CHAPTER 17

Representation Learning and Generative Learning Using Autoencoders and GANs

Chapter 17 presents **autoencoders** and **GANs** as key tools for unsupervised representation learning and generative modeling, showing how to learn compact latent spaces and synthesize realistic data such as images.

Autoencoder fundamentals and uses

An **autoencoder** consists of an encoder that maps inputs to a latent **coding**, and a decoder that reconstructs the inputs from this coding; it is trained to minimize reconstruction loss (e.g., MSE) on unlabeled data. Main uses:

- **Dimensionality reduction & visualization**: undercomplete autoencoders (bottleneck smaller than input) learn compressed representations similar to PCA, but can be nonlinear and scale to large datasets.
- **Feature learning & unsupervised pretraining**: stacked autoencoders can pretrain deep nets when labeled data is scarce, by reusing encoder layers as initialization for supervised models.
- **Generative modeling & denoising**: certain variants (e.g., VAEs) can generate realistic new samples; denoising autoencoders learn robust representations by reconstructing clean inputs from noisy versions.

A **linear** undercomplete autoencoder with MSE essentially performs PCA on the data.

Stacked, tied, denoising, and sparse autoencoders

Stacked autoencoders add multiple hidden layers (symmetrical around the bottleneck) to capture more complex structure, often with nonlinear activations like ReLU or SELU. Techniques:

- **Weight tying**: decoder weights set to the transpose of encoder weights, reducing parameters and encouraging symmetric mappings; implemented by sharing variables or using DenseTranspose-style layers.
- **Greedy layer-wise training**: train shallow autoencoders one at a time on successively encoded data, then stack them; used historically to stabilize deep training before modern methods.
- **Denoising autoencoders**: corrupt inputs with noise (e.g., masking, Gaussian) and train to reconstruct the original clean input, forcing the model to learn robust structure and act as a regularizer.

- **Sparse autoencoders**: use large hidden layers but enforce sparsity via L1 regularization or KL-divergence penalties on average activations, encouraging each neuron to specialize on rare patterns.

These variants help avoid trivial identity mappings and encourage meaningful latent representations.

Variational autoencoders (VAEs)

VAEs are probabilistic autoencoders that learn a smooth latent **distribution** rather than point codings, making them powerful generative models. Key ideas:

- Encoder outputs parameters of a Gaussian distribution per input (mean μ and log-variance $\log \sigma^2$), defining $q_\phi(z \mid x)$.
- Latent sample via **reparameterization trick**: $z = \mu + \sigma \odot \epsilon$ with $\epsilon \sim \mathcal{N}(0, I)$ so gradients can flow through randomness.
- Decoder defines $p_\theta(x \mid z)$; training minimizes **reconstruction loss** plus a **KL divergence** term that pushes $q_\phi(z \mid x)$ towards a unit Gaussian prior $p(z)$.

Loss per example:

- For continuous data: $\mathrm{MSE}(x, \hat{x}) + \beta\,\mathrm{KL}(q_\phi(z \mid x) \mid\mid \mathcal{N}(0, I))$ with optional scaling β.

After training, sampling $z \sim \mathcal{N}(0, I)$ and decoding produces new data points (e.g., novel Fashion-MNIST images), and interpolating between two latent codes yields smooth transitions between samples.

Generative adversarial networks (GANs)

A **GAN** consists of a **generator** G that maps random noise z to fake samples $G(z)$, and a **discriminator** D that tries to distinguish real samples from fake; they are trained in an adversarial min–max game. Basic training:

- Update D to maximize $\log D(x) + \log(1 - D(G(z)))$.
- Update G to maximize $\log D(G(z))$ (or equivalently minimize $\log(1 - D(G(z)))$).

Applications include **image synthesis**, super-resolution, inpainting, style transfer, data augmentation, and more.

Training challenges and remedies:

- **Mode collapse**: generator produces limited variety; mitigations include minibatch discrimination, feature matching, and architectural tweaks.
- **Training instability/oscillation**: addressed by improved losses (e.g., Wasserstein GANs with gradient penalty), careful architecture (DCGAN guidelines), and learning-rate/batch-size tuning.

- **Overpowerful discriminator**: can be tempered by label smoothing, noisy labels, or updating G more frequently than D.

DCGAN adds convolutional structure and practical rules (strided convs, transposed convs, BatchNorm, ReLU in G and LeakyReLU in D) to generate convincing medium-resolution images (e.g., faces, bedrooms).

Advanced GAN variants: Progressive GAN, StyleGAN

To generate **high-resolution, coherent images**, the chapter reviews more advanced designs:

- **Progressive Growing of GANs**: start with very low-res images (e.g., 4×4), then progressively add layers to G and D to reach higher resolutions (8×8 → 16×16 → … → 1024×1024), smoothly fading in new layers; stabilizes training and improves global structure.
- **StyleGAN**: refines the generator with:
    - A **mapping network** that converts latent code z into a **style vector** w, then applies per-layer affine transforms (AdaIN) to control feature scales and biases, giving fine-grained control over attributes like pose, hairstyle, or lighting.
    - Per-layer **noise inputs** added to features to model stochastic local detail (e.g., hair strands, freckles) separately from global style, improving realism.
    - **Style mixing regularization**, combining styles from different w vectors at different depths to encourage disentangled and local controls in latent space.

These models enable **semantic arithmetic in latent space** (e.g., "woman with glasses" ≈ "woman" + "man with glasses" − "man"), and produce faces so realistic that they underpin sites like "this person does not exist."