Autoencoders: From Unsupervised to Generative Deep Learning

Handout Notes - Generative and Deep Learning (GDL)

Davide Bacciu - University of Pisa

Notation. Inputs are denoted by $\mathbf{x} \in \mathbb{R}^D$, reconstructed outputs by $\tilde{\mathbf{x}}$, and latent codes by $\mathbf{z} \in \mathbb{R}^K$. An encoder is denoted by f_θ and a decoder by g_θ, so that $\mathbf{z} = f_\theta(\mathbf{x})$ and $\tilde{\mathbf{x}} = g_\theta(\mathbf{z})$. A dataset is $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$ with $N = |\mathcal{D}|$. A reconstruction loss is written $L(\mathbf{x}, \tilde{\mathbf{x}})$. Regularization terms are denoted by $\Omega(\cdot)$ and weighted by $\lambda > 0$. When probabilistic notation is useful, model parameters are denoted by θ and the log-likelihood by $\ell(\theta) = \log p(\mathcal{D} \mid \theta)$.

1 Why generative deep learning?

A large part of modern machine learning has been driven by discriminative objectives: predict a label, classify a sample, rank alternatives. These objectives are powerful, but they do not directly force a model to understand the structure of the data distribution itself. That limitation matters.

If a model learns only a decision boundary, it may remain brittle, hard to interpret, and overly sensitive to perturbations that do not alter the semantics of the input. By contrast, generative modeling aims to characterize the data distribution:

$$p(\mathbf{x}) \quad \text{or an approximation } p_\theta(\mathbf{x}).$$

This broader objective opens several possibilities:

- understanding factors of variation in the data,
- generating novel samples,
- detecting anomalous or out-of-distribution observations,
- learning task-independent representations that can later be fine-tuned.

This is one reason generative learning is closely related to unsupervised or self-supervised learning. Labelled data are expensive, but raw data are abundant. A model that learns useful structure from unlabeled examples can later transfer that knowledge to downstream tasks.

2 A broad taxonomy of generative models

From a deep-learning perspective, the generative problem can be phrased as follows: given training examples sampled from an unknown data-generating distribution $p(\mathbf{x})$, learn a neural model that can produce new samples from an approximation $p_\theta(\mathbf{x})$.

A useful high-level distinction is between:

- **explicit models**, which define or approximate a density $p_\theta(\mathbf{x})$,
- **implicit models**, which define a stochastic generation process without necessarily providing a tractable density.

Autoencoders sit at an interesting point in this landscape. Basic autoencoders are not explicit density models in the usual probabilistic sense, yet they are central to generative deep learning because they learn latent representations, geometric structure, and reconstruction mechanisms that later lead naturally to probabilistic generative models such as variational autoencoders.

3 What is an autoencoder?

An autoencoder is a model trained to reconstruct its input after passing it through a latent bottleneck. It is composed of two parts:

$$\mathbf{z} = f_\theta(\mathbf{x}), \qquad \tilde{\mathbf{x}} = g_\theta(\mathbf{z}).$$

The encoder f_θ maps the input to a latent representation, and the decoder g_θ maps that latent representation back to input space. The training objective is to make

$$\tilde{\mathbf{x}} \approx \mathbf{x}$$

by minimizing a reconstruction loss

$$\min_\theta \sum_{\mathbf{x}\in\mathcal{D}} L\big(\mathbf{x}, g_\theta(f_\theta(\mathbf{x}))\big).$$

The bottleneck is the essential ingredient. Without some restriction, the model could simply learn the identity map and the task would be trivial. The bottleneck can be created in several ways:

- by forcing $K \ll D$,
- by enforcing sparsity on the latent code,
- by corrupting the input and reconstructing the clean signal,
- by constraining the sensitivity of the encoder,
- by using dropout or other regularization to prevent exact copying.

Thus the autoencoder task is not interesting because copying input to output is difficult. It is interesting because reconstruction must happen through a constrained representation.

Worked example — Basic autoencoder objective

Let

$$\mathbf{z} = f_\theta(\mathbf{x}), \qquad \tilde{\mathbf{x}} = g_\theta(\mathbf{z}).$$

If the input is real-valued, a common reconstruction loss is mean squared error:

$$L(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2.$$

The full training objective is then

$$J_{\mathrm{AE}}(\theta) = \sum_{\mathbf{x}\in\mathcal{D}} \|\mathbf{x} - g_\theta(f_\theta(\mathbf{x}))\|_2^2.$$

If the decoder output models Bernoulli pixel probabilities, one instead often uses cross-entropy:

$$L(\mathbf{x}, \tilde{\mathbf{x}}) = -\sum_{i=1}^{D} \Big[x_i \log \tilde{x}_i + (1 - x_i) \log(1 - \tilde{x}_i)\Big].$$

In either case, the learning signal is entirely self-generated: the target is the input itself.

4 Representation learning and the bottleneck principle

Autoencoders are among the classical tools for representation learning. The latent code $\mathbf{z}$ is useful when it captures the information needed to reconstruct training examples while discarding irrelevant variability.

A low-dimensional bottleneck is the simplest version of this idea. If

$$K \ll D,$$

then the encoder must compress the input. This forces the model to represent only the most informative features. However, dimensionality reduction is not the only way to create a useful bottleneck. Modern autoencoders are often nonlinear and may even use latent dimension $K > D$. What matters is not merely code size, but the presence of constraints that prevent trivial memorization.

5 The manifold hypothesis

A key conceptual justification for autoencoders comes from the manifold hypothesis. Real-world data do not usually fill the ambient input space uniformly. Instead, they tend to lie near a lower-dimensional nonlinear manifold embedded in a high-dimensional space.

For example, handwritten digits may live in a pixel space of dimension hundreds or thousands, but the true degrees of freedom that generate meaningful variation are much fewer: stroke thickness, slant, size, local deformation, and so on. Similarly, natural images, speech, and sensor data exhibit strong dependencies among raw coordinates.

If data lie near a manifold $\mathcal{M}$ of much lower intrinsic dimension, then a good representation should preserve directions along that manifold while being insensitive to perturbations orthogonal to it. Regularized autoencoders approximate this behavior: they learn representations that keep variation useful for reconstruction while suppressing irrelevant directions.

Worked example — Why the manifold hypothesis makes reconstruction meaningful

Suppose that meaningful data lie near a manifold $\mathcal{M} \subset \mathbb{R}^D$ of intrinsic dimension $d \ll D$. A perturbation can be decomposed into:

$$\delta\mathbf{x} = \delta\mathbf{x}_{\parallel} + \delta\mathbf{x}_{\perp},$$

where $\delta\mathbf{x}_{\parallel}$ lies approximately in the tangent space of the manifold and $\delta\mathbf{x}_{\perp}$ points away from it. A good autoencoder should remain sensitive to $\delta\mathbf{x}_{\parallel}$, because those variations correspond to meaningful changes among data samples. But it should be relatively insensitive to $\delta\mathbf{x}_{\perp}$, because those perturbations move the input away from the data manifold. This geometric picture explains why regularized autoencoders are often viewed as manifold learners rather than mere compressors.

6 Sparse autoencoders

A sparse autoencoder introduces a regularization term that penalizes active latent units. The goal is to encourage representations in which only a small subset of code dimensions is used for a given input.

A generic sparse-autoencoder objective is

$$J_{\mathrm{SAE}}(\theta) = \sum_{\mathbf{x}\in\mathcal{D}} \Big(L(\mathbf{x}, \tilde{\mathbf{x}}) + \lambda\, \Omega(\mathbf{z}(\mathbf{x})) \Big), \qquad \mathbf{z}(\mathbf{x}) = f_\theta(\mathbf{x}).$$

A standard choice is an ℓ_1 penalty:

$$\Omega(\mathbf{z}) = \|\mathbf{z}\|_1 = \sum_{j=1}^{K} |z_j|.$$

This biases the encoder toward codes in which many components are close to zero.

Sparse coding is useful because it creates a bottleneck even when the latent dimension is not smaller than the input dimension. The restriction is no longer based on the number of latent coordinates available, but on how many are allowed to be active for any one sample.

6.1 A probabilistic reading

Although standard autoencoders are not full density models, regularized training can still be interpreted in a MAP-like way. If the decoder induces a reconstruction term similar to $-\log p(\mathbf{x} \mid \mathbf{z})$ and the regularizer behaves like $-\log p(\mathbf{z})$, then the sparse-autoencoder objective resembles

$$\max_{\mathbf{z}} \log p(\mathbf{x} \mid \mathbf{z}) + \log p(\mathbf{z}).$$

For an ℓ_1 penalty,

$$\lambda \|\mathbf{z}\|_1$$

corresponds formally to a Laplace-type prior

$$p(\mathbf{z}) \propto \exp(-\lambda \|\mathbf{z}\|_1).$$

The correspondence is not exact in all neural implementations, but it provides useful intuition: sparsity regularization plays the role of a prior that favors simple latent descriptions.

7 Denoising autoencoders

A denoising autoencoder (DAE) is trained not to reconstruct its input directly, but to reconstruct the clean input from a corrupted version. Let

$$\hat{\mathbf{x}} \sim C(\hat{\mathbf{x}} \mid \mathbf{x})$$

be a noisy or partially destroyed version of $\mathbf{x}$ generated by a corruption process C. The encoder receives $\hat{\mathbf{x}}$, while the target remains the original $\mathbf{x}$:

$$\mathbf{z} = f_\theta(\hat{\mathbf{x}}), \qquad \tilde{\mathbf{x}} = g_\theta(\mathbf{z}).$$

The objective becomes

$$J_{\mathrm{DAE}}(\theta) = \sum_{\mathbf{x} \in \mathcal{D}} \mathbb{E}_{\hat{\mathbf{x}} \sim C(\cdot \mid \mathbf{x})} \Big[L(\mathbf{x}, g_\theta(f_\theta(\hat{\mathbf{x}}))) \Big].$$

A common corruption is additive Gaussian noise:

$$\hat{\mathbf{x}} = \mathbf{x} + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 I),$$

or masking noise, in which a subset of components is randomly set to zero.

The intuition is that the representation should be stable under small local damage to the input. Rather than memorizing exact coordinates, the model must learn structure sufficient to undo the corruption.

Worked example — Why denoising discourages the identity map

If an ordinary autoencoder is sufficiently expressive and unconstrained, it may learn

$$g_\theta(f_\theta(\mathbf{x})) \approx \mathbf{x}$$

by implementing a near-identity transformation. In a denoising autoencoder, however, the model sees $\hat{\mathbf{x}}$ but must output $\mathbf{x}$:

$$g_\theta(f_\theta(\hat{\mathbf{x}})) \approx \mathbf{x}.$$

Because $\hat{\mathbf{x}} \neq \mathbf{x}$, a trivial identity map would instead produce

$$g_\theta(f_\theta(\hat{\mathbf{x}})) \approx \hat{\mathbf{x}},$$

which is not the training target. Thus denoising forces the model to learn regularities of the data distribution rather than merely copying inputs.

8 Denoising autoencoders as manifold learning

Denoising autoencoders admit an especially appealing geometric interpretation. If a point is perturbed away from the data manifold by noise, the trained autoencoder learns to map it back toward a nearby plausible point on the manifold. In this sense, the reconstruction function defines a vector field pointing from corrupted samples toward regions of high data density.

This view is important because it begins to connect autoencoders with generative modeling. The denoising map is not yet a normalized probability distribution, but it contains information about the geometry of the data-generating process.

9 Contractive autoencoders

A contractive autoencoder (CAE) makes the encoder explicitly insensitive to small perturbations of the input. Instead of corrupting the input explicitly as in a DAE, it penalizes the Jacobian of the encoder:

$$J_{\mathrm{CAE}}(\theta) = \sum_{\mathbf{x}\in\mathcal{D}} \Big(L(\mathbf{x}, \tilde{\mathbf{x}}) + \lambda\, \Omega(\mathbf{z}) \Big),$$

with

$$\Omega(\mathbf{z}) = \left\| \frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}} \right\|_F^2.$$

Here $\|\cdot\|_F$ denotes the Frobenius norm.

The penalty encourages the latent code to change only mildly when the input changes infinitesimally. So the encoder becomes locally contractive in directions that are not needed for reconstruction.

This gives a direct differential version of the denoising intuition: robust representations should not react strongly to small, irrelevant perturbations.

Worked example — Contractive penalty for a one-layer encoder

Consider a one-hidden-layer encoder

$$\mathbf{z} = f_\theta(\mathbf{x}) = \phi(W\mathbf{x} + \mathbf{b}),$$

where ϕ acts elementwise. Then the Jacobian is

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \operatorname{diag}\big(\phi'(W\mathbf{x} + \mathbf{b})\big)\, W.$$

Hence the contractive penalty becomes

$$\left\| \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right\|_F^2 = \left\| \operatorname{diag}\big(\phi'(W\mathbf{x} + \mathbf{b})\big)\, W \right\|_F^2 .$$

This penalty is small when either:

- the weights are small, or
- the units are in regions where the activation derivative is small.

So the encoder is encouraged to become locally insensitive, except where sensitivity is needed to reconstruct the training data.

10 Dropout and other anti-triviality mechanisms

Another way to prevent trivial identity learning is to place dropout layers inside the autoencoder. Randomly removing input or hidden features during training forces the model to distribute information and learn redundant, useful representations rather than relying on brittle coordinate-wise copying.

This idea is close in spirit to denoising: training takes place under partial destruction of the input or representation, so useful structure must be recovered from incomplete evidence.

11 Deep autoencoders

Autoencoders need not be shallow. A deep autoencoder stacks multiple encoder layers and multiple decoder layers:

$$\mathbf{x} \to \mathbf{z}_1 \to \mathbf{z}_2 \to \cdots \to \mathbf{z}_L \to \tilde{\mathbf{z}}_{L-1} \to \cdots \to \tilde{\mathbf{z}}_1 \to \tilde{\mathbf{x}}.$$

The representation becomes hierarchical: lower layers may capture simple structures, while deeper latent variables encode increasingly abstract factors.

Historically, deep autoencoders were also important as an unsupervised route toward deep learning itself. Before deep networks became easy to optimize end-to-end, unsupervised layerwise pretraining provided a practical way to initialize deep architectures.

12 Layerwise unsupervised pretraining

The classical layerwise strategy works incrementally.

1. Train a first autoencoder on the raw input $\mathbf{x}$ to learn an encoder matrix W_1 and a first latent code $\mathbf{z}_1$.
2. Use $\mathbf{z}_1$ as the training data for a second autoencoder, learning W_2 and a second code $\mathbf{z}_2$.
3. Continue this process to build a stack of latent spaces $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_L$.

4. Optionally, assemble the full encoder and decoder and fine-tune the entire deep autoencoder by backpropagation on input reconstruction.

This procedure is appealing because each stage solves a relatively simple unsupervised learning problem, and each new layer is built on a more abstract representation learned by the previous one.

Worked example — Two-step layerwise pretraining

Suppose the first encoder is

$$\mathbf{z}_1 = f_1(\mathbf{x}) = \phi(W_1\mathbf{x} + \mathbf{b}_1),$$

with decoder approximately reconstructing $\mathbf{x}$ from $\mathbf{z}_1$. After training this first autoencoder, freeze or record its encoder and compute

$$\mathbf{z}_1^{(n)} = f_1(\mathbf{x}^{(n)})$$

for every training example. Then train a second autoencoder on these codes:

$$\mathbf{z}_2 = f_2(\mathbf{z}_1) = \phi(W_2\mathbf{z}_1 + \mathbf{b}_2).$$

Now the deep encoder is approximately

$$\mathbf{z}_2 = f_2(f_1(\mathbf{x})).$$

Repeating this construction yields a hierarchy of increasingly abstract latent spaces.

13 Fine-tuning the whole deep autoencoder

After layerwise pretraining, one can connect the encoder and decoder stacks and optimize the full network jointly:

$$\min_{\theta} \sum_{\mathbf{x}\in\mathcal{D}} L(\mathbf{x}, \tilde{\mathbf{x}}).$$

This fine-tuning stage allows all layers to adapt together to improve reconstruction.

The same pretrained encoder can also be reused for supervised tasks. In that case, the decoder may be discarded and the encoder is combined with a task-specific predictor. This reuse was historically one of the motivations for unsupervised pretraining.

14 Links with Boltzmann machines and deep belief networks

Layerwise pretraining of deep autoencoders is closely related to greedy stacking of Restricted Boltzmann Machines (RBMs). Indeed, if one rearranges the encoder–decoder construction graphically, one sees a close formal relation between:

- stacked autoencoders,
- stacks of pairwise RBMs,
- deep belief networks (DBNs).

A *deep belief network* is not merely a "deep RBM". It is a hybrid directed/undirected model built by stacking RBMs in a particular way. Likewise, a *deep Boltzmann machine* (DBM) is a different undirected generative model whose training is more delicate because of interactions between hidden layers.

For the present lecture, the key message is that deep autoencoders historically provided a bridge between neural representation learning and probabilistic generative models built by layerwise pretraining.

15 A note on pretraining deep Boltzmann machines

Pretraining a DBM requires some care because an intermediate hidden layer is influenced both from below and from above. If one naively stacks pretrained RBMs and combines all contributions at full strength, some effects are effectively counted twice. A common approximation is to halve the contributions of appropriate layers when stitching the pretrained components together. The mathematical details depend on the specific DBM architecture, but the conceptual point is that deep generative stacks are not assembled by simple concatenation alone; one must respect the probabilistic semantics of the intermediate layers.

16 What are autoencoders good for?

Autoencoders are useful not only as historical stepping stones but as practical tools in their own right.

16.1 Visualization and representation learning

A low-dimensional latent space can provide a compact view of complex data. By projecting high-dimensional observations into two or three latent coordinates, one can often reveal clusters, semantic neighborhoods, or progression structures that are hard to see in raw space.

This applies beyond images to audio and other structured signals. What matters is that the latent representation organizes samples according to features useful for reconstruction.

16.2 Image restoration and colorization

When equipped with convolutional encoders and decoders, autoencoders become natural models for restoration tasks. Examples include:

- denoising,
- inpainting,
- super-resolution,
- grayscale-to-color translation.

In all these cases, the model learns to reconstruct a clean or completed target from a degraded or partial input.

16.3 Multimodal representation learning

Deep Boltzmann-like autoencoding ideas can also be extended to multimodal settings, where one wants to couple representations from different data sources such as image and text. In such models, a shared latent structure can support:

- cross-modal retrieval,
- modality completion,
- multimodal generation,
- joint semantic embedding.

16.4 Anomaly detection

A particularly important use of autoencoders is anomaly detection. If an autoencoder is trained mostly on normal data, it should reconstruct those normal samples well. Abnormal data, being off-manifold relative to the training distribution, often reconstruct poorly. One then uses a reconstruction-error score such as

$$s(\mathbf{x}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|^2$$

and declares anomalies when $s(\mathbf{x})$ exceeds a threshold.

Worked example — Reconstruction-based anomaly scoring

Suppose an autoencoder is trained on normal data only. For a new input $\mathbf{x}$, compute its reconstruction $\tilde{\mathbf{x}}$ and define

$$s(\mathbf{x}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2.$$

Estimate the empirical distribution of $s(\mathbf{x})$ on normal validation data and choose a threshold τ. Then classify

$$\mathbf{x} \text{ as anomalous if } s(\mathbf{x}) > \tau.$$

This method works best when anomalies truly lie away from the learned data manifold. It may fail if the autoencoder is so expressive that it reconstructs abnormal inputs well too.

This approach is especially attractive in domains where anomalies are rare and labels are hard to obtain, such as industrial monitoring or medical imaging.

17 Are autoencoders generative models?

At first sight, basic autoencoders seem not to belong to generative modeling at all. There is no explicit normalized density $p_\theta(\mathbf{x})$, no likelihood maximization, and often no principled latent prior. So in the strict probabilistic sense, a plain neural autoencoder is not yet an explicit generative density model.

However, the situation is subtler than that. Denoising and contractive autoencoders, in particular, learn geometric information about the data distribution. In the small-noise limit, the denoising vector field

$$g_\theta(f_\theta(\mathbf{x})) - \mathbf{x}$$

is closely related to the score of the data distribution,

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}).$$

This means that a denoising autoencoder is not merely reconstructing inputs: it is also learning how the data density changes in input space.

That observation is one of the conceptual bridges from deterministic representation-learning autoencoders to explicit probabilistic generative models, culminating in variational autoencoders.