Autoencoders: Unsupervised Learning

Contents: Introduction, Linear Autoencoders, Undercomplete Autoencoders, Overcomplete Autoencoders, Regularization in Autoencoders, Denoising autoencoders, Sparse autoencoders, Contractive autoencoders, deep autoencoders, Stochastic Encoders and Decoders

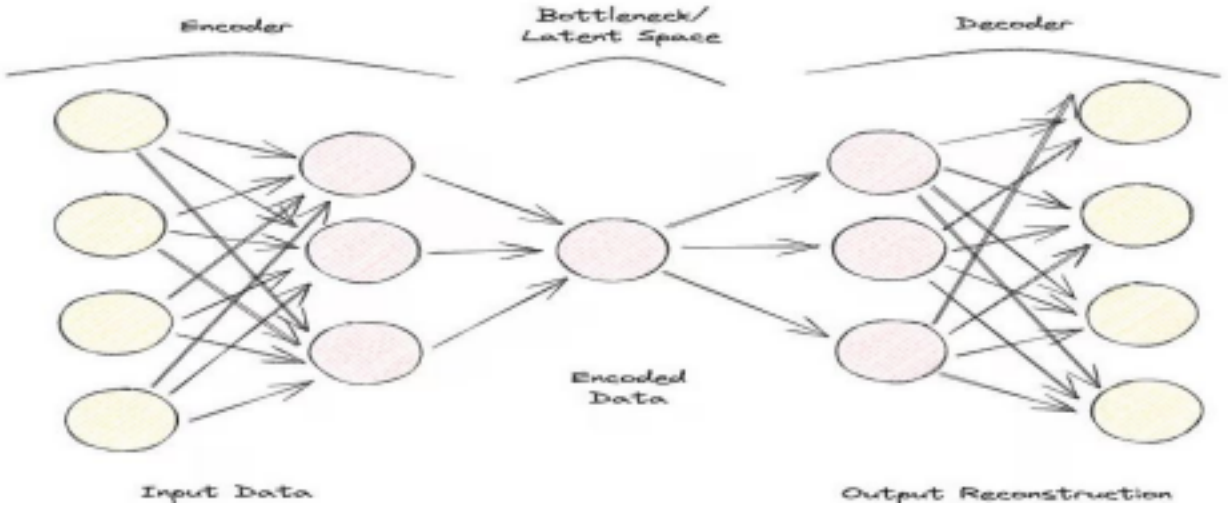
Definition:
An **autoencoder** is a type of neural network used for **unsupervised learning**, primarily for **dimensionality reduction, feature extraction, and data reconstruction**. It learns to encode input data into a compressed representation (encoding) and then reconstruct the original data (decoding).

Autoencoders are a special type of unsupervised feedforward neural network (no labels needed!). The main application of Autoencoders is to accurately capture the key aspects of the provided data to provide a compressed version of the input data, generate realistic synthetic data, or flag anomalies.

Autoencoders are composed of 2 key fully connected feedforward neural networks (Figure 1):

- **Encoder**: compresses the input data to remove any form of noise and generates a latent space/bottleneck. Therefore, the output neural network dimensions are smaller than the input and can be adjusted as a hyperparameter in order to decide how much lossy our compression should be.
- **Decoder**: making use of only the compressed data representation from the latent space, tries to reconstruct with as much fidelity as possible the original input data (the architecture of this neural network is, therefore, generally a mirror image of the encoder). The "goodness" of the prediction can then be measured by calculating the reconstruction error between the input and output data using a loss function.

Repeating iteratively this process of passing data through the encoder and decoder and measuring the error to tune the parameters through backpropagation, the Autoencoder can, with time, correctly work with extremely difficult forms of data. Following shows the Autoencoder Architecture



Some of the most common hyperparameters that can be tuned when optimizing your Autoencoder are:

- The number of layers for the Encoder and Decoder neural networks
- The number of nodes for each of these layers
- The loss function to use for the optimization process (e.g., binary cross-entropy or mean squared error)
- The size of the latent space (the smaller, the higher the compression, acting, therefore as a regularization mechanism)

Finally, Autoencoders can be designed to work with different types of data, such as tabular, time-series, or image data, and can, therefore, be designed to use a variety of layers, such as convolutional layers, for image analysis.

Mathematical Representation:

Given an input x, an autoencoder learns two functions:

- **Encoding function:** $h = f(x)$, where h is the latent representation.
- **Decoding function:** $x' = g(h)$, where x' is the reconstructed output.

The goal is to minimize the reconstruction error:

$$L(x, x') = ||x - x'||^2$$

Applications:

- ✔ Image Denoising
- ✔ Anomaly Detection
- ✔ Data Compression
- ✔ Feature Learning
- ✔ Generative Modeling

Simple Steps on How an Autoencoder Works

1. **Input Data**
   - Start with any data (e.g., an image, text, or numbers).
   - Example: A **handwritten digit image (like 5)** from a dataset.
2. **Encoder (Compressing the Data)**
   - The encoder **shrinks** the input size by finding important patterns. ○ It removes unnecessary details while keeping essential features. ○ Example: Instead of storing **every pixel of the digit "5"**, it only keeps key features like curves and edges.
3. **Latent Space (Compressed Form)**
   - The middle part, where the data is stored in a **small, compact form**. ○ Example: The digit "5" is now stored as a few key numbers instead of a full image.
4. **Decoder (Reconstructing the Data)**
   - The decoder **rebuilds** the data from the compressed form.
   - It tries to **restore the original image or data** as closely as possible. ○

Example: It converts the stored numbers back into a readable digit "5". 5.

Checking Accuracy

- The model compares the original and reconstructed data.
- If they look different, the model **learns and improves** by adjusting itself.

6. **Why Use Autoencoders?**
   - **Reduce storage space** (e.g., for images, videos).
   - **Remove noise** (e.g., improving blurry or corrupted images).
   - **Find hidden patterns** (e.g., for fraud detection or medical images).

Types of Autoencoders

- Undercomplete Autoencoder
- Sparse Autoencoder
- Contractive Autoencoder
- Denoising Autoencoder
- Convolutional Autoencoder
- Variational Autoencoder

2.1 Linear Autoencoders

Linear autoencoders are a simpler form of autoencoders where both the encoder and decoder are linear transformations. In these models, the encoder and decoder functions are linear mappings, typically represented as matrices.

- **Mathematical Representation:**
  - Encoder: $z = W_{enc}x + b_{enc}$
  - Decoder: $x' = W_{dec}z + b_{dec}$

Where W_{enc} and W_{dec} are weight matrices, and b_{enc} and b_{dec} are biases.

- **Main Properties:**
  - Linear autoencoders perform similar to Principal Component Analysis (PCA), which is a linear method for dimensionality reduction.
  - They work well when the underlying data is linearly separable or can be effectively approximated by linear combinations.
  - **Limitation:** They cannot capture non-linear relationships in the data, so they are less effective for more complex data patterns.
- **Use Cases:**
  - **Dimensionality Reduction:** Used when the goal is to project data into a lower-dimensional space.
  - **Feature Extraction:** Extracts the most important features of data when the

data distribution is linear.

2.2 Undercomplete Autoencoders

Undercomplete autoencoders are designed such that the number of neurons in the hidden layer (latent space) is smaller than the number of input neurons. This forces the autoencoder to learn a compressed, efficient representation of the data.

- **Properties:**
  - The encoder compresses the data into a lower-dimensional latent space, where only the most important features are preserved.
  - **Goal:** The autoencoder is forced to learn the most essential, compact features that can still reconstruct the original data accurately.
  - **Training Objective:** Minimize the reconstruction error, often using Mean Squared Error (MSE) between the original input and the reconstructed input.
- **Use Cases:**
  - **Anomaly Detection:** Outliers or anomalies in data will have a higher reconstruction error when passed through the undercomplete autoencoder since they don't match the learned features.
  - **Data Compression:** Reduces the dimensionality of input data while preserving essential information.
- **Example:**
  - If the input is a 1000-dimensional data point, an undercomplete autoencoder might compress it into a 200-dimensional latent space. Only the most important 200 features would be learned and retained.

2.3 Overcomplete Autoencoders

Overcomplete autoencoders, in contrast to undercomplete ones, have more neurons in the latent space than the input layer. This architecture allows the encoder to use more features to represent the data, increasing the capacity of the model.

- **Properties:**
  - **High Capacity:** The overcomplete autoencoder has a high number of latent space dimensions, allowing it to capture more intricate patterns in the data.
  - **Risk of Overfitting:** Since the model has more parameters than needed, it may learn the identity function, meaning it could memorize the input data rather than generalize it, leading to overfitting.

- **Regularization:** Techniques like sparsity constraints (where only a few latent features are active at any time) or denoising can be used to prevent the model from overfitting.

• **Use Cases:**

- **Feature Learning:** Useful when the goal is to learn more detailed features or representations of the data, such as when dealing with complex, non-linear data patterns.
- **Pre-training for Deep Networks:** Overcomplete autoencoders can serve as a pre-training step for deep neural networks by learning useful feature representations before training the entire network.

• **Example:**

- In an image reconstruction task, an overcomplete autoencoder might learn a very rich representation of the image with more neurons in the latent space than there are pixels in the original image.

3. Key Differences between Undercomplete and Overcomplete Autoencoders

Property	Undercomplete Autoencoder	Overcomplete Autoencoder
Latent Space Dimensions	Smaller than the input data dimension (compression)	Larger than the input data dimension (higher capacity)
Learning Objective	Learn compact features for efficient data reconstruction	Learn detailed features with risk of overfitting
Risk of Overfitting	Low risk, as the model must generalize	Higher risk, as the model has high capacity to memorize data
Use Cases	Feature learning, deep network pre training	

	Anomaly detection, dimensionality reduction, denoising	

4. Variants of Autoencoders

1. Denoising Autoencoders (DAEs)

Overview: Denoising Autoencoders are a variant of autoencoders designed to reconstruct data that has been corrupted or noised. The objective is to learn robust features by training the model to recover the original data from noisy or incomplete inputs.

How Denoising Autoencoders Work:

- Corruption Process: During training, the input data is corrupted by adding noise (e.g., Gaussian noise, random pixel dropout for images). This corruption can be in the form of missing values or pixel distortions.
- Objective: The autoencoder is trained to reconstruct the original, clean data from the corrupted version.

Applications:

- Image Denoising: Removing noise from corrupted images.
- Data Preprocessing: Helps to learn invariant features that generalize well across different types of noise.

2. Sparse Autoencoders

Overview: Sparse Autoencoders are autoencoders that enforce sparsity in the hidden layer's activations. Sparsity means that only a small number of neurons in the hidden layer are active at any given time, encouraging the network to learn more efficient and meaningful features.

Applications:

- Feature Learning: Encourages the model to learn efficient features by forcing it to focus on a small set of active neurons.
- Anomaly Detection: By limiting the number of active features, sparse autoencoders can detect outliers or anomalies in the data.
- Compression: The sparse representation reduces the dimensionality of the data effectively.

3. Contractive Autoencoders (CAE)

Overview: Contractive Autoencoders are designed to learn more robust features by enforcing a local contractivity constraint. This means the encoder learns to create representations that are less sensitive to small changes in the input data, making the learned features more stable.

Applications:

- Robust Feature Learning: Helps in learning features that are insensitive to small perturbations in data.
- Data Augmentation: Can be used for tasks where the model needs to generalize well even when input data is noisy or slightly altered.
- Improved Generalization: Helps with generalizing to new, unseen data by discouraging overfitting.

4. Deep Autoencoders

Overview: Deep Autoencoders are multi-layered neural networks, which consist of deep encoder and decoder networks. The primary difference between shallow autoencoders (with one hidden layer) and deep autoencoders is the depth of the network, which allows deep autoencoders to learn more complex features and hierarchies in the data. How Deep Autoencoders Work:

- Multi-Layer Architecture: The encoder and decoder consist of multiple hidden layers. This deep architecture allows the model to learn hierarchical representations of data. For example:
  - Encoder: The encoder learns successive compressed representations of the input by passing it through several layers of nonlinear activations.
  - Decoder: Similarly, the decoder reconstructs the data by progressively expanding the latent representation back to the original data space.

Applications:

- Feature Hierarchies: Deep autoencoders can learn complex hierarchical features, which is useful for image and speech recognition.
- Data Preprocessing: Deep autoencoders can be used for tasks like data compression, feature extraction, or denoising at a deeper level of abstraction.

Challenges:

• Training Deep Networks: Training deep autoencoders is more difficult than shallow ones due to issues like vanishing gradients. Techniques like pretraining or using modern optimizers (Adam, RMSprop) can help alleviate these problems.

5. Stochastic Encoders and Decoders

Overview: Stochastic Encoders and Decoders are a variant where the encoding and decoding process incorporates randomness or probabilistic components. Unlike standard autoencoders, which are deterministic, stochastic autoencoders model distributions over the latent space and the output space.

Applications:

• Variational Autoencoders (VAE): A popular method based on stochastic encoders and decoders. VAEs introduce probabilistic elements to the autoencoder architecture, enabling it to model complex data distributions and perform generative tasks. • Generative Models: Stochastic autoencoders are widely used in generative modeling tasks such as generating new samples from a distribution (e.g., generating new images from a latent space).

• Uncertainty Estimation: Since the model can learn distributions over latent spaces, it can quantify the uncertainty in its predictions, making it useful for tasks where uncertainty is important (e.g., in healthcare or robotics).

Real-world applications where autoencoders have been successfully implemented

1. Image Processing and Computer Vision

• **Denoising Images:** Autoencoders are used to remove noise from images, improving clarity in applications like medical imaging and photography.

• **Super-Resolution:** Enhancing image resolution using techniques like Deep Convolutional Autoencoders (DCAEs).

• **Anomaly Detection in Images:** Identifying defects in manufacturing, medical scans, and surveillance footage.

2. Anomaly Detection

• **Fraud Detection:** Banks and financial institutions use autoencoders to detect fraudulent transactions by identifying patterns that deviate from normal user behavior. • **Industrial Equipment Monitoring:** Used in predictive maintenance by detecting

anomalies in sensor data from machinery.

- **Cybersecurity:** Detecting network intrusions by analyzing network traffic for unusual patterns.

3. Healthcare and Medical Diagnosis

- **Medical Image Reconstruction:** Improving the quality of MRI and CT scans while reducing scan time.
- **Disease Prediction:** Autoencoders help in early disease detection, such as cancer or neurological disorders, by analyzing medical data.
- **Genomic Data Analysis:** Identifying patterns in genetic sequences for disease research.

4. Natural Language Processing (NLP)

- **Text Summarization:** Compressing long documents into concise summaries while preserving key information.
- **Sentiment Analysis:** Identifying emotions and opinions in large-scale text data.
- **Chatbots and Conversational AI:** Used to learn efficient representations of language for response generation.

5. Recommendation Systems

- **Personalized Content Recommendation:** Platforms like Netflix, YouTube, and Spotify use autoencoders to analyze user preferences and recommend movies, videos, and songs.
- **E-commerce Product Recommendations:** Autoencoders help suggest relevant products based on a customer's browsing and purchase history.

6. Self-Driving Cars

- **Sensor Data Compression:** Reducing the size of LiDAR and camera data without losing essential details.
- **Anomaly Detection in Traffic Data:** Identifying unusual traffic patterns and hazardous road conditions.

7. Speech and Audio Processing

- **Speech Enhancement and Noise Reduction:** Used in voice assistants like Alexa and Google Assistant to improve speech recognition in noisy environments. • **Music Generation and Style Transfer:** Creating new compositions by learning latent features from existing music.

8. Robotics and Manufacturing

- **Defect Detection in Products:** Autoencoders analyze product images and flag defective items in automated quality control.
- **Predictive Maintenance in Factories:** Monitoring machine health to prevent failures before they occur.

9. Astronomy and Space Research

- **Galaxy Classification:** Used to classify astronomical objects based on telescope data. • **Anomaly Detection in Space Missions:** Detecting unusual behaviors in satellite and spacecraft telemetry data.

Implementation of autoencoder:

Scenario: Handwritten Digit Compression and Reconstruction

Imagine you have a collection of handwritten digits (0-9) from the **MNIST dataset**, and you want to **reduce their size** while still being able to reconstruct them accurately.

Step 1: What is an Autoencoder?

An **autoencoder** is a type of neural network that learns to **compress** data (encoding) and then **reconstruct** it (decoding) as close to the original as possible.

It consists of:

1. **Encoder**: Compresses input data into a lower-dimensional representation.
2. **Bottleneck (Latent Space)**: The compressed version of the data.
3. **Decoder**: Reconstructs the original input from the compressed representation.

Step 2: How Does It Work?

Let's apply it to handwritten digits.

1. **Input**: A 28x28 grayscale image of a digit (e.g., "5").
2. **Encoder**: Reduces the 28x28 image into a **smaller representation** (e.g., a 10-dimensional vector).
3. **Latent Space**: This compressed representation captures essential features of the digit.
4. **Decoder**: Expands the 10-dimensional vector back to a 28x28 image. 5. **Output**: The reconstructed digit, which should look similar to the original "5."

Even though the autoencoder learns a compressed form, it **still retains enough information** to reconstruct the digit.

Step 3: Why Is It Useful?

• **Dimensionality Reduction** (like PCA but more powerful).

• **Denoising Images** (removing noise while preserving details).

• **Anomaly Detection** (e.g., detecting fake digits by looking at reconstruction errors). • **Feature Extraction** (learning meaningful representations).

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Input, Dense
import numpy as np
import matplotlib.pyplot as plt
# Load the MNIST dataset (handwritten digits 0-9)
(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()
# Normalize pixel values to the range [0, 1] (improves training performance)
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
# Flatten the 28x28 images into 1D vectors of 784 elements
x_train = x_train.reshape((len(x_train), 28 * 28))
x_test = x_test.reshape((len(x_test), 28 * 28))
# Define the size of input and encoding dimension
input_dim = 28 * 28 # 784 pixels (since each image is 28x28) encoding_dim = 32
# We compress the image to a 32-dimensional representation # ----- Building the
Autoencoder Model -----
# Define the Input Layer (784-dimensional input)
input_img = Input(shape=(input_dim,))
# Define the Encoder Layer (reduces 784 to 32)
encoded = Dense(encoding_dim, activation='relu')(input_img)
# Define the Decoder Layer (expands 32 back to 784)
decoded = Dense(input_dim, activation='sigmoid')(encoded)
# Combine Encoder & Decoder to form the Autoencoder
Model autoencoder = keras.Model(input_img, decoded)
# Compile the Autoencoder
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
# ----- Train the Autoencoder -----
```

```
autoencoder.fit(
 x_train, x_train, # Input = Output (Self-learning)
 epochs=10, # Train for 10 epochs
 batch_size=256, # Process 256 images at a time
 shuffle=True, # Shuffle data to improve generalization
 validation_data=(x_test, x_test) # Evaluate on test data
)
# ----- Encode and Decode Test Images -----
# Pass test images through the autoencoder
encoded_imgs = autoencoder.predict(x_test) # Get compressed representations
decoded_imgs = autoencoder.predict(encoded_imgs) # Reconstruct images #
----- Visualizing Original vs Reconstructed Images -----
n = 10 # Display 10 images
plt.figure(figsize=(20, 4))
for i in range(n):
 # Display Original Image
 ax = plt.subplot(2, n, i + 1)
 plt.imshow(x_test[i].reshape(28, 28), cmap="gray")
 plt.axis("off")
 # Display Reconstructed Image
 ax = plt.subplot(2, n, i + 1 + n)
 plt.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray")
 plt.axis("off")

plt.show()
```

Observations from the Visualization (Original vs. Reconstructed Images) After training and running the autoencoder, we can analyze the **original vs. reconstructed**

images to understand how well the model performs.

1️⃣ **Overall Structure is Preserved**

The reconstructed images **maintain the basic shape and outline** of the original handwritten digits.

The numbers are still **clearly recognizable**, meaning the encoder effectively compressed important information.

2️⃣ **Slight Blurring & Loss of Fine Details**

The reconstructed digits are **slightly blurred**, especially at the edges. Some **small strokes or tiny details** are missing, as the encoder only retains key patterns in 32 dimensions.

3️⃣ **Variability in Reconstruction Quality**

✔ Some digits (e.g., **0, 1, 7**) are **reconstructed more clearly** because they have **simpler shapes**.

Digits with **complex strokes** (e.g., **8, 5, 3**) **lose finer details**, making them harder to recognize.

4️⃣ **Compression Effect**

Since we compressed a **784-dimensional image to just 32 dimensions**, the autoencoder **removes redundant details** and keeps only the most significant features. The reconstruction quality depends on how well the **32-dimensional representation** captures the essence of each digit.

https://colab.research.google.com/drive/1gCJqMX6bcoYy7r0Oa_c6kOjIOBmzMofi?usp=sharing