Generative Models: Autoencoders, Adversarial Networks, and Diffusion

October 22, 2025

Contents

1 Autoencoders (AE, VAE)

Autoencoders learn latent representations by reconstructing inputs. The encoder f_ϕ maps $\mathbf{x}$ to latent code $\mathbf{z}$, and the decoder g_θ reconstructs $\hat{\mathbf{x}}$. Figure **??** illustrates the bottleneck structure.

1.1 Deterministic Autoencoders

The reconstruction loss (often mean squared error) is

$$\mathcal{L}_{\mathrm{AE}}(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} \left\| g_\theta\big(f_\phi(\mathbf{x}_i)\big) - \mathbf{x}_i \right\|_2^2 . \tag{1}$$

While AEs excel at dimensionality reduction, latent spaces may be discontinuous. Regularized variants enforce structure:

- **Sparse AE:** Adds L_1 penalties on activations to encourage sparse latent representations.
- **Denoising AE:** Trains with corrupted inputs $\tilde{\mathbf{x}}$ but reconstructs the clean $\mathbf{x}$, enhancing robustness.
- **Contractive AE:** Penalizes the Jacobian $\|\nabla_{\mathbf{x}} f_\phi(\mathbf{x})\|_F^2$ to learn invariant features.

1.2 Variational Autoencoders

VAEs impose a probabilistic latent variable model. Given prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and decoder likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$, the evidence lower bound (ELBO) for each sample is

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x} \mid \mathbf{z})\big] - \mathrm{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\big). \tag{2}$$

Parameterizing $q_\phi(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}\big(\boldsymbol{\mu}_\phi(\mathbf{x}), \mathrm{diag}(\boldsymbol{\sigma}_\phi^2(\mathbf{x}))\big)$ enables reparameterization:

$$\mathbf{z} = \boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\sigma}_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{3}$$

The KL term has closed form:

$$\mathrm{KL} = -\tfrac{1}{2} \sum_{j=1}^{d} \left(1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2\right). \tag{4}$$

1.3 Beta-VAE and Disentanglement

β-VAE scales the KL divergence, $\mathcal{L} = \mathbb{E}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] - \beta\, \mathrm{KL}(q_\phi \| p)$, promoting disentangled latent factors when $\beta > 1$. Mutual information penalties and Total Correlation regularizers (TC-VAE) refine latent independence.

1.4 Training Procedure

Listing 1: Variational autoencoder training loop with KL annealing.

```
for epoch in range(num_epochs):
    beta = min(1.0, epoch / kl_warmup_epochs)
    for x in dataloader:
        mu, logvar = encoder(x)
        z = mu + torch.exp(0.5 * logvar) * torch.randn_like(mu)
        x_recon = decoder(z)
        recon_loss = reconstruction_criterion(x_recon, x)
        kl = -0.5 * (1 + logvar - mu.pow(2) - logvar.exp()).sum(dim=1).mean()
        loss = recon_loss + beta * kl
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```

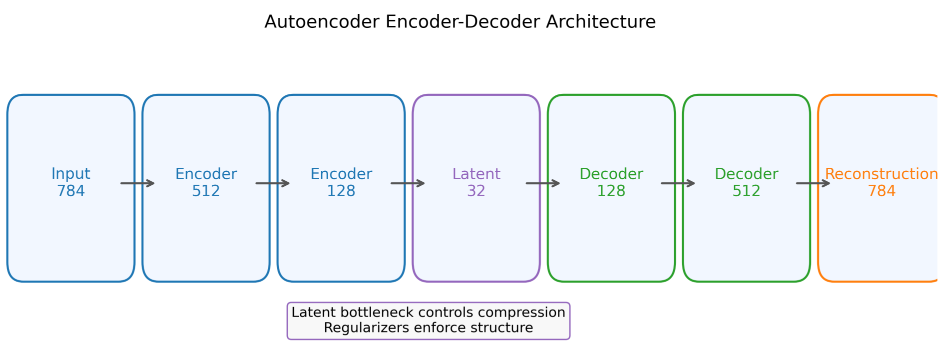

Figure 1: Autoencoder architecture with deterministic encoder/decoder. Bottleneck dimensionality controls compression.

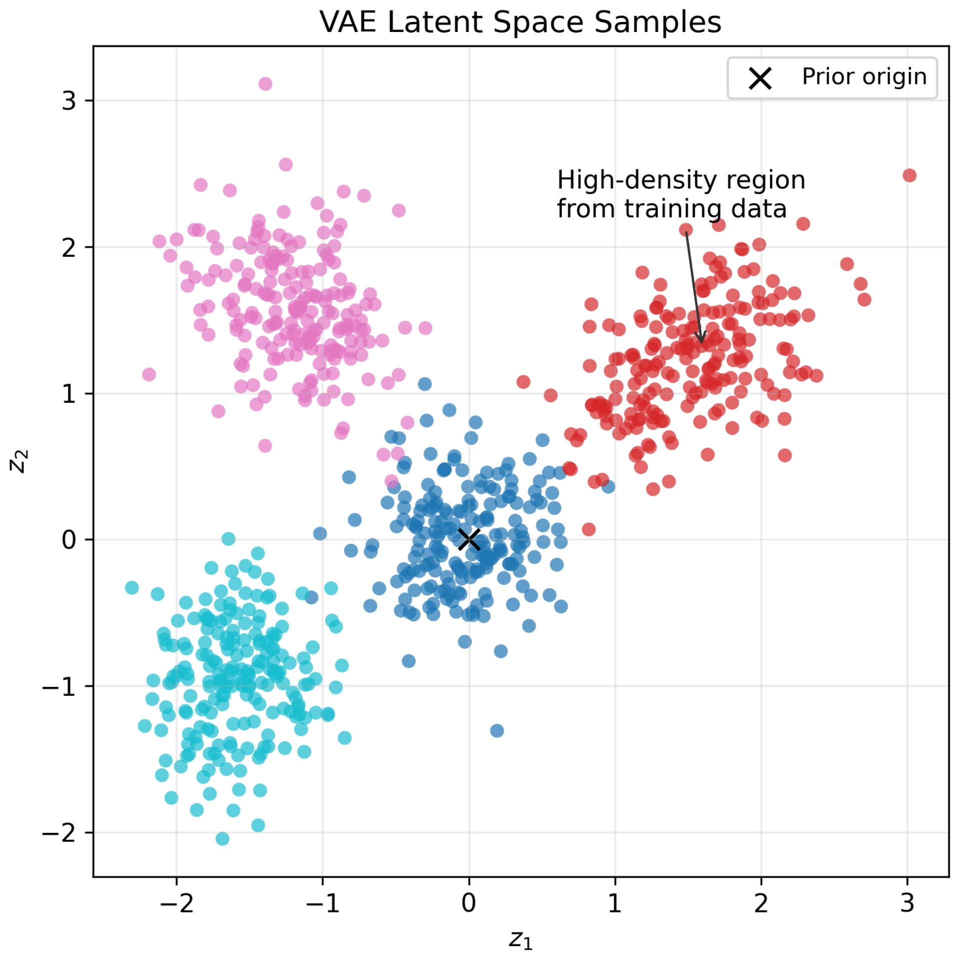

Figure 2: Latent space sampling in a VAE. Points near the origin correspond to plausible reconstructions.

2 Generative Adversarial Networks (GANs)

GANs pit a generator G against a discriminator D. The min-max objective,

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\big[\log D(\mathbf{x})\big] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\big[\log(1 - D(G(\mathbf{z})))\big], \tag{5}$$

drives G to synthesize realistic samples. Figure **??** depicts loss curves and mode coverage.

2.1 GAN Variants

- **DCGAN:** Convolutional architectures with strided convolutions, batch normalization, and ReLU/Leaky ReLU activations for stable image synthesis.
- **WGAN and WGAN-GP:** Replace Jensen-Shannon divergence with Earth-Mover distance. The WGAN objective

  $$\min_G \max_{D\in\mathcal{D}_1} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[D(G(\mathbf{z}))], \tag{6}$$

  constrains D to 1-Lipschitz. Gradient penalty adds $\lambda(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2$ to encourage Lipschitz continuity.

- **StyleGAN:** Introduces style-based modulation. Latent $\mathbf{w}$ modulates convolution kernels via adaptive instance normalization, enabling control over coarse-to-fine features. Path-length regularization and noise injection improve fidelity.

2.2 Training Stabilization

Mode collapse, vanishing gradients, and discriminator overpowering require safeguards:

- Feature matching and minibatch discrimination regularize G.
- Spectral normalization enforces Lipschitz constraints by rescaling weight matrices.
- Two-time-scale update rule (TTUR) adjusts learning rates (η_D, η_G) to balance convergence.

2.3 Evaluation Metrics

Fréchet Inception Distance (FID) approximates the Wasserstein-2 distance between Inception features:

$$\mathrm{FID} = \|\boldsymbol{\mu}_r - \boldsymbol{\mu}_g\|_2^2 + \mathrm{Tr}\left(\boldsymbol{\Sigma}_r + \boldsymbol{\Sigma}_g - 2(\boldsymbol{\Sigma}_r\boldsymbol{\Sigma}_g)^{1/2}\right). \tag{7}$$

Precision/recall for generative models and Inception Score complement FID.

2.4 StyleGAN2 Generator Forward Pass

Listing 2: Simplified StyleGAN2 generator block with style modulation.

```python
class StyledConv(nn.Module):
    def __init__(self, in_channels, out_channels, style_dim, upsample):
        super().__init__()
        self.upsample = upsample
        self.weight = nn.Parameter(torch.randn(1, out_channels, in_channels, 3, 3))
        self.modulation = nn.Linear(style_dim, in_channels)
        self.noise_weight = nn.Parameter(torch.zeros(1, out_channels, 1, 1))
        self.activation = nn.LeakyReLU(0.2)

    def forward(self, x, style, noise):
        style = self.modulation(style).view(-1, 1, x.size(1), 1, 1)
        weight = self.weight * (style + 1e-8)
        if self.upsample:
            x = upsample_2x(x)
        x = conv2d_modulated(x, weight)
        x = x + self.noise_weight * noise
        return self.activation(x)
```

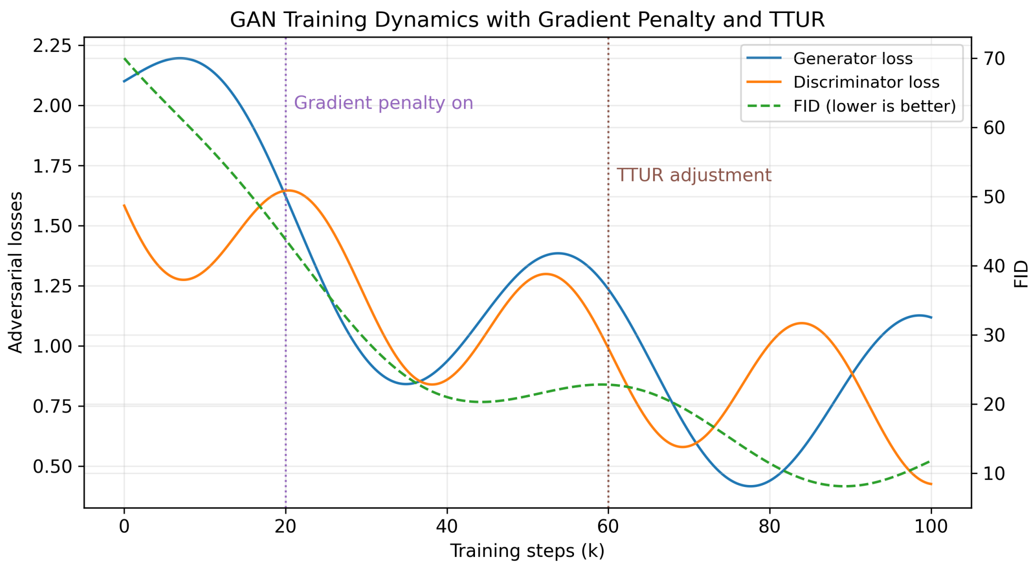

Figure 3: Generator and discriminator loss trajectories with gradient penalty and TTUR scheduling.

3 Diffusion Models Overview

Diffusion models generate data by reversing a gradual noising process. The forward process corrupts data $\mathbf{x}_0$ into $\mathbf{x}_T$ using a variance schedule $\{\beta_t\}_{t=1}^T$:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\big(\sqrt{1-\beta_t}\,\mathbf{x}_{t-1}, \beta_t \mathbf{I}\big). \tag{8}$$

Due to Gaussian composition,

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\big(\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}\big), \quad \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s). \tag{9}$$

3.1 Denoising Diffusion Probabilistic Models (DDPM)

The model learns $\epsilon_\theta(\mathbf{x}_t, t)$ to predict noise. The simplified training objective is

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon} - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, t)\|_2^2\right]. \tag{10}$$

Sampling begins from $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and iteratively denoises:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{11}$$

Figure **??** visualizes forward and reverse trajectories.

3.2 Improvements and Variants

- **Guided diffusion:** Classifier guidance adjusts sampling drift, $\hat{\epsilon} = \epsilon_\theta - \sigma_t \nabla_{\mathbf{x}_t} \log p_\phi(y \mid \mathbf{x}_t)$, improving class-conditional fidelity.
- **Score-based generative models:** Learn $\nabla_{\mathbf{x}} \log q_t(\mathbf{x})$ with stochastic differential equations (SDEs) and integrate using predictor-corrector samplers.
- **Latent diffusion:** Compress data into latent space via VAE before diffusion (e.g., Stable Diffusion), reducing computational cost while leveraging cross-attention for conditioning.

3.3 Pseudo-code

Listing 3: Diffusion training step with cosine noise schedule.

```
def diffusion_training_step(model, scheduler, x0):
    t = torch.randint(0, scheduler.num_steps, (x0.size(0),), device=x0.device)
    noise = torch.randn_like(x0)
    alpha_bar = scheduler.alpha_bar(t).view(-1, 1, 1, 1)
    xt = torch.sqrt(alpha_bar) * x0 + torch.sqrt(1 - alpha_bar) * noise
    noise_pred = model(xt, t)
    loss = (noise - noise_pred).pow(2).mean()
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    return loss
```

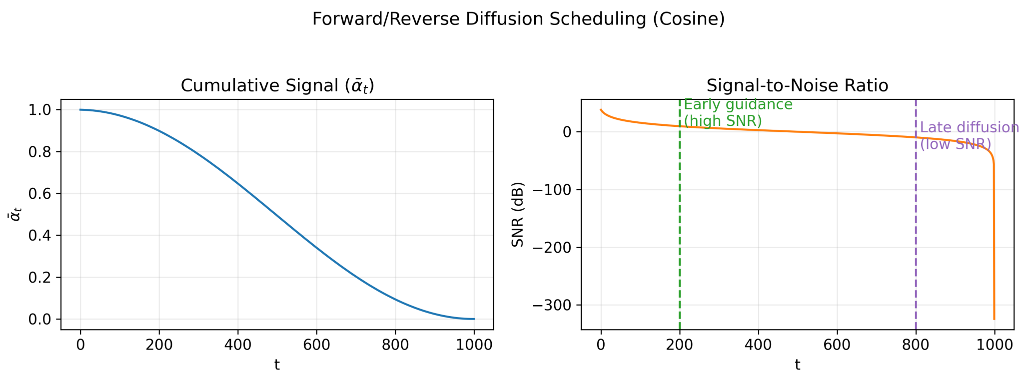

Figure 4: Forward noising and reverse denoising trajectories in a diffusion model with cosine schedule.

Further Reading

- Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes." ICLR 2014.
- Ian Goodfellow et al. "Generative Adversarial Networks." NIPS 2014.
- Tero Karras et al. "Analyzing and Improving the Image Quality of StyleGAN." CVPR 2020.
- Jonathan Ho et al. "Denoising Diffusion Probabilistic Models." NeurIPS 2020.
- Prafulla Dhariwal and Alexander Nichol. "Diffusion Models Beat GANs on Image Synthesis." NeurIPS 2021.