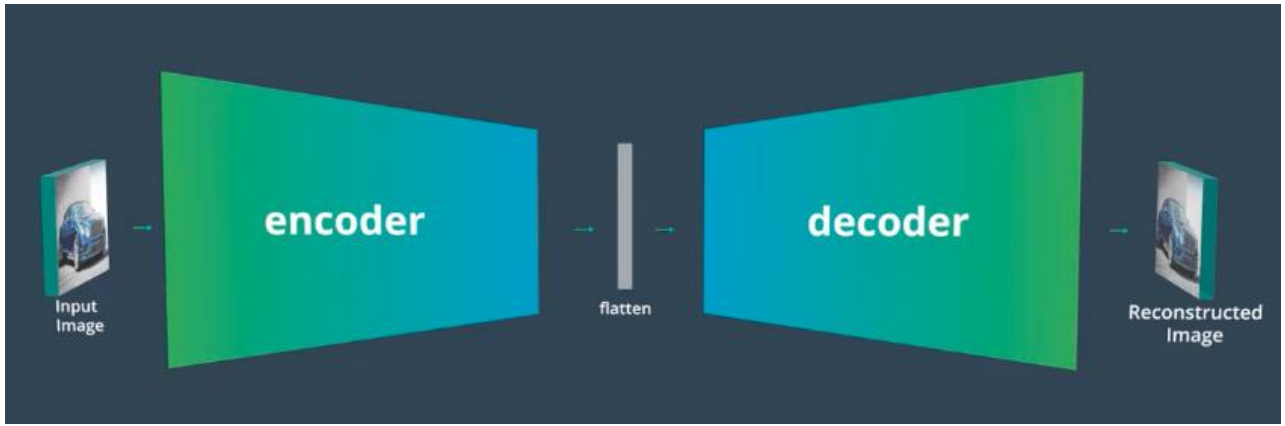
Autoencoders: an overview

Introduction

An autoencoder is a type of neural network architecture designed to efficiently compress (encode) input data down to its essential features, then reconstruct (decode) the original input from this compressed representation.



Introduction

Using unsupervised machine learning, autoencoders are trained to discover *latent variables* of the input data: hidden or random variables that, despite not being directly observable, fundamentally inform the way data is distributed.

Collectively, the latent variables of a given set of input data are referred to as *latent space*.

During training, the autoencoder learns which latent variables can be used to most accurately reconstruct the original data: this latent space representation thus represents only the most essential information contained within the original input.

Autoencoder: Terminology

Information Bottleneck (IB) was introduced in 1999 with a hypothesis that it can extract vital information or representation by compressing the amount of information that can traverse through the network. This information is known as **latent variables** or **latent representations**.

Latent variables are random variables that can’t be observed directly but are extracted from the distribution. These variables are very fundamental and they give us abstract knowledge of the topology and the distribution of the data.

Why autoencoders are used

- The size of the latent space is smaller than the rest of the layers and it forces the information loss and therefore the encoder and decoder needs to work together to find the most efficient compressed form of the input data.

- It helps in extracting most relevant features and can also confirms that “abstract” features we were talking about till now are indeed correct representation of data.

Applications

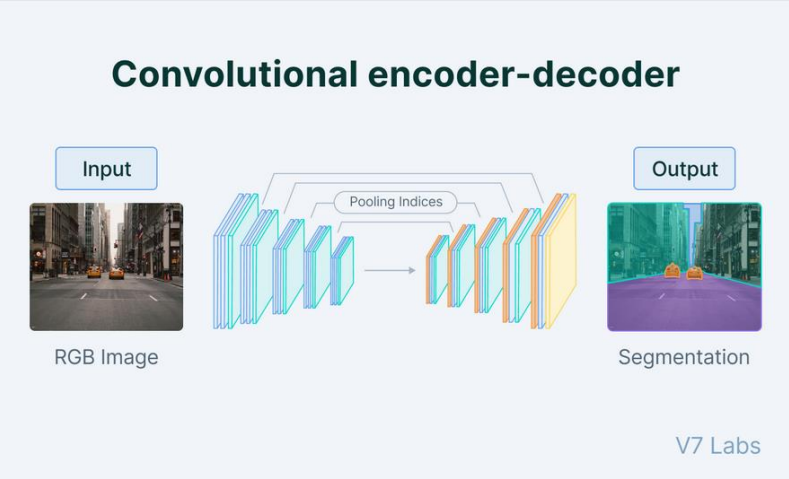
Image Denoising

Autoencoders can be trained where the predicted and actuals are identical and the task is to reproduce the prediction as closely as possible. This verifies the extracted features are correct and if there is some noise then it would be reduced.

Applications

Image Segmentation

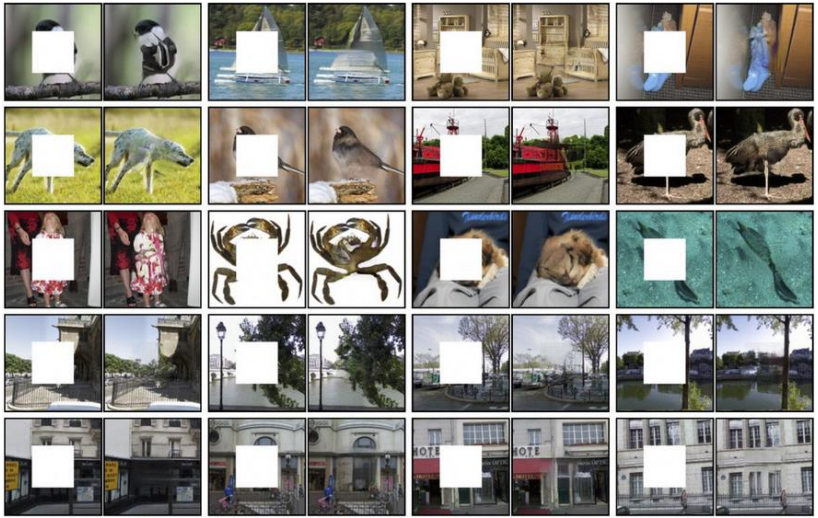
Image segmentation is another example where autoencoders are used to take image inputs preserving spatial representation and aim to output semantic segmented counterparts of the image.



Applications

Neural Inpainting

Some area of image is occluded(hidden) and the autoencoder can be used to generate the representation of occluded space.



Mathematical intuition

The encoded representation (h) can be written as

$$f(x) = h$$

Where f represents the function of encoding or the function our CNN is implementing, x is the input. So, hidden representation is function of input.

The decoder takes the latent variables from the information bottleneck, and then maps them into some output which can be denoted as g(h) = x`. The decoder is usually the mirror opposite of the encoder.

Mathematical intuition

The entire loss (L) can be calculated as deviation between x and decoded representation and can be given as ;

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))),$$

Types: **Undercomplete autoencoder**

Undercomplete autoencoder: Aims to map input x to output x’ by limiting the capacity of model as much as possible, minimizing the amount of information that flows through the network.

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))),$$

Where *L* is the loss function penalizing *g(f(x))* from diverging from the original input *x*. *L* can be a mean squared error or even a mean absolute error.

Types: **Regularized/Sparse autoencoder**

Regularised autoencoders are designed based on data complexity, and they address the problems of Undercomplete autoencoders.

The encoder and decoder, along with the information bottleneck, can have a higher capacity.

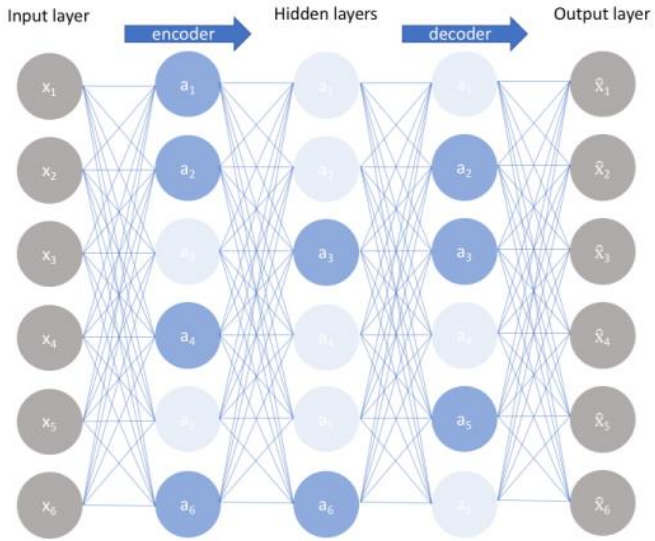
This makes them more flexible and powerful.

One such regularized autoencoder is **sparse autoencoders.**

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}),$$

Types: **Regularized autoencoder**

This approach of penalizing the hidden layers means that the autoencoder can have a larger capacity while still constraining the network to learn representations. The network is constrained by activating only a certain number of neurons in the hidden layer.



Types: **Regularized autoencoder**

There are two ways in which the sparsity can be implemented on a given network:

1. L1 regularization
2. KL-Divergence

In **L1 regularization**, we add a lambda term that penalizes the absolute value of activation *a* in layer *h* to our loss function.

$$L(x,\ g(f(x))\ +\ \lambda h$$

$$L(x,\ g(f(x))\ +\ \lambda \sum_{i}(a_i^h)$$

Types: **Regularized autoencoder**

Kullback-Leibler Divergence or KL-Divergence, on the other hand, calculates the difference between two probability distributions. KL-divergence is excellent at measuring how much data is lost while performing an approximation.

$$L(x, x`) + D_{KL}(P \| Q)$$

Types: **Regularized autoencoder**

Pros:

In sparse autoencoders, **overfitting** is prevented by applying a sparsity penalty. The sparsity penalty is applied on both the hidden layers and reconstruction error. This allows the model to be more versatile by increasing the capacity and learning complex topologies.

Cons:

It is important that the nodes are data-dependent since the input vectors are responsible for the activation of different nodes that yields results during training. Hence, any slight statistical change in the test data can yield different results.

Types: **Contractive autoencoder**

A contractive autoencoder learns representations that are robust to a slight variation of the input data. The idea behind a contractive autoencoder is to map a finite distribution of input to a smaller distribution of output. This idea essentially trains an autoencoder to learn representation even if the neighboring points change slightly.

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') + \lambda \sum_i ||\nabla_x h_i||^2$$

Types: **Contractive autoencoder**

Pros:

Contractive autoencoders are a good choice over sparse autoencoders to learn good representation since they are robust to slight variations and the nodes are not data-dependent.

Cons:

Contractive autoencoders suffer from a major drawback in its reconstruction error during the encoding and decoding process of the input vectors. This leads to neglecting finer details worth considering for reconstruction.

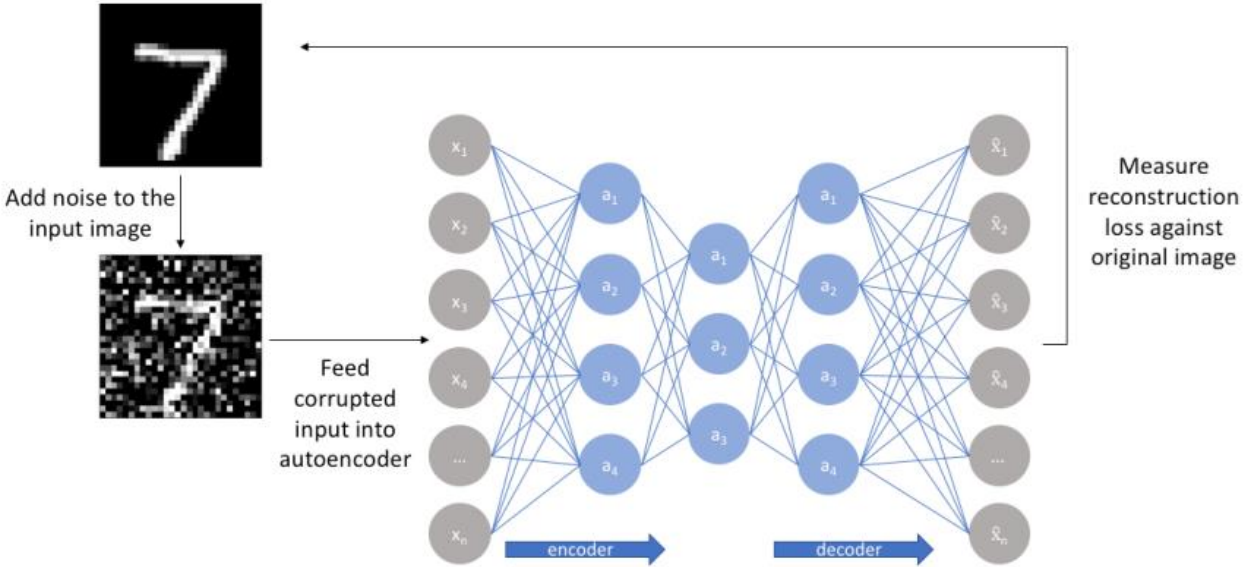
Types: **Denoising autoencoder**

In denoising autoencoders, we pass input that has noise added. The goal here is to train an autoencoder to remove those noises and yield an output that’s noise-free. It’s assumed that the **higher-level representations are relatively stable** and can be easily extracted.

$$L(x, g(f(\grave{x})))$$

Types: **Denoising autoencoder**

The denoising autoencoder is trained to learn the representations and not simply memorize and copy the input to the output, because the input and output aren't the same any



Types: **Denoising autoencoder**

Pros:

Denoising autoencoders is good for **learning the latent representation in corrupted data** while creating a robust representation of the same, allowing the model to recover true features.

Cons:

In order to train a denoising autoencoder, it is important to perform preliminary stochastic mapping to corrupt the data and then use it as input. This does not allow the model to create a mapping because the input and output are different.

Types: **Variational autoencoder**

VAE uses a probabilistic approach to find latent variables or representations. The information bottleneck of VAE consists of two components. One component represents the mean of input distribution while the other represents the standard deviation of the distribution.

The encoder of the VAE (also known as the approximate inference network) tries to infer the properties of latent variables z. This can be described as:

$$q(\boldsymbol{z} \mid \boldsymbol{x})$$

Types: **Variational autoencoder**

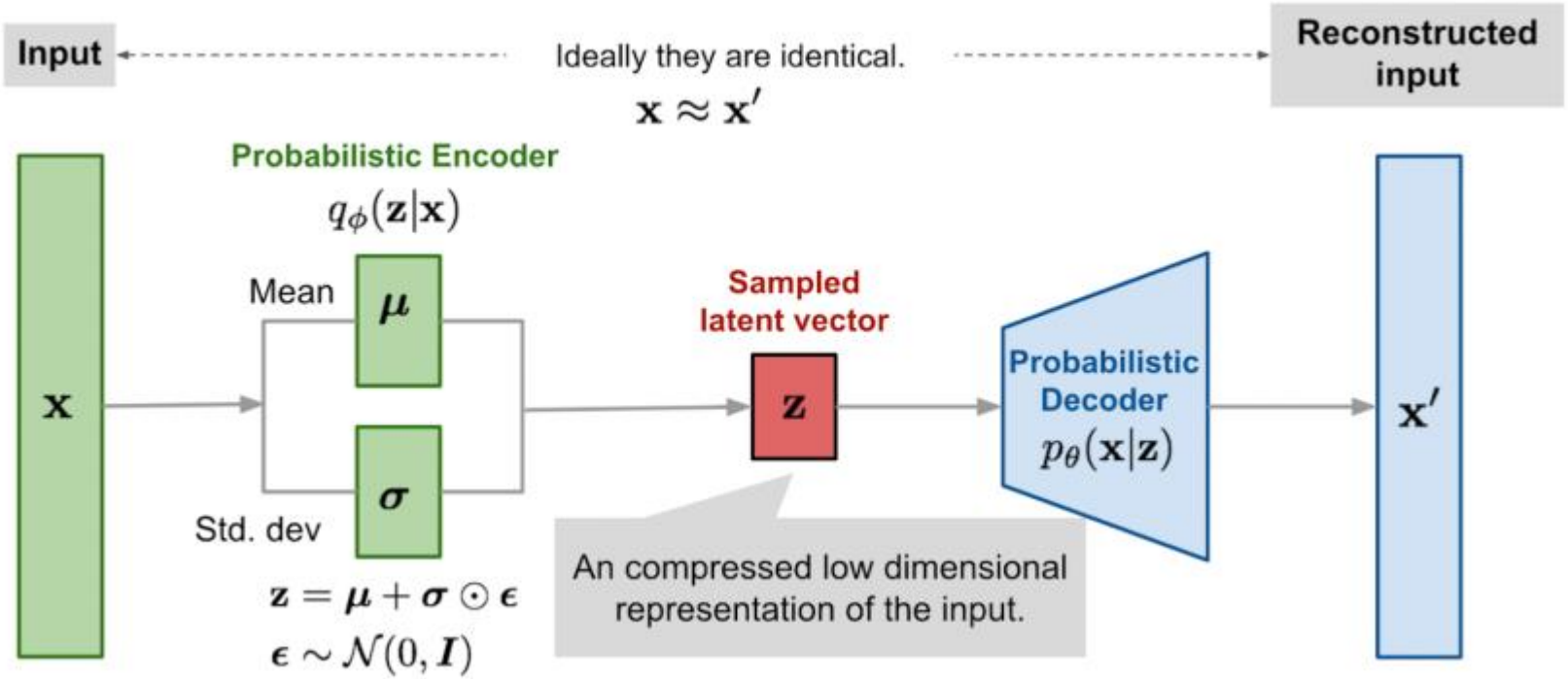
The decoder (known as the generator) takes samples from the latent and generates output. The decoder can be described as:

$$p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{z})$$

VAE can be described as:

$$\mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\boldsymbol{x})} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{z}) - D_{\text{KL}}(q(\mathbf{z} \mid \boldsymbol{x}) || p_{\text{model}}(\mathbf{z}))$$

Types: Variational autoencoder



Types: **Variational autoencoder**

Pros:

VAE gives us control over how we would like to model the distribution of latent variables over other models which can be later used to generate new data.

Cons:

The generated image is blurry because of the injected Gaussian distribution in the latent space.