D}_{\mathrm{KL}}(q_\phi \| p) = \mathbf{0}$ — decoder parameters don't appear in the KL
- $\nabla_\theta \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})]$ — expectation is "constant" w.r.t. θ
- $\nabla_\phi \mathrm{D}_{\mathrm{KL}}(q_\phi \| p)$ — often closed-form (e.g., for Gaussians)

Hard gradient (requires special handling):

- $\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$ — parameters appear *inside* the sampling distribution
- Cannot use autodiff directly because ∇_ϕ cannot pass through the expectation

Approaches for the Hard Gradient

Consider

$$\nabla_\phi \, \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x}|\mathbf{z})\big].$$

For convenience take $f(\mathbf{z}) = \log p_\theta(\mathbf{x} \mid \mathbf{z})$. Then

$$\nabla_\phi \, \mathbb{E}_{q_\phi}[f(\mathbf{z})] = \nabla_\phi \int f(\mathbf{z}) \, q_\phi(\mathbf{z} \mid \mathbf{x}) \, d\mathbf{z} = \int f(\mathbf{z}) \, \nabla_\phi q_\phi(\mathbf{z} \mid \mathbf{x}) \, d\mathbf{z}.$$

Two approaches to getting a stochastic estimator for this gradient:

1. **Score-function estimator (REINFORCE):** apply the log-derivative trick to get an unbiased but high-variance gradient.
2. **Reparameterization trick:** transfer ϕ out of the sampling distribution, yielding a low-variance estimator — when q_ϕ permits it.

Option 1: The Score-Function Estimator (REINFORCE)

By the chain rule, $\nabla_\phi \log q_\phi(\mathbf{z} \mid \mathbf{x}) = \frac{\nabla_\phi q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})}$, so

$$\begin{aligned}\nabla_\phi \mathbb{E}_{q_\phi}[f(\mathbf{z})] &= \int f(\mathbf{z}) \nabla_\phi q_\phi(\mathbf{z} \mid \mathbf{x}) \, d\mathbf{z} \\ &= \int f(\mathbf{z}) \, q_\phi(\mathbf{z} \mid \mathbf{x}) \nabla_\phi \log q_\phi(\mathbf{z} \mid \mathbf{x}) \, d\mathbf{z} \\ &= \mathbb{E}_{q_\phi}\big[f(\mathbf{z}) \nabla_\phi \log q_\phi(\mathbf{z} \mid \mathbf{x})\big].\end{aligned}$$

Draw $\mathbf{z}^{(s)} \sim q_\phi(\mathbf{z}|\mathbf{x})$ and estimate:

$$\nabla_\phi \mathbb{E}_{q_\phi}[f(\mathbf{z})] \approx \frac{1}{S} \sum_{s=1}^{S} f\big(\mathbf{z}^{(s)}\big) \nabla_\phi \log q_\phi\big(\mathbf{z}^{(s)}|\mathbf{x}\big).$$

Sample S samples from the approximate posterior $q_\phi(\cdot \mid \mathbf{x}_i)$ for *each* observation in the minibatch, to estimate its gradient. This estimator is unbiased, but also high variance: $f(\mathbf{z}^{(s)})$ fluctuates across samples and multiplies an already noisy score $\nabla_\phi \log q_\phi$. **In practice, REINFORCE is too noisy to train deep encoders.**

Option 2: The Reparameterization Trick

Suppose $q_\phi(\mathbf{z}|\mathbf{x})$ admits a representation

$$\mathbf{z} = g_\phi(\mathbf{x}, \boldsymbol{\epsilon}), \qquad \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}),$$

where g_ϕ is deterministic and $p(\boldsymbol{\epsilon})$ does not depend on ϕ. Then:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\big[f(g_\phi(\mathbf{x}, \boldsymbol{\epsilon}))\big].$$

Since $p(\boldsymbol{\epsilon})$ is ϕ-independent, the gradient passes inside:

$$\nabla_\phi \, \mathbb{E}_{p(\boldsymbol{\epsilon})}\big[f(g_\phi(\mathbf{x}, \boldsymbol{\epsilon}))\big] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\big[\nabla_\phi f(g_\phi(\mathbf{x}, \boldsymbol{\epsilon}))\big] \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_\phi f\big(g_\phi(\mathbf{x}, \boldsymbol{\epsilon}^{(s)})\big),$$

where $\boldsymbol{\epsilon}^{(s)} \sim p(\boldsymbol{\epsilon})$. The gradient flows through g_ϕ by backpropagation — no score-function weighting, so variance is much lower.

This requires that $q_\phi(\mathbf{z}|\mathbf{x})$ admit such a reparameterization. We specialize to the Gaussian encoder next.

Explicit Reparameterization for Gaussian Encoder

For a Gaussian encoder $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu_\phi(\mathbf{x}), \mathrm{Diag}(\sigma^2_\phi(\mathbf{x})))$:

Reparameterization: $\mathbf{z} = g_\phi(\mathbf{x}, \boldsymbol{\epsilon}) = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Gradient computation:

$$\nabla_\phi \mathbb{E}_{q_\phi}[f(\mathbf{z})] = \mathbb{E}_{\boldsymbol{\epsilon}}[\nabla_\phi f(\mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon})]$$

Unbiased estimate with single sample:

$$\nabla_\phi \mathbb{E}_{q_\phi}[f(\mathbf{z})] \approx \nabla_\phi f(\underbrace{\mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon}}_{\mathbf{z}})\Big|_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})}$$

This gradient flows through:

- $\mu_\phi(\mathbf{x})$ directly (encoder network output)
- $\sigma_\phi(\mathbf{x})$ directly (encoder network output)
- Through the sampled $\mathbf{z}$ into the decoder and loss

Gaussian VAEs in Practice

Putting It Together: Training an Image VAE

We now apply reparameterization to train the image encoder–decoder pair from earlier.

Recall the distributions:

- Encoder: $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu_\phi(\mathbf{x}), \mathrm{Diag}(\sigma^2_\phi(\mathbf{x})))$
- Decoder: $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mu_\theta(\mathbf{z}), \mathrm{Diag}(\sigma^2_\theta(\mathbf{z})))$

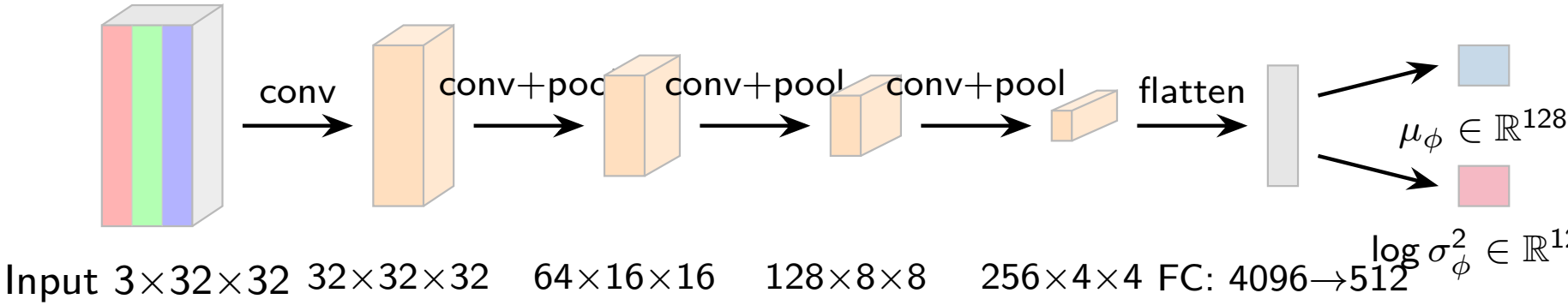
Neural network architectures:

- Encoder: ConvNet (4 layers) $\to$ FC $\to$ $(\mu_\phi, \log \sigma^2_\phi) \in \mathbb{R}^{d_{\text{lat}}}$
- Decoder: FC $\to$ DeConvNet (4 layers) $\to$ $(\mu_\theta, \log \sigma^2_\theta) \in \mathbb{R}^{3\times 32\times 32}$

Gradient computation:

- $\nabla_\phi \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}|\mathbf{z})] \to$ **reparameterization** (stochastic, low variance)
- $\nabla_\phi \, \mathrm{D}_{\mathrm{KL}}(q_\phi \| p) \to$ **closed-form** (exact, no sampling)

Encoder Architecture for Images

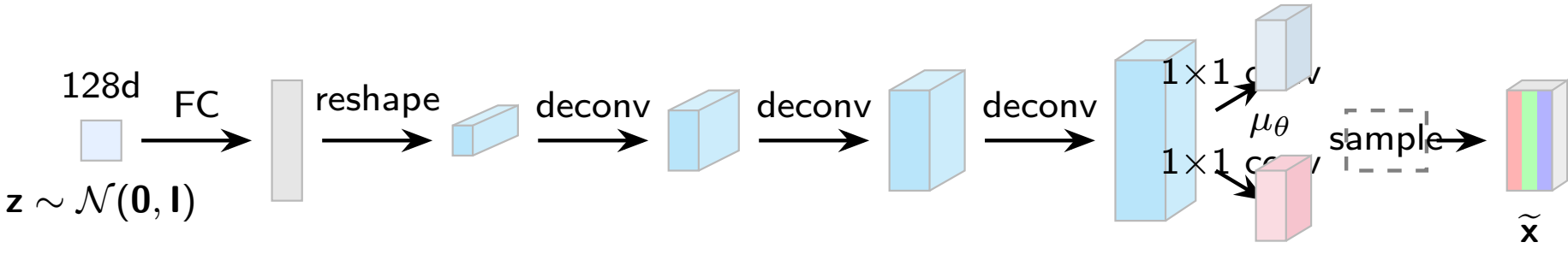


$$\text{Encoder } q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\big(\mu_\phi(\mathbf{x}),\, \mathrm{Diag}(\sigma^2_\phi(\mathbf{x}))\big)$$

Why this architecture?

- **ConvNet backbone:** Images have spatial structure; convolutions exploit locality and translation invariance
- **Progressive downsampling:** Each layer doubles channels and halves spatial dims, increasing receptive field
- **Split to $(\mu, \log \sigma^2)$:** Both are deterministic functions of $\mathbf{x}$; stochasticity comes from sampling step

Decoder Architecture for Images



$$\text{Decoder } p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}\big(\mu_\theta(\mathbf{z}),\, \mathrm{Diag}(\sigma^2_\theta(\mathbf{z}))\big)$$

Why this architecture?

- **Mirror of encoder:** Latent → FC expansion mirrors the encoder's final FC compression
- **DeConv layers:** Learned upsampling that can recover spatial structure
- **Per-pixel variance:** Learned heteroscedastic uncertainty (different noise level per pixel)
- **Final 1×1 conv:** Separate heads for mean and variance allow independent prediction per pixel

Training a Gaussian VAE on CINIC-10

Architecture.
Encoder: 4-layer ConvNet $\to (\mu_\phi, \log \sigma_\phi^2) \in \mathbb{R}^{128} \times \mathbb{R}^{128}$.

Decoder: FC + 4-layer DeConvNet $\to$
$(\mu_\theta, \log \sigma_\theta^2) \in \mathbb{R}^{3\times32\times32} \times \mathbb{R}^{3\times32\times32}$.

Latent dim $d_{\text{lat}} = 128$, batch size 128, Adam ($\text{lr} = 10^{-3}$), 100 epochs.

VAE Training Algorithm: Gaussian Encoder–Decoder

Algorithm: Gaussian VAE (SGD on the ELBO)

Input: data $\{\mathbf{x}_i\}_{i=1}^n$, encoder q_ϕ, decoder p_θ, step size η

Initialize ϕ, θ

Repeat:

1. Sample mini-batch $B \subset \{1, \ldots, n\}$
2. For each $\mathbf{x}_i \in B$:

a. $\mu_\phi(\mathbf{x}_i),\ \log \sigma^2_\phi(\mathbf{x}_i) \leftarrow \mathrm{Encoder}_\phi(\mathbf{x}_i)$

b. Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$; $\mathbf{z}_i \leftarrow \mu_\phi(\mathbf{x}_i) + \sigma_\phi(\mathbf{x}_i) \odot \boldsymbol{\epsilon}$

c. $\mu_\theta(\mathbf{z}_i),\ \log \sigma^2_\theta(\mathbf{z}_i) \leftarrow \mathrm{Decoder}_\theta(\mathbf{z}_i)$

d. $\mathcal{L}_i \leftarrow -\log p_\theta(\mathbf{x}_i|\mathbf{z}_i) + \mathrm{D}_{\mathrm{KL}}\big(q_\phi(\mathbf{z}|\mathbf{x}_i) \,\big\|\, \mathcal{N}(\mathbf{0}, \mathbf{I})\big)$

3. $\hat{\mathcal{L}}_B \leftarrow \frac{1}{|B|} \sum_{i \in B} \mathcal{L}_i$
4. Update ϕ, θ via `loss.backward()` + optimizer step

Return ϕ, θ

Implementation Notes

Implement `Encoder` and `Decoder` as `nn.Module` subclasses; PyTorch autograd handles step 4 automatically.

The algorithm uses $S = 1$ sample from q_ϕ per data point. This gives an **unbiased but stochastic** gradient estimate. Variance is manageable when combined with:

- The reparameterization trick (gradient flows through μ_ϕ, σ_ϕ)
- Large mini-batches (implicit averaging across data points)
- Multiple training epochs (iterative refinement)

Training Diagnostics and Best Practices

Initialization:

- Initialize $\log \sigma_\phi^2 \approx 0$ (unit variance encoder)
- Bad initialization can cause early posterior collapse

Monitoring:

- Track reconstruction loss vs KL loss separately
- Watch for KL collapse (KL $\to 0$ means decoder ignoring $\mathbf{z}$)
- Check that latent codes have non-zero variance

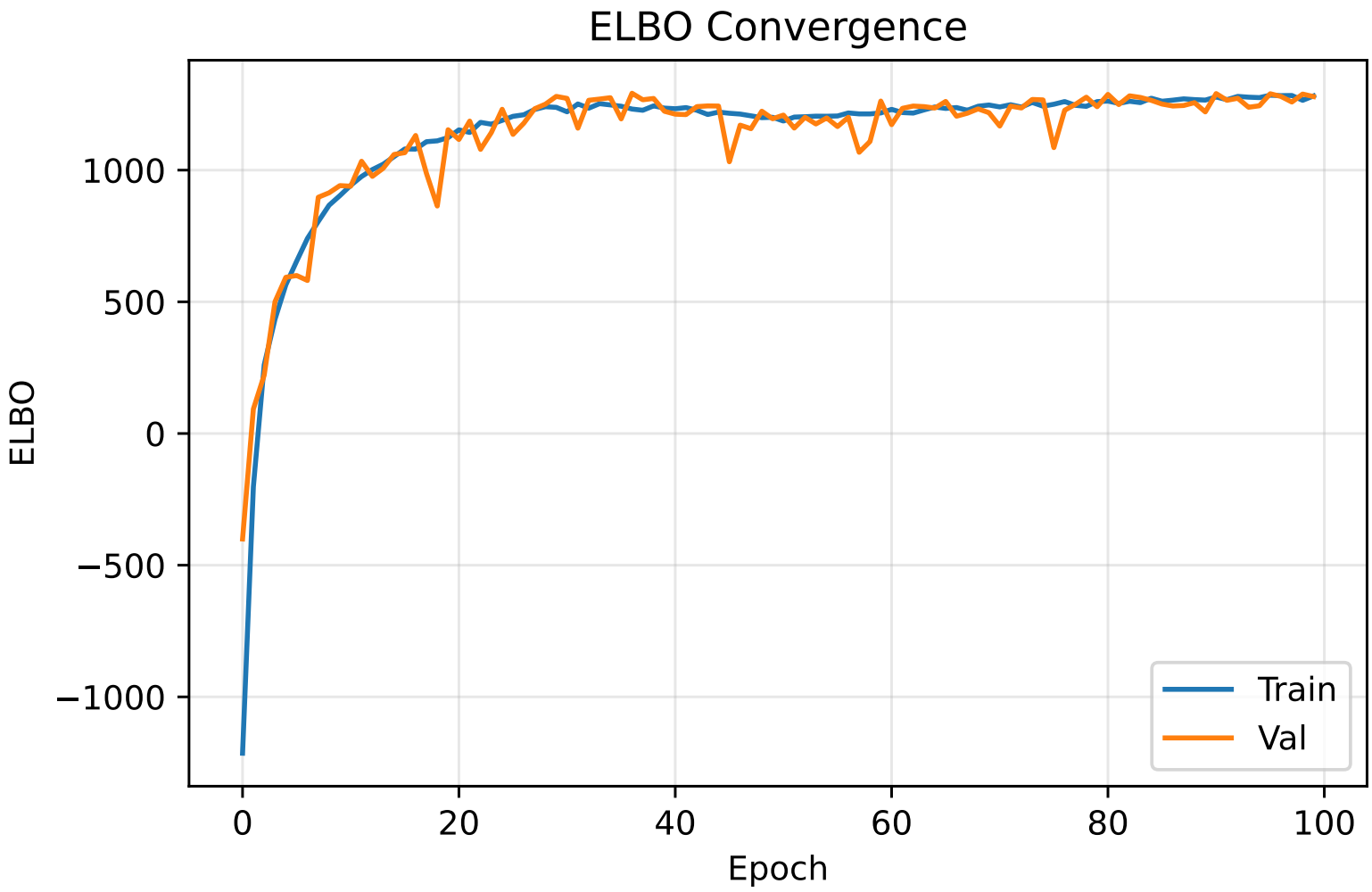
Optimization:

- Consider KL annealing schedule (start with $\beta = 0$, increase to 1)

Data Preprocessing:

- Normalize inputs to $[-1, 1]$ or $[0, 1]$
- Gaussian likelihood assumes continuous data

Training Dynamics



VAEs Produce Blurry Images

When multiple plausible reconstructions exist (e.g., object boundaries, fine textures), the Guassian likelihood model learns to predict the *average* over possibilities:

$$\mu_\theta(\mathbf{z}) \approx \mathbb{E}[\mathbf{x} \mid \text{plausible reconstructions}]$$

Blurring arises from averaging mutually exclusive sharp details. This is a fundamental limitation of the Gaussian decoder.

Alternatives:

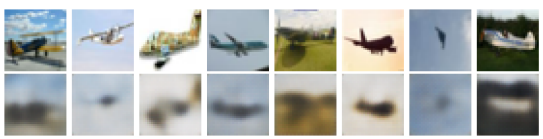
- **Autoregressive decoder** (PixelCNN): Models pixels sequentially, capturing multimodal distributions without averaging. Tradeoff: generation is sequential and slower.
- **Hierarchical/multi-scale VAE**: Generates coarse structure first, then adds fine details at higher resolution. Tradeoff: more complex architecture and training.
- **Adversarial loss**: Add a discriminator to encourage perceptually sharp outputs. Tradeoff: training instability and risk of mode collapse.

Training Progress: Reconstructions from CINIC-10

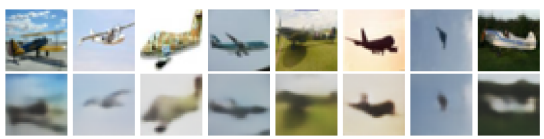
Configuration: CINIC-10 dataset, Gaussian likelihood, latent dim 128, β annealed from 4 (warmup) to 5 (full training).

Each panel shows original validation images (top row) and their reconstructions via $\mu_\theta(\mathbf{z})$ where $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ (bottom row).

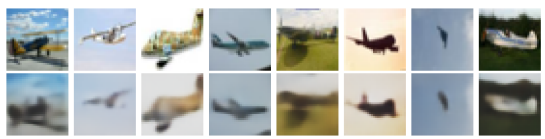
Epoch 1 (warmup, $\beta = 4$)



Epoch 25 (warmup, $\beta = 4$)



Epoch 50 (full, $\beta = 5$)



Sampling: Why We Return Only the Mean

Generation process:

1. Sample latent code: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2. Decode to image: $\mathbf{x} = \mu_\theta(\mathbf{z})$ (no noise added)

Why not add noise?

- If we sampled $\mathbf{x} \sim \mathcal{N}(\mu_\theta(\mathbf{z}), \sigma^2_\theta(\mathbf{z}))$, we'd add random noise to each pixel
- This makes images *blurrier*, not sharper
- The variance $\sigma^2_\theta(\mathbf{z})$ measures uncertainty about *which* image should appear — it's useful for understanding reconstructions, but adding it would degrade quality

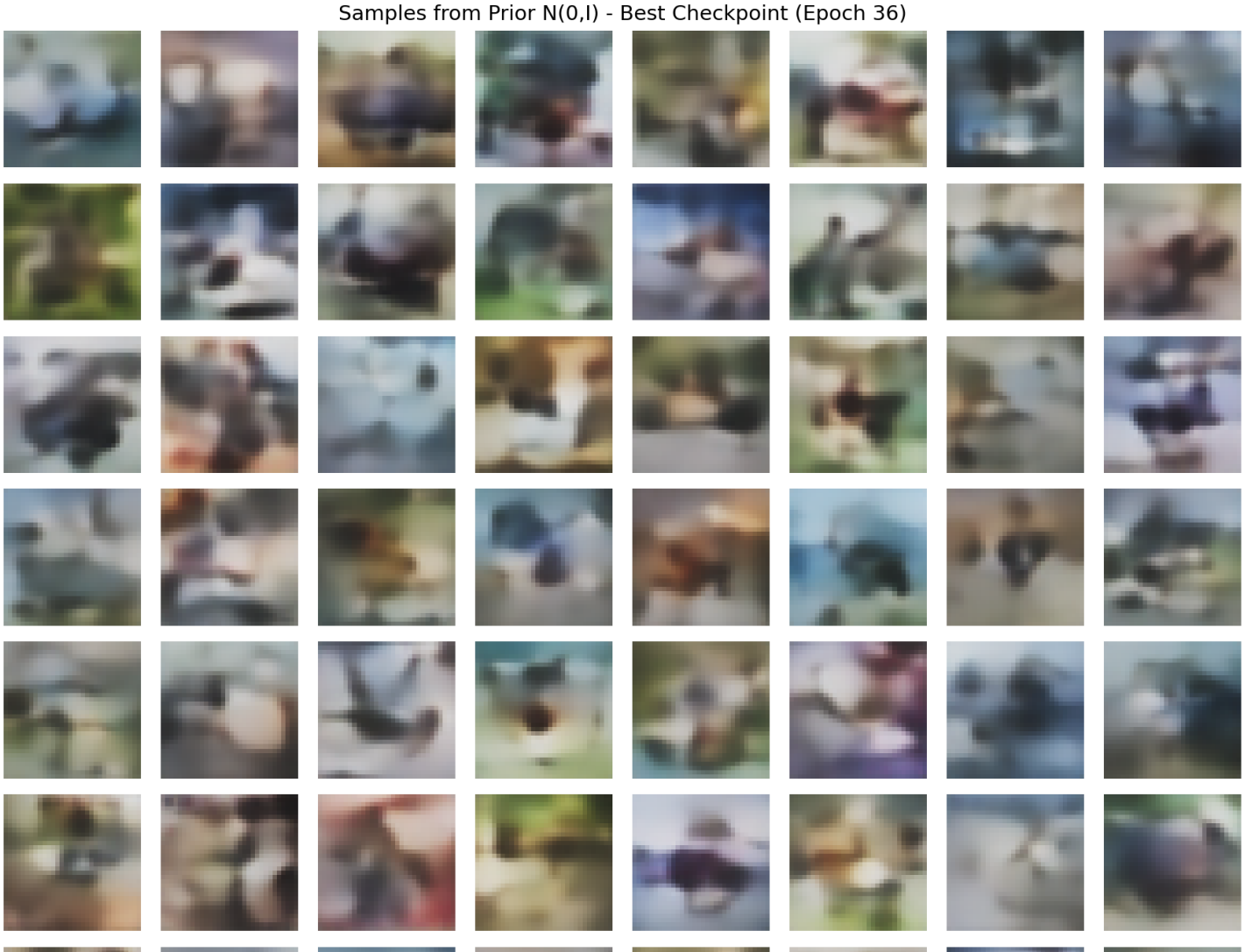
Standard practice: All VAE implementations return only $\mu_\theta(\mathbf{z})$ for generation. To get truly sharp images with continuous data, we need either:

- VQ-VAE with discrete latents
- VAE-GAN hybrids

(Alternatively: Bernoulli likelihood for binary data)

Samples Drawn from the Prior After Training

Generation: Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, decode to $\mu_\theta(\mathbf{z})$ (decoder mean). Best checkpoint (epoch 36, highest validation ELBO) from CINIC-10 training.



Beyond Basic VAEs: Extensions and Improvements

The Gaussian VAE we've derived is the simplest case. Key limitations and solutions:

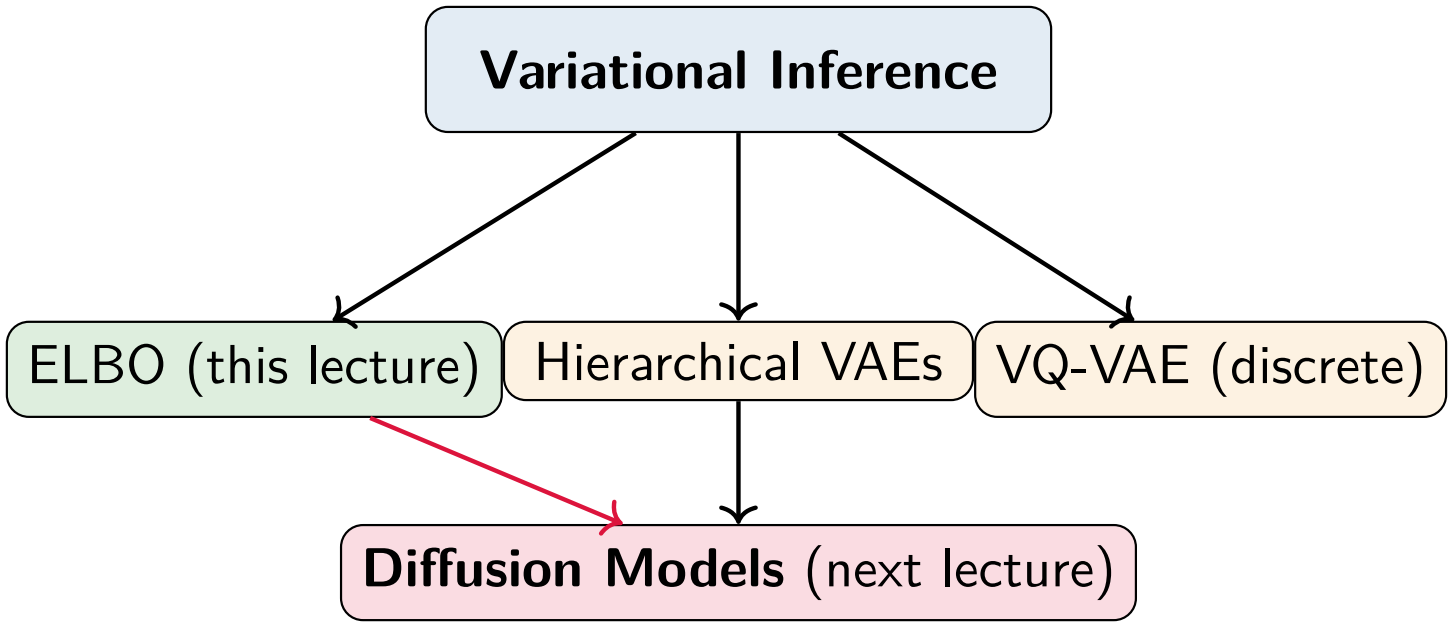
1. **Blurriness (from slide 28):**
   - *Cause:* Gaussian likelihood averages over plausible modes
   - *Solutions:* Discrete latent codes (VQ-VAE), autoregressive decoders, adversarial loss
2. **Posterior collapse:**
   - *Cause:* Decoder ignores latents (KL goes to zero)
   - *Solutions:* KL annealing, β-VAE (weighted KL), skip connections
3. **Limited expressivity:**
   - *Cause:* Single Gaussian prior is too simple for complex data
   - *Solutions:* Hierarchical VAEs (Ladder VAE, NVAE), normalizing flows for prior

Next lecture: Variational diffusion models — progressively denoise via latent Markov chain.

Wrap-up and Future Directions

Variational Inference

The ELBO is one instance of the broader framework of **Variational Inference**, an approach to machine learning that approximates intractable posteriors with tractable distributions.



Common thread: All methods approximate $p(\mathbf{z}|\mathbf{x})$ with a tractable q and optimize a lower bound on the evidence.

Lecture Summary

1. **Latent variable models** enable efficient sampling by decoupling representation from generation
2. **Maximum likelihood is intractable** due to high-dimensional integration over neural networks
3. **The ELBO provides a tractable lower bound** with reconstruction and KL terms in tension
4. **Gradient estimation requires care:** Reparameterization enables low-variance gradients for Gaussian encoders
5. **Architectural choices matter:** Encoder/decoder design, single-sample MC, and Gaussian assumptions all affect quality
6. **VAEs are one instance** of the broader variational inference framework, with extensions addressing blur, collapse, and expressivity

Questions?

Next: Variational Diffusion Models

Continuous-time latent variable models via stochastic processes