Understanding Autoencoders

From Theory to Applications

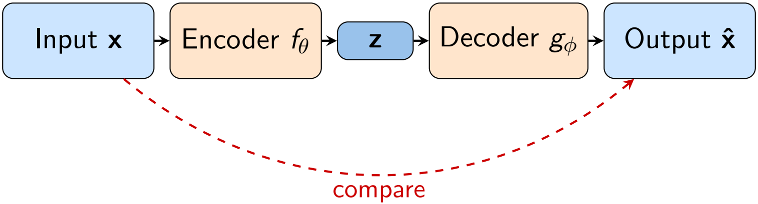
Sayan CHAKI

April 9, 2025

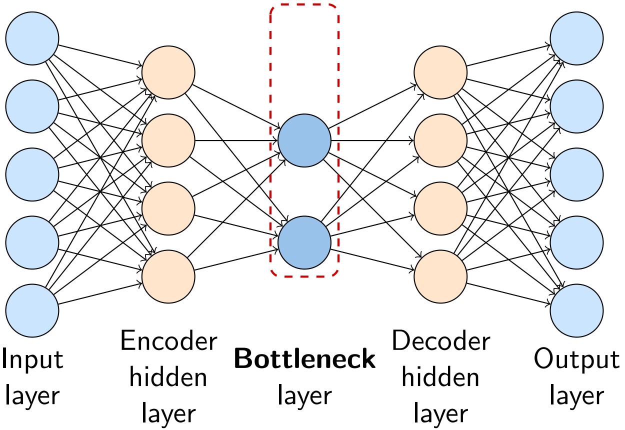
What are Autoencoders? - Definition

Core Definition

- **Autoencoders** are neural networks trained to **copy their input to their output**
- They work by compressing the input into a **latent-space representation**
- Then reconstructing the output from this representation
- The network is forced to learn efficient encodings by limiting the dimensions of the latent space



Autoencoder Architecture - Visual Breakdown



Contd.

Network Components

Input Layer	Original data $\mathbf{x} \in \mathbb{R}^n$
Encoder	Progressively compresses information
Bottleneck	The critical compressed representation $\mathbf{z} \in \mathbb{R}^d$ where $d < n$
Decoder	Progressively reconstructs original data
Output Layer	Reconstructed data $\hat{\mathbf{x}} \in \mathbb{R}^n$

Information Processing

The bottleneck forces the network to learn efficient encoding that preserves essential information while discarding noise

Why Use Autoencoders? - Motivation

Primary Applications

- **Dimensionality reduction**
  - Non-linear alternative to PCA
  - Preserves more complex relationships
- **Feature learning**
  - Unsupervised extraction of important features
  - Discovers hidden patterns in data

Advanced Applications

- **Data denoising**
  - Remove noise from corrupted inputs
  - Restore original data characteristics
- **Generative modeling**
  - Sample from latent space to generate new data
  - Foundation for many generative models

Practical Example - Image Autoencoding

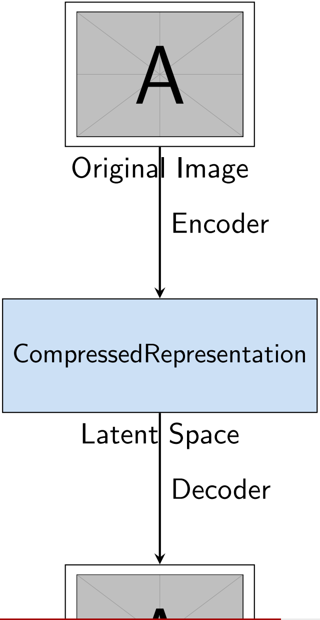


Image Data Processing

- Original image: $28 \times 28 = 784$ pixels
- Compressed to e.g., 32 numbers in latent space
- **Compression ratio**: $784 : 32 = 24.5 : 1$

Benefits

- **Storage efficiency**: 24x smaller representation
- **Feature extraction**: Identify core visual patterns
- **Denoising**: Remove pixel-level noise

Mathematical Formulation - Core Functions

Key Definitions

$$\textbf{Encoder Function}: \quad z = f_\theta(x) \quad \text{where } x \in \mathbb{R}^n, z \in \mathbb{R}^d, d < n \tag{1}$$

$$\textbf{Decoder Function}: \quad \hat{x} = g_\phi(z) \quad \text{where } \hat{x} \in \mathbb{R}^n \tag{2}$$

Encoder Components

- Input vector $\mathbf{x} \in \mathbb{R}^n$
- Weight matrices $\mathbf{W}$ and bias vectors $\mathbf{b}$
- Activation functions σ (e.g., ReLU, sigmoid)
- For single layer: $\mathbf{z} = \sigma(\mathbf{W}x + \mathbf{b})$
- Parameters $\theta = \{\mathbf{W}, \mathbf{b}\}$

Decoder Components

- Latent vector $\mathbf{z} \in \mathbb{R}^d$
- Weight matrices $\mathbf{W}'$ and bias vectors $\mathbf{b}'$
- Activation functions σ' (e.g., ReLU, sigmoid)
- For single layer: $\hat{\mathbf{x}} = \sigma'(\mathbf{W}'z + \mathbf{b}')$
- Parameters $\phi = \{\mathbf{W}', \mathbf{b}'\}$

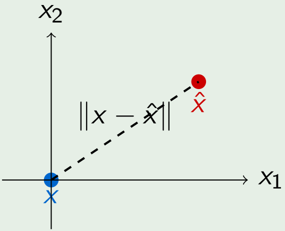
Mathematical Formulation - Loss Functions

Reconstruction Loss

$$\mathcal{L}_{\text{recon}}(x, \hat{x}) = \|x - \hat{x}\|^2 = \|x - g_\phi(f_\theta(x))\|^2 \tag{3}$$

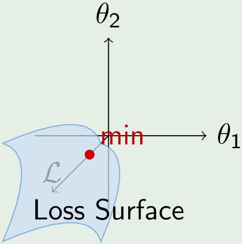
$$= \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \tag{4}$$

MSE Loss Visualization



2D Example

Loss Surface



Mathematical Formulation - Regularization

Extended Loss Functions

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{recon}} + \lambda \cdot \mathcal{L}_{\text{reg}} \tag{6}$$

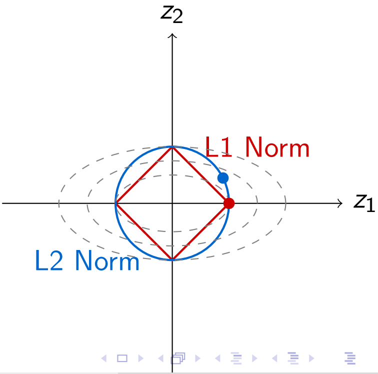
Types of Regularization

- **L1 Regularization (Sparsity)**

$$\mathcal{L}_{\text{L1}} = \sum_{i=1}^{d} |z_i| \tag{7}$$

- **L2 Regularization (Weight Decay)**

$$\mathcal{L}_{\text{L2}} = \sum_{i=1}^{d} z_i^2 \tag{8}$$



Sparse Autoencoders

Definition & Motivation

- **Sparse autoencoders** enforce most hidden units to be inactive
- Inspired by biological neurons that are rarely active
- Creates more distributed and efficient representations
- Prevents the "identity function" shortcut

Mathematical Formulation

$$\mathcal{L}_{\text{sparse}} = \mathcal{L}_{\text{recon}} + \lambda \sum_{j=1}^{d} |z_j| \quad (10)$$

where $\hat{\rho}_j$ is the average activation of unit j

Sparsity Constraints in Autoencoders

Goal: Encourage only a few hidden units to be active for each input.

Key Variables:

- $x \in \mathbb{R}^d$: Input data vector
- $\hat{x} \in \mathbb{R}^d$: Reconstructed output
- $z = f_\theta(x)$: Encoder output (latent representation)
- $\hat{x} = g_\phi(z)$: Decoder output
- $a_j^{(i)}$: Activation of hidden unit j for input $x^{(i)}$
- m: Number of training examples
- $\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} a_j^{(i)}$: Average activation of hidden unit j
- $\rho \in (0, 1)$: Desired sparsity level (e.g., 0.05)
- $\beta \geq 0$: Weight of sparsity penalty
- n_{hidden}: Number of hidden units

Sparsity Constraint

Sparsity Penalty (KL Divergence): $\hat{\rho}_j$: Empirical average activation of hidden neuron j over all inputs: - $\rho \in (0, 1)$:

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} a_j^{(i)}$$

Desired sparsity level (e.g., $\rho = 0.05$ means each neuron should be active only 5- $\mathrm{KL}(\rho \,\|\, \hat{\rho}_j)$: Kullback-Leibler divergence between the desired sparsity and the actual average activation of neuron j

$$\mathrm{KL}(\rho \,\|\, \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}$$

Total Loss:

$$\mathcal{L}_{\text{total}} = \|x - \hat{x}\|^2 + \beta \sum_{j=1}^{n_{\text{hidden}}} \mathrm{KL}(\rho \,\|\, \hat{\rho}_j)$$

Denoising Autoencoders

Definition & Motivation

- **Denoising autoencoders** are trained to reconstruct clean data from corrupted inputs
- Forces learning robust features resistant to noise
- Prevents simple identity mapping
- Excellent for data cleaning applications

Training Process

1. Start with clean input data $\mathbf{x}$
2. Apply corruption process: $\tilde{\mathbf{x}} = q(\tilde{\mathbf{x}}|\mathbf{x})$
3. Train to reconstruct original: $\hat{\mathbf{x}} = g_\phi(f_\theta(\tilde{\mathbf{x}}))$
4. Loss: $\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$