Variational Autoencoders (continued)

ECE 57000 — November 9, 2026

David I. Inouye

The encoder helps us fit a latent generative model

- **Generate:** sample a code from the prior, then an image from the decoder.
- **Infer:** given an image, the encoder approximates its posterior over codes.
- **Train:** maximize an ELBO that we can estimate without the difficult image marginal.

$$p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{z}) p_\theta(\boldsymbol{x} \mid \boldsymbol{z}), \qquad q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \approx p_\theta(\boldsymbol{z} \mid \boldsymbol{x}).$$

Today: turn the bound into a loss, compute gradients, then inspect what the trained model can do.

The ELBO gives a lower bound on image log likelihood

For one image, the following identity connects inference and generation:

$$\log p_\theta(\boldsymbol{x}) = \underbrace{\mathcal{L}(\theta, \phi; \boldsymbol{x})}_{\text{ELBO}} + \underbrace{D_{\mathrm{KL}}(q_\phi(\cdot \mid \boldsymbol{x}) \,\|\, p_\theta(\cdot \mid \boldsymbol{x}))}_{\text{gap from approximate inference}}.$$

Because KL is nonnegative, $\mathcal{L}(\theta, \phi; \boldsymbol{x}) \leq \log p_\theta(\boldsymbol{x})$.

- Fit θ and ϕ by maximizing this **evidence lower bound**.
- The bound is tight when the encoder equals the true posterior.

The VAE loss balances reconstruction and agreement with the prior

Factor $p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{z})p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ inside the ELBO:

$$\mathcal{L} = \mathbb{E}_q[\log p_\theta(\boldsymbol{x} \mid \mathbf{z})] + \mathbb{E}_q\left[\log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z} \mid \boldsymbol{x})}\right].$$

Minimize the negative ELBO:

$$\mathcal{J} = -\mathcal{L} = \underbrace{\mathbb{E}_q[-\log p_\theta(\boldsymbol{x} \mid \mathbf{z})]}_{\text{reconstruction penalty}} + \underbrace{D_{\mathrm{KL}}(q_\phi(\cdot \mid \boldsymbol{x}) \,\|\, p(\cdot))}_{\text{prior regularization}}.$$

Two different KL terms	Meaning
$D_{\mathrm{KL}}(q_\phi(\cdot \mid \boldsymbol{x}) \| p_\theta(\cdot \mid \boldsymbol{x}))$	**True posterior:** inference error for this image
$D_{\mathrm{KL}}(q_\phi(\cdot \mid \boldsymbol{x}) \| p(\cdot))$	**Prior:** regularization toward shared sampling codes

β controls how strongly codes must agree with the prior

$$\mathcal{J}_\beta = \underbrace{\mathbb{E}_q[-\log p_\theta(\boldsymbol{x} \mid \mathbf{z})]}_{\text{reconstruction}} + \beta \underbrace{D_{\mathrm{KL}}(q_\phi(\cdot \mid \boldsymbol{x}) \| p(\cdot))}_{\text{prior regularization}}.$$

Weight	What we optimize
$\beta = 0$	Probabilistic autoencoder: reconstruction alone
$0 < \beta < 1$	Weaker prior regularization
$\beta = 1$	Standard VAE: **exactly the negative ELBO**
$\beta > 1$	Stronger prior regularization (**β-VAE**)

More prior pressure can reduce information about the image and worsen reconstruction.

Side point: as $\beta \to \infty$, $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = p(\boldsymbol{z})$ gives zero KL. A sufficiently expressive decoder could still fit images with $p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = p_\theta(\boldsymbol{x})$, although $\mathbf{x}$ and $\mathbf{z}$ are independent.

A Gaussian decoder turns likelihood into scaled squared error

Part 1 of the VAE objective: reconstruction.

Let the decoder predict an image mean $g_\theta(\boldsymbol{z})$ with fixed variance τ^2:

$$p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}\left(\boldsymbol{x}; g_\theta(\boldsymbol{z}), \tau^2 I\right).$$

Then, for a d-feature image,

$$-\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = \frac{\|\boldsymbol{x} - g_\theta(\boldsymbol{z})\|_2^2}{2\tau^2} + \frac{d}{2}\log(2\pi\tau^2).$$

- τ is the **decoder observation spread**; $\boldsymbol{\sigma}$ describes encoder codes.
- With fixed τ, the last term is constant during training.
- Changing the error scale changes its balance with KL.

For Gaussian codes, we can calculate KL directly

Part 2 of the VAE objective: regularization. Compare the **encoder's approximate posterior** with the **prior**:

$$q_\phi(\cdot \mid \boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{\mu}_\phi(\boldsymbol{x}), \operatorname{diag}(\boldsymbol{\sigma}_\phi^2(\boldsymbol{x}))\right), \qquad p(\cdot) = \mathcal{N}(\mathbf{0}, I).$$

$$D_{\mathrm{KL}}(q_\phi(\cdot \mid \boldsymbol{x}) \,\|\, p(\cdot)) = \frac{1}{2} \sum_{j=1}^{k} \left(\mu_j^2 + \sigma_j^2 - 1 - \log \sigma_j^2\right).$$

Here $\boldsymbol{\mu} = \boldsymbol{\mu}_\phi(\boldsymbol{x})$ and $\boldsymbol{\sigma} = \boldsymbol{\sigma}_\phi(\boldsymbol{x})$ are this image's encoder outputs.

- Mean $\mathbf{0}$ and spread $\mathbf{1}$ match the prior and give zero KL.
- This term is analytic: **no latent-code sampling is needed**.

The encoder changes the distribution inside the expectation

To maximize the ELBO, we need $\nabla_{\theta,\phi}\mathcal{L}$ (or minimize $\mathcal{J} = -\mathcal{L}$). The Gaussian KL is explicit; reconstruction still needs an expectation.

Let $\ell_\theta(\boldsymbol{z}) = -\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ for fixed $\boldsymbol{x}$:

$$\nabla_\phi \mathbb{E}_{\mathbf{z}\sim q_\phi}[\ell_\theta(\mathbf{z})] = \nabla_\phi \int q_\phi(\boldsymbol{z} \mid \boldsymbol{x})\ell_\theta(\boldsymbol{z})\, d\boldsymbol{z} \neq \mathbb{E}_{q_\phi}[\nabla_\phi \ell_\theta(\mathbf{z})] \quad \text{in general.}$$

- At a fixed code, ℓ_θ has no direct ϕ dependence; the naive right side is zero.
- But changing the encoder changes **which codes are likely**.
- We need to move ϕ into a differentiable function of **parameter-independent noise**.

Reparameterization makes sampling a differentiable calculation

For a diagonal Gaussian encoder, an **exact alternative sampling procedure** is

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I), \qquad \boldsymbol{z} = \boldsymbol{\mu}_\phi(\boldsymbol{x}) + \boldsymbol{\sigma}_\phi(\boldsymbol{x}) \odot \boldsymbol{\epsilon}.$$

1. Draw noise from a distribution independent of the encoder parameters.
2. Scale it by the encoder's standard deviations.
3. Shift it by the encoder's means.

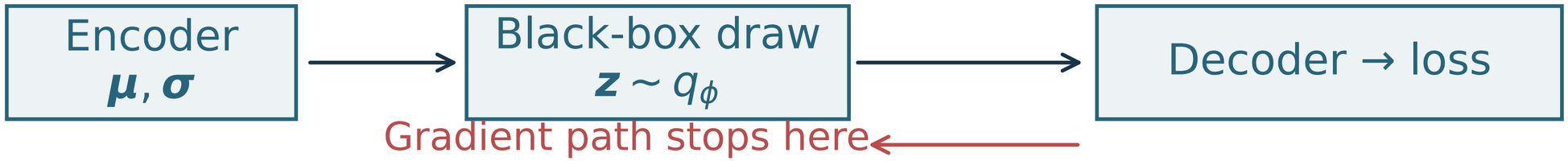
The transformed code has exactly distribution $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$:

$$\nabla_\phi \mathbb{E}_{q_\phi}[\ell_\theta(\mathbf{z})] = \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I)}[\nabla_\phi \ell_\theta(\boldsymbol{\mu}_\phi + \boldsymbol{\sigma}_\phi \odot \boldsymbol{\epsilon})] \,.$$

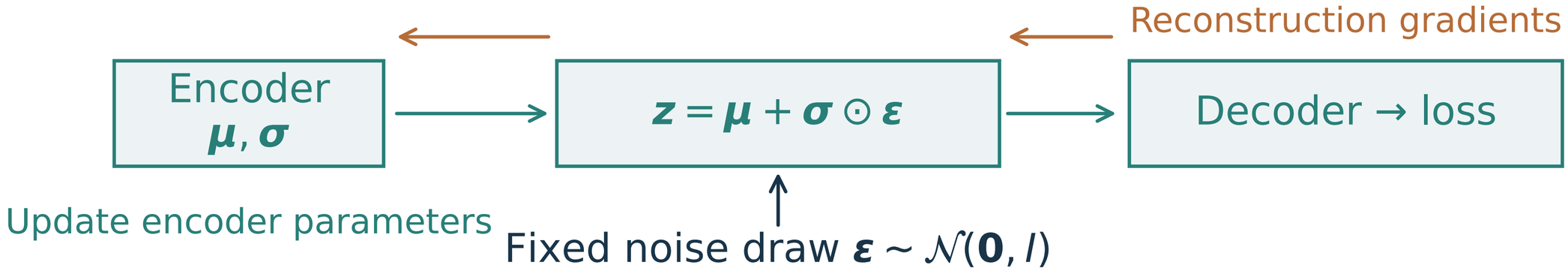
Now the noise distribution is fixed. More complex reparameterizations exist; this Gaussian case is the simplest common one.

Fixed noise lets backpropagation follow the encoder-to-loss path

Direct sampling: no ordinary pathwise derivative through the draw



Reparameterized sampling: differentiate the scale and shift



Hold ϵ fixed **for this gradient calculation**; differentiate the scale and shift, not the random draw.

Monte Carlo sampling estimates the expectation and its gradient

Draw M independent noises and form codes **via the reparameterization trick**:

$$\boldsymbol{\epsilon}^{(m)} \sim \mathcal{N}(\mathbf{0}, I), \qquad \boldsymbol{z}^{(m)} = \boldsymbol{\mu}_\phi(\boldsymbol{x}) + \boldsymbol{\sigma}_\phi(\boldsymbol{x}) \odot \boldsymbol{\epsilon}^{(m)}.$$

$$\widehat{\mathcal{J}}_\beta = \underbrace{\frac{1}{M}\sum_{m=1}^{M} -\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}^{(m)})}_{\text{Monte Carlo reconstruction estimate}} + \beta \underbrace{D_{\mathrm{KL}}(q_\phi(\cdot \mid \boldsymbol{x}) \| p(\cdot))}_{\text{closed-form Gaussian KL}}.$$

- Backpropagate $\nabla_{\theta,\phi}\widehat{\mathcal{J}}_\beta$; at $\beta = 1$, it estimates $-\nabla_{\theta,\phi}\mathcal{L}$.
- **Usually $M = 1$ per image**; average over a minibatch and draw fresh noise each update.
- Reparameterization supplies the gradient path; Monte Carlo approximates the integral.

Each minibatch trains the encoder and decoder together

For a batch $\{\boldsymbol{x}_i\}_{i=1}^{B}$, compute all B examples in parallel:

Step	Batched computation (each $i = 1, \dots, B$)
1. Encode	$(\boldsymbol{\mu}_i, \boldsymbol{s}_i) = f_\phi(\boldsymbol{x}_i)$; $\boldsymbol{s}_i$ is log variance
2. Sample codes	$\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, I)$; $\boldsymbol{z}_i = \boldsymbol{\mu}_i + \exp(\boldsymbol{s}_i/2) \odot \boldsymbol{\epsilon}_i$
3. Decode and score	$r_i = -\log p_\theta(\boldsymbol{x}_i \mid \boldsymbol{z}_i)$; $k_i = \frac{1}{2}\sum_j(\mu_{ij}^2 + e^{s_{ij}} - 1 - s_{ij})$
4. Average the loss	$\widehat{\mathcal{J}}_\beta = \frac{1}{B}\sum_{i=1}^{B}(r_i + \beta k_i)$
5. Update both networks	$\theta \leftarrow \theta - \eta\nabla_\theta\widehat{\mathcal{J}}_\beta$; $\phi \leftarrow \phi - \eta\nabla_\phi\widehat{\mathcal{J}}_\beta$

One fresh code per image. At $\beta = 1$, the batch loss estimates the average negative ELBO.

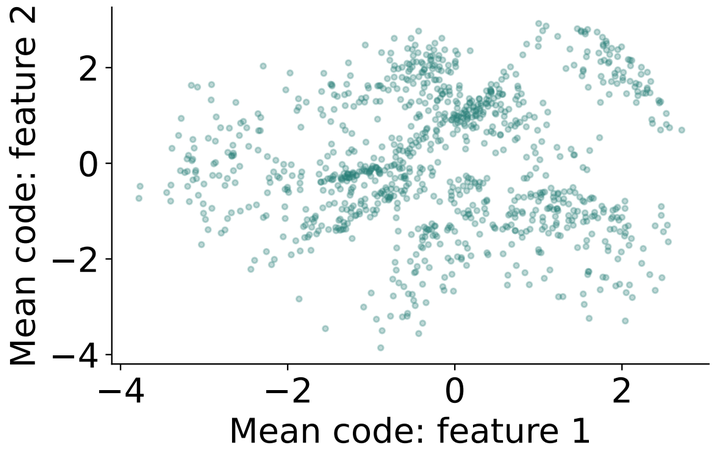
What happens if we train using only the KL term?

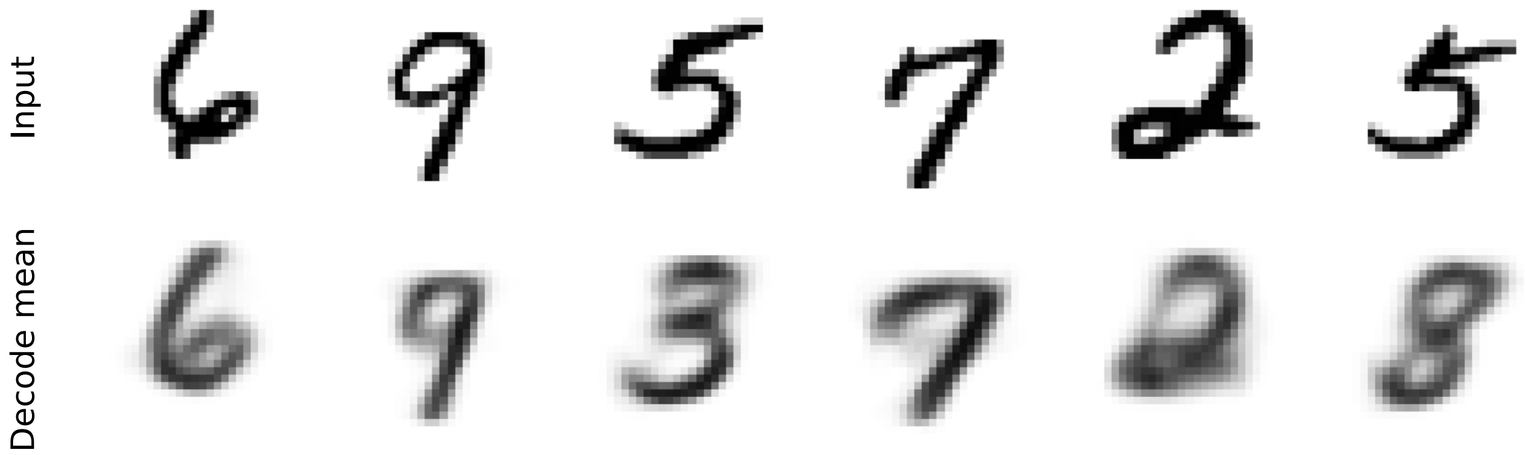
The prior is $\mathcal{N}(0, I)$ and the encoder can represent it exactly.
Which encoder minimizes KL for every image?
A. A different narrow distribution for each image.
B. A point concentrated at zero for every image.
C. The same standard Gaussian for every image.

C. Matching both center and spread gives zero KL; that encoder distribution carries no image-specific information.

After training, the encoder provides representations and reconstructions

- **Represent:** $x \mapsto \mu_\phi(x)$ gives a feature vector for downstream tasks.
- **Reconstruct:** encode an image, then decode its mean code (or a sampled code).

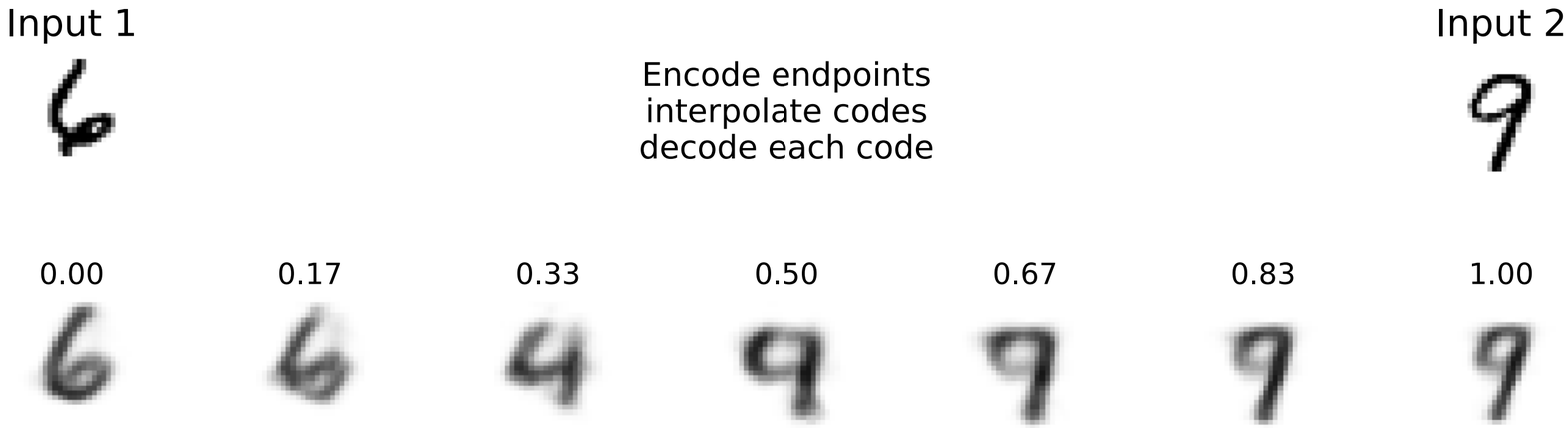




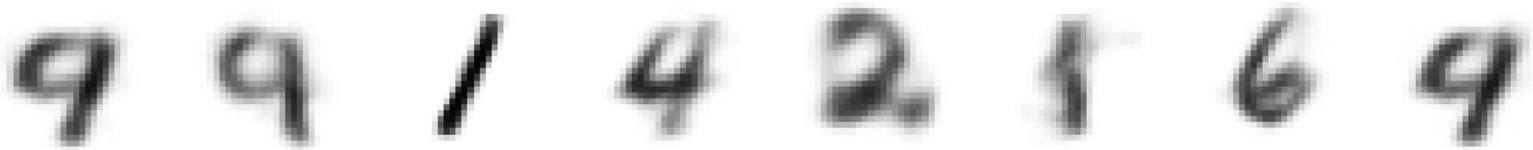
Selected held-out MNIST examples; this demo uses a **Bernoulli decoder**. Reconstructions can lose detail.

Interpolation uses encoded endpoints; generation needs only the decoder

Interpolate: encode two images, then decode $z(a) = (1-a)\mu_1 + a\mu_2$.



Generate: draw $z \sim p(z)$ and decode; **no input image or encoder** is needed.



Decoder means shown; smooth interpolation does not guarantee good prior samples.

A trained VAE still needs evidence for its intended use

Intended use	Evidence to inspect
Reconstruct an observed image	Held-out reconstruction under the stated loss
Generate new images	Fresh prior samples: quality, diversity and failure cases
Use the learned code for another task	Held-out performance on that downstream task

Audit the assumptions: chosen prior, decoder likelihood and approximate-posterior family.

A good training objective gives a learning procedure; it does not guarantee all three uses succeed.

Diffusion returns to denoising at many noise levels

Two ways to generate an image:

1. **VAE:** sample a latent code, then decode an image.
2. **Diffusion:** start from noise and form an image through repeated denoising steps.

The next model learns how to denoise at different noise levels, using the noise level as an input.

Remember the denoising AE: corrupted input, clean-image information as the training signal.

Generating images by diffusion

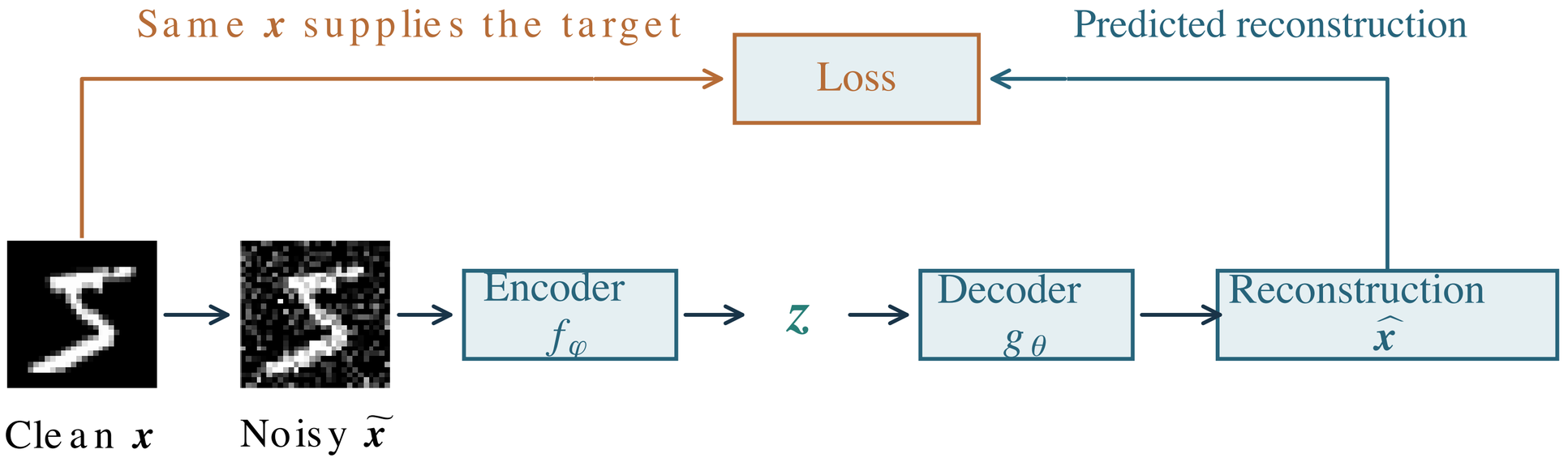
David I. Inouye

Dynamic Content · Last Updated: November 9, 2026

Arc 7 · Image generation and evaluation

Denoising → the circle example → many noise levels → reverse generation

A denoising autoencoder predicts the clean input



MNIST training image 0; reconstruction is a symbol, not a trained output.

$$\boldsymbol{y} = \boldsymbol{x} + \sigma\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I), \qquad \widehat{\boldsymbol{x}} = g_\theta(f_\phi(\boldsymbol{y})).$$

Input: corrupted image. **Target:** original clean image.

Denoising needs a predictor, not a required bottleneck

$$\widehat{\boldsymbol{x}} = D_\theta(\boldsymbol{y}), \qquad \min_\theta \ \mathbb{E}\left[\|\boldsymbol{x} - D_\theta(\boldsymbol{x} + \sigma\boldsymbol{\epsilon})\|^2\right].$$

- Directly map a noisy image to a clean-image prediction of the **same shape**.
- We can use any suitable neural network; an explicit latent code is optional.
- An encoder–decoder or U-Net can still be a useful architecture.

Predicting the clean image and predicting its noise are equivalent

For $\boldsymbol{y} = \boldsymbol{x} + \sigma\boldsymbol{\epsilon}$ at a **fixed** $\sigma > 0$:

$$\widehat{\boldsymbol{x}} = \boldsymbol{y} - \sigma\epsilon_\theta(\boldsymbol{y}), \qquad \hat{\boldsymbol{\epsilon}} = \frac{\boldsymbol{y} - D_\theta(\boldsymbol{y})}{\sigma}.$$

The squared errors differ by a constant:

$$\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|^2 = \sigma^2 \|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}\|^2.$$

Same prediction problem; a different output parameterization.