



## Module 3

# Autoencoders

Autoencoders have emerged as one of the technologies and techniques that enable computer systems to solve data compression problems more efficiently.

They became a popular solution for reducing noisy data.

Simple autoencoders provide outputs that are the same or similar to the input data—only compressed. In the case of variational autoencoders, often discussed in the context of large language models, the output is newly generated content.

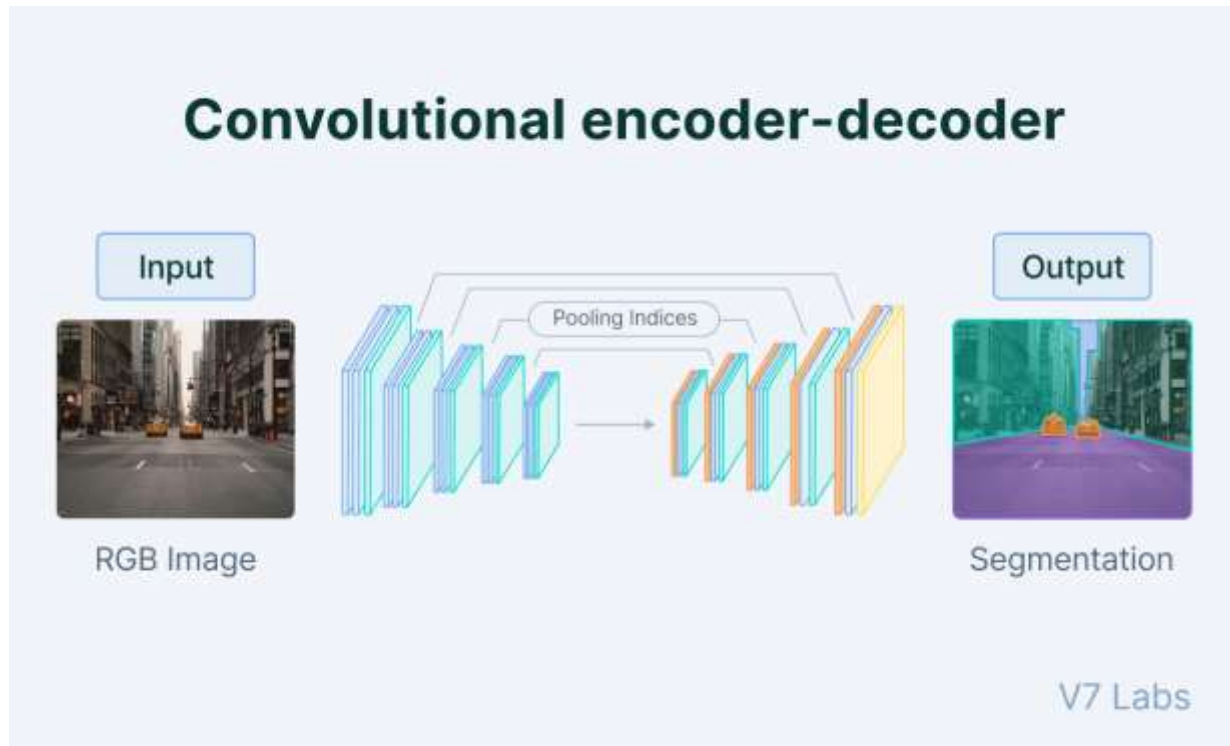
## What is an autoencoder?

An autoencoder is a type of artificial neural network used to learn data encodings in an unsupervised manner.

The aim of an autoencoder is to learn a lower-dimensional representation (encoding) for a higher-dimensional data, typically for dimensionality reduction, by training the network to capture the most important parts of the input image.



## Module 3



# The architecture of autoencoders

Let's start with a quick overview of autoencoders' architecture.

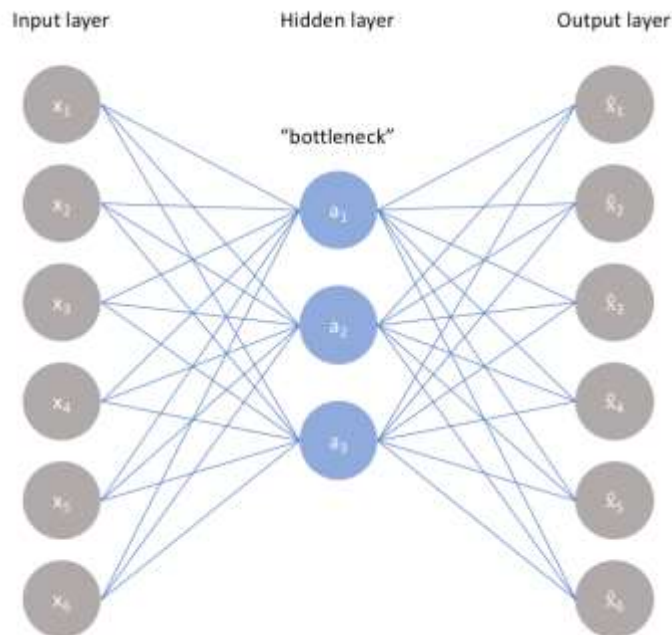
Autoencoders consist of 3 parts:

1. **Encoder:** A module that compresses the train-validate-test set input data into an encoded representation that is typically several orders of magnitude smaller than the input data.
2. **Bottleneck:** A module that contains the compressed knowledge representations and is therefore the most important part of the network.
3. **Decoder:** A module that helps the network "decompress" the knowledge representations and reconstructs the data back from its encoded form. The output is then compared with a ground truth.



## Module 3

The architecture as a whole looks something like this:



The relationship between the Encoder, Bottleneck, and Decoder

### Encoder

The encoder is a set of convolutional blocks followed by pooling modules that compress the input to the model into a compact section called the bottleneck.

The bottleneck is followed by the decoder that consists of a series of upsampling modules to bring the compressed feature back into the form of an image. In case of simple autoencoders, the output is expected to be the same as the input data with reduced noise.

However, for variational autoencoders it is a completely new image, formed with information the model has been provided as input.

### Bottleneck



---

## Module 3

The most important part of the neural network, and ironically the smallest one, is the bottleneck. The bottleneck exists to restrict the flow of information to the decoder from the encoder, thus, allowing only the most vital information to pass through.

Since the bottleneck is designed in such a way that the maximum information possessed by an image is captured in it, we can say that the bottleneck helps us form a *knowledge-representation* of the input.

Thus, the encoder-decoder structure helps us extract the most from an image in the form of data and establish useful correlations between various inputs within the network.

A bottleneck as a compressed representation of the input further prevents the neural network from memorising the input and overfitting on the data.

As a rule of thumb, remember this: The smaller the bottleneck, the lower the risk of overfitting.

However—

Very small bottlenecks would restrict the amount of information storable, which increases the chances of important information slipping out through the pooling layers of the encoder.

### Decoder

Finally, the decoder is a set of upsampling and convolutional blocks that reconstructs the bottleneck's output.

Since the input to the decoder is a compressed knowledge representation, the decoder serves as a “decompressor” and builds back the image from its latent attributes.



# How to train autoencoders?

You need to set 4 hyperparameters before *training* an autoencoder:

1. **Code size:** The code size or the size of the bottleneck is the most important hyperparameter used to tune the autoencoder. The bottleneck size decides how much the data has to be compressed. This can also act as a regularisation term.
2. **Number of layers:** Like all neural networks, an important hyperparameter to tune autoencoders is the depth of the encoder and the decoder. While a higher depth increases model complexity, a lower depth is faster to process.
3. **Number of nodes per layer:** The number of nodes per layer defines the weights we use per layer. Typically, the number of nodes decreases with each subsequent layer in the autoencoder as the input to each of these layers becomes smaller across the layers.
4. **Reconstruction Loss:** The loss function we use to train the autoencoder is highly dependent on the type of input and output we want the autoencoder to adapt to. If we are working with image data, the most popular loss functions for reconstruction are MSE Loss and L1 Loss. In case the inputs and outputs are within the range  $[0,1]$ , as in MNIST, we can also make use of Binary Cross Entropy as the reconstruction loss.

## 5 types of autoencoders

The idea of autoencoders for neural networks isn't new.

In fact—

The first applications date to the 1980s. Initially used for dimensionality reduction and feature learning, an autoencoder concept has evolved over the years and is now widely used for learning generative models of data.



---

## Module 3

Here are five popular autoencoders that we will discuss:

1. Undercomplete autoencoders
2. Sparse autoencoders
3. Contractive autoencoders
4. Denoising autoencoders
5. Variational Autoencoders (for generative modelling)

### 1. Undercomplete autoencoders

An undercomplete autoencoder is one of the simplest types of autoencoders.

The way it works is very straightforward—

Undercomplete autoencoder takes in an image and tries to predict the same image as output, thus reconstructing the image from the compressed bottleneck region.

Undercomplete autoencoders are truly unsupervised as they do not take any form of label, the target being the same as the input.

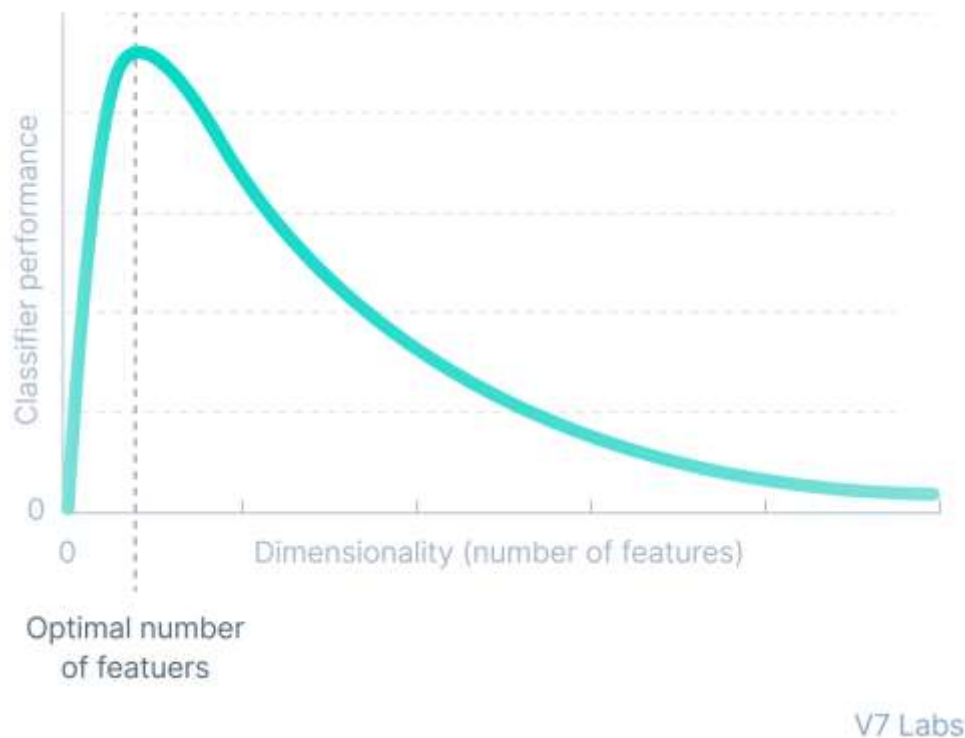
The primary use of autoencoders like such is the generation of the latent space or the bottleneck, which forms a compressed substitute of the input data and can be easily decompressed back with the help of the network when needed.

This form of compression in the data can be modeled as a form of **dimensionality reduction**.





## Module 3



When we think of dimensionality reduction, we tend to think of methods like PCA (Principal Component Analysis) that form a lower-dimensional hyperplane to represent data in a higher-dimensional form without losing information.

However—

PCA can only build linear relationships. As a result, it is put at a disadvantage compared with methods like undercomplete autoencoders that can learn non-linear relationships and, therefore, perform better in dimensionality reduction.

This form of nonlinear dimensionality reduction where the autoencoder learns a non-linear manifold is also termed as *manifold learning*.



## Module 3

Effectively, if we remove all non-linear activations from an undercomplete autoencoder and use only linear layers, we reduce the undercomplete autoencoder into something that works at an equal footing with PCA.

The loss function used to train an undercomplete autoencoder is called *reconstruction loss*, as it is a check of how well the image has been reconstructed from the input data.

Although the reconstruction loss can be anything depending on the input and output, we will use an L1 loss to depict the term (also called the *norm loss*) represented by:

$$L = \|x - \hat{x}\|$$

Where  $\hat{x}$  represents the predicted output and  $x$  represents the ground truth.

As the loss function has no explicit regularisation term, the only method to ensure that the model is not memorising the input data is by regulating the size of the bottleneck and the number of hidden layers within this part of the network—the architecture.

## 2. Sparse autoencoders

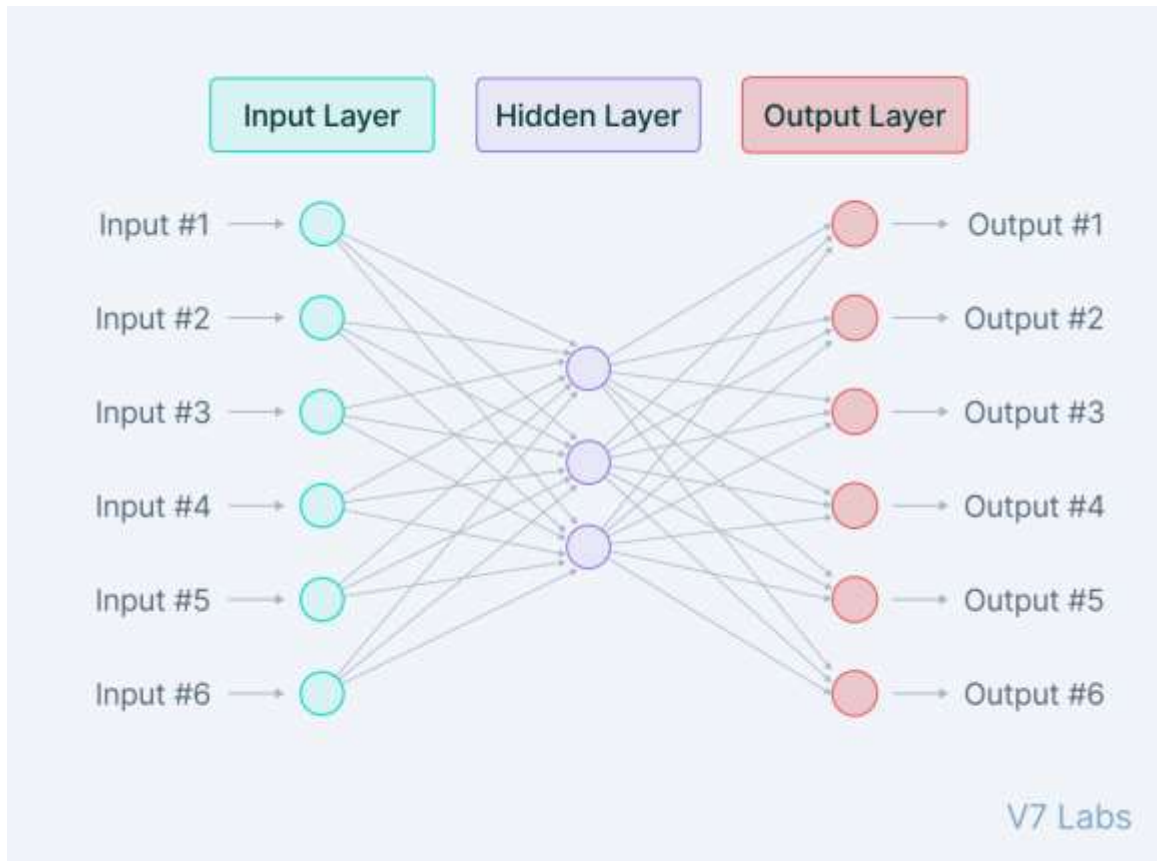
Sparse autoencoders are similar to the undercomplete autoencoders in that they use the same image as input and ground truth. However—

The means via which encoding of information is regulated is significantly different.





## Module 3



While undercomplete autoencoders are regulated and fine-tuned by regulating the size of the bottleneck, the sparse autoencoder is regulated by changing the number of nodes at each hidden layer.

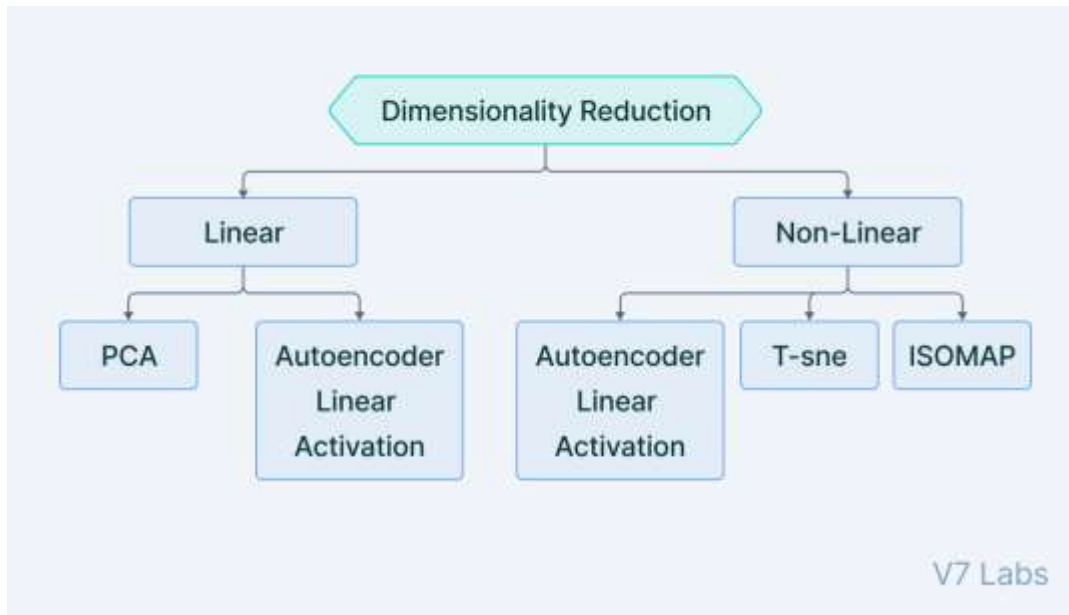
Since it is not possible to design a neural network that has a flexible number of nodes at its hidden layers, sparse autoencoders work by penalizing the activation of some neurons in hidden layers.

In other words, the loss function has a term that calculates the number of neurons that have been activated and provides a penalty that is directly proportional to that.

This penalty, called the *sparsity function*, prevents the neural network from activating more neurons and serves as a regularizer.



## Module 3



While typical regularizers work by creating a penalty on the size of the weights at the nodes, sparsity regularizer works by creating a penalty on the number of nodes activated.

This form of regularization allows the network to have nodes in hidden layers dedicated to find specific features in images during training and treating the regularization problem as a problem separate from the latent space problem.

We can thus set latent space dimensionality at the bottleneck without worrying about regularization.

There are two primary ways in which the sparsity regularizer term can be incorporated into the loss function.

**L1 Loss:** In here, we add the magnitude of the sparsity regularizer as we do for general regularizers:

$$L = x - x^{\wedge} + \lambda \sum |a_i(h)|$$



### Module 3

Where  $h$  represents the hidden layer,  $i$  represents the image in the minibatch, and  $a$  represents the activation.

**KL-Divergence:** In this case, we consider the activations over a collection of samples at once rather than summing them as in the L1 Loss method. We constrain the average activation of each neuron over this collection.

Considering the ideal distribution as a Bernoulli distribution, we include KL divergence within the loss to reduce the difference between the current distribution of the activations and the ideal (Bernoulli) distribution:

$$L = x - x^{\wedge} + \sum_j KL(p || p^{\wedge}_j)$$

Where  $p^{\wedge}_j = \frac{1}{m} \sum_i a_i(h)(x)$  and  $j$  denote the specific neuron for layer  $h$  and a collection of  $m$  samples is being made here, each denoted as  $x$ .

### 3. Contractive autoencoders

Similar to other autoencoders, contractive autoencoders perform task of learning a representation of the image while passing it through a bottleneck and reconstructing it in the decoder.

The contractive autoencoder also has a regularization term to prevent the network from learning the identity function and mapping input into the output.

Contractive autoencoders work on the basis that similar inputs should have similar encodings and a similar latent space representation. It means that the latent space should not vary by a huge amount for minor variations in the input.



## Module 3

To train a model that works along with this constraint, we have to ensure that the derivatives of the hidden layer activations are small with respect to the input data.

Mathematically:

$$\delta h / \delta x$$

Where  $h$  represents the hidden layer and  $x$  represents the input.

An important thing to note in the loss function (formed from the norm of the derivatives and the reconstruction loss) is that the two terms contradict each other.

While the reconstruction loss wants the model to tell differences between two inputs and observe variations in the data, the frobenius norm of the derivatives says that the model should be able to ignore variations in the input data.

Putting these two contradictory conditions into one loss function enables us to train a network where the hidden layers now capture only the most essential information. This information is necessary to separate images and ignore information that is non-discriminatory in nature, and therefore, not important.

The total loss function can be mathematically expressed as:

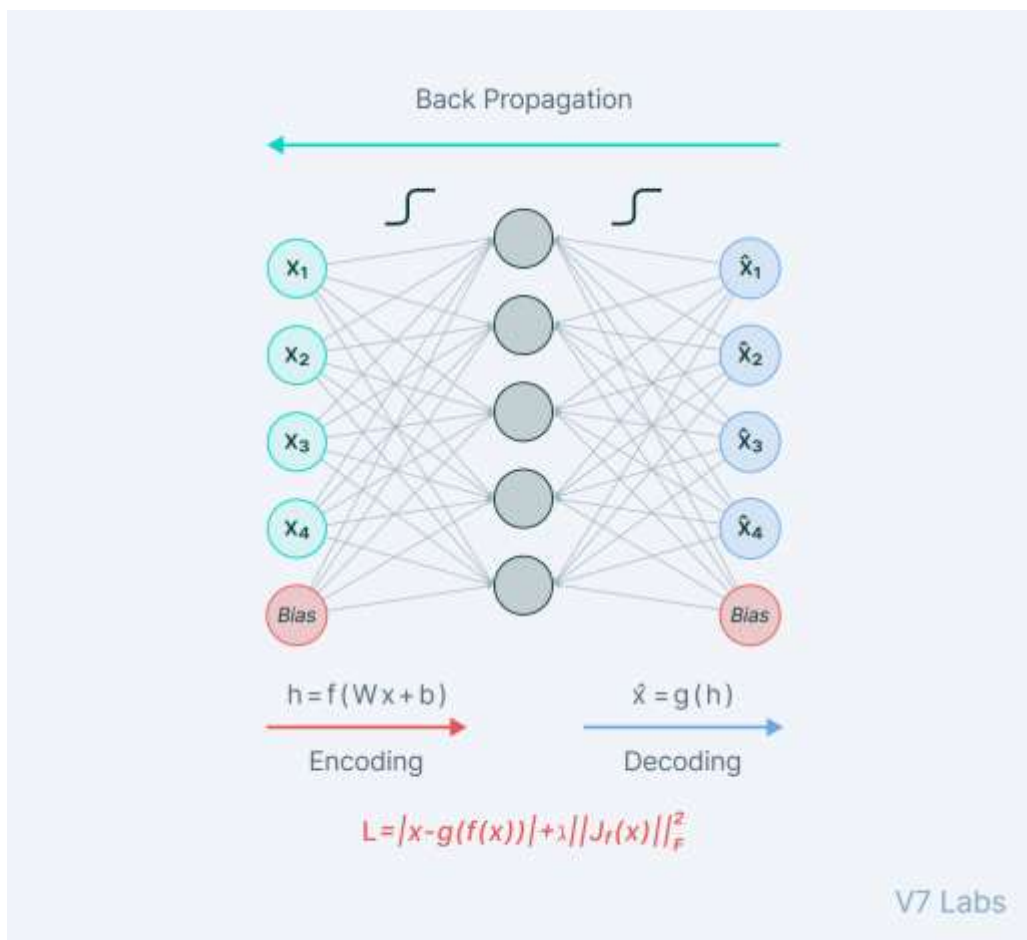
$$L = \|x - \hat{x}\|^2 + \lambda \sum_i \|\nabla x_i(h)(x)\|^2$$

Where  $h_i$  is the hidden layer for which a gradient is calculated and represented with respect to the input  $x$  as  $\nabla x_i(h)(x)$ .



## Module 3

The gradient is summed over all training samples, and a frobenius norm of the same is taken.



## 4. Denoising autoencoders

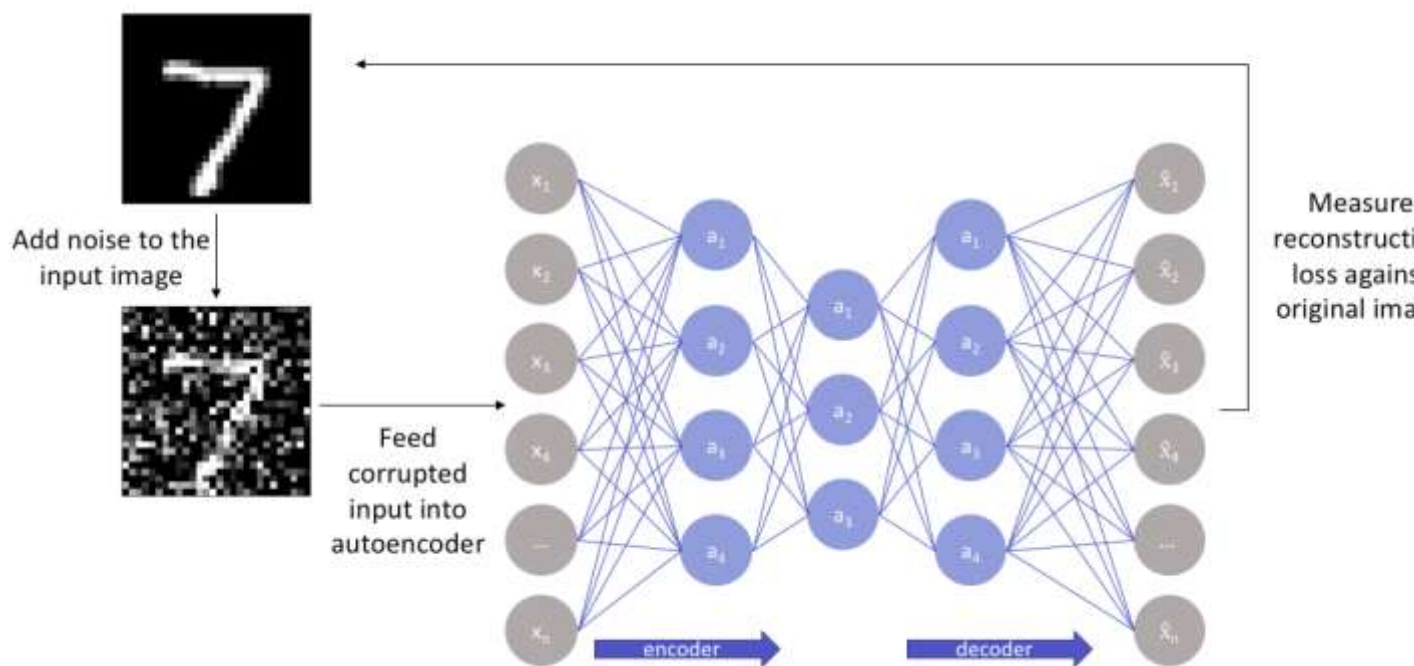
Denoising autoencoders, as the name suggests, are autoencoders that remove noise from an image.

As opposed to autoencoders we've already covered, this is the first of its kind that does not have the input image as its ground truth.



## Module 3

In denoising autoencoders, we feed a noisy version of the image, where noise has been added via digital alterations. The noisy image is fed to the encoder-decoder architecture, and the output is compared with the ground truth image.



The denoising autoencoder gets rid of noise by learning a representation of the input where the noise can be filtered out easily.

While removing noise directly from the image seems difficult, the autoencoder performs this by mapping the input data into a lower-dimensional manifold (like in undercomplete autoencoders), where filtering of noise becomes much easier.

Essentially, denoising autoencoders work with the help of non-linear dimensionality reduction. The loss function generally used in these types of networks is L2 or L1 loss.

## 5. Variational autoencoders





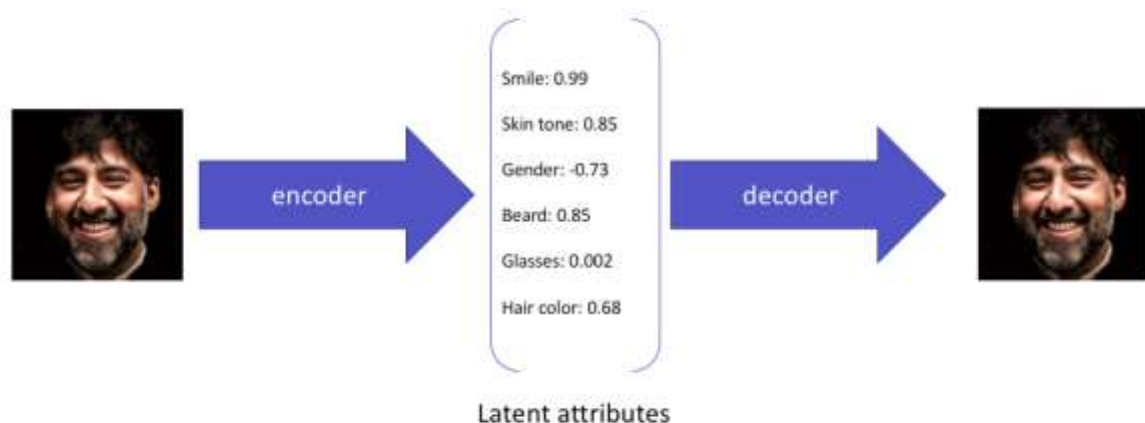
### Module 3

Standard and variational autoencoders learn to represent the input just in a compressed form called the latent space or the bottleneck.

Therefore, the latent space formed after training the model is not necessarily continuous and, in effect, might not be easy to interpolate.

For example—

This is what a variational autoencoder would learn from the input:



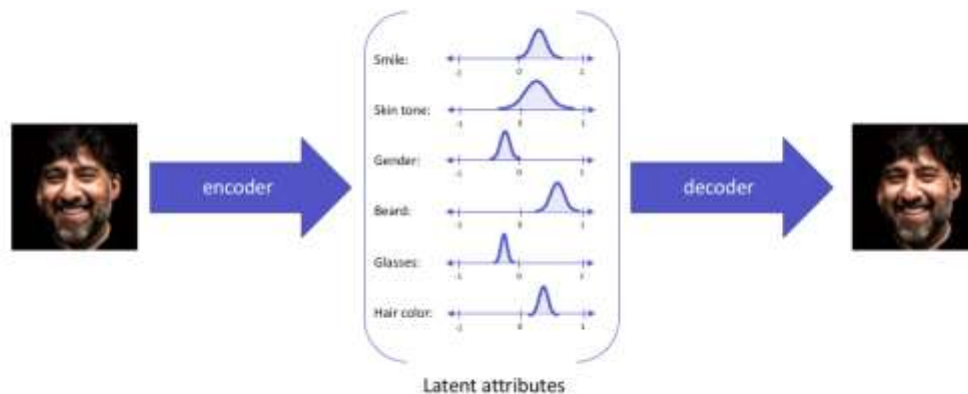
While these attributes explain the image and can be used in reconstructing the image from the compressed latent space, they do not allow the latent attributes to be expressed in a probabilistic fashion.

Variational autoencoders deal with this specific topic and express their latent attributes as a probability distribution, leading to the formation of a continuous latent space that can be easily sampled and interpolated.

When fed the same input, a variational autoencoder would construct latent attributes in the following manner:



## Module 3



The latent attributes are then sampled from the latent distribution formed and fed to the decoder, reconstructing the input.

The motivation behind expressing the latent attributes as a probability distribution can be very easily understood via statistical expressions.

Here's how this works—

We aim at identifying the characteristics of the latent vector  $z$  that reconstructs the output given a particular input. Effectively, we want to study the characteristics of the latent vector given a certain output  $x[p(z/x)]$ .

While estimating the distribution becomes impossible mathematically, a much simpler and easier option is to build a parameterized model that can estimate the distribution for us. It does it by minimizing the KL divergence between the original distribution and our parameterized one.

Expressing the parameterized distribution as  $q$ , we can infer the possible latent attributes used in the image reconstruction.



## Module 3

Assuming the prior  $z$  to be a multivariate Gaussian model, we can build a parameterized distribution as one containing two parameters, the mean and the variance. The corresponding distribution is then sampled and fed to the decoder, which then proceeds to reconstruct the input from the sample points.

But—

While this seems easy in theory, it becomes impossible to implement because backpropagation cannot be defined for a random sampling process performed before feeding the data to the decoder.

To get by this hurdle, we use the reparameterization trick—a cleverly defined way to bypass the sampling process from the neural network.

What is it all about?

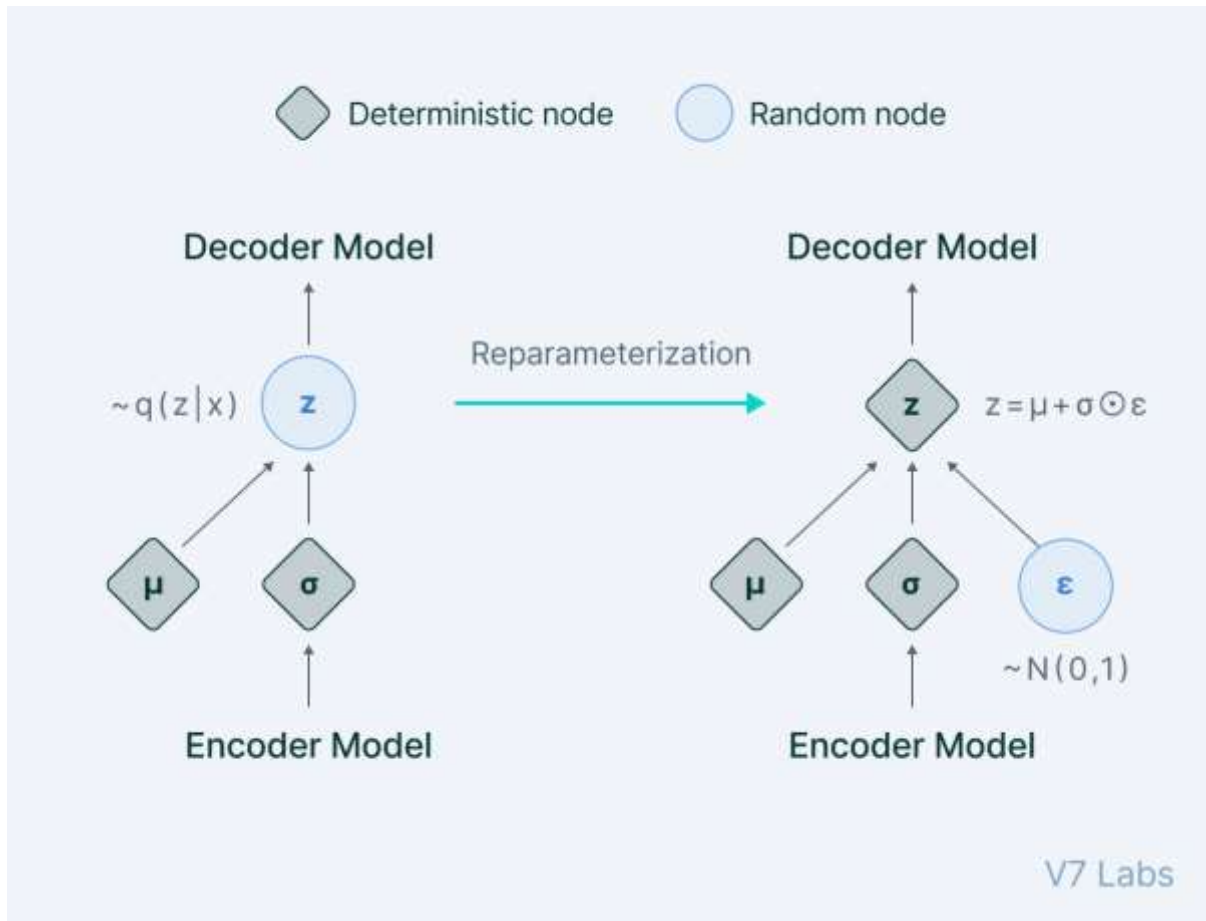
In the reparameterization trick, we randomly sample a value  $\epsilon$  from a unit Gaussian and then scale this by the latent distribution variance  $\sigma$  and shift it by the mean  $\mu$  of the same.

Now, we have left behind the sampling process as something done outside what the backpropagation pipeline handles, and the sampled value  $\epsilon$  acts just like another input to the model, that is fed at the bottleneck.

A diagrammatic view of what we attain can be expressed as:



## Module 3



The variational autoencoder thus allows us to learn smooth latent state representations of the input data.

To train a VAE, we use two loss functions: the reconstruction loss and the other being the KL divergence.

While reconstruction loss enables the distribution to correctly describe the input, by focusing only on minimizing the reconstruction loss, the network learns very *narrow* distributions—akin to discrete latent attributes.

The KL divergence loss prevents the network from learning narrow distributions and tries to bring the distribution closer to a unit normal distribution.

The summarised loss function can be expressed as:



## Module 3

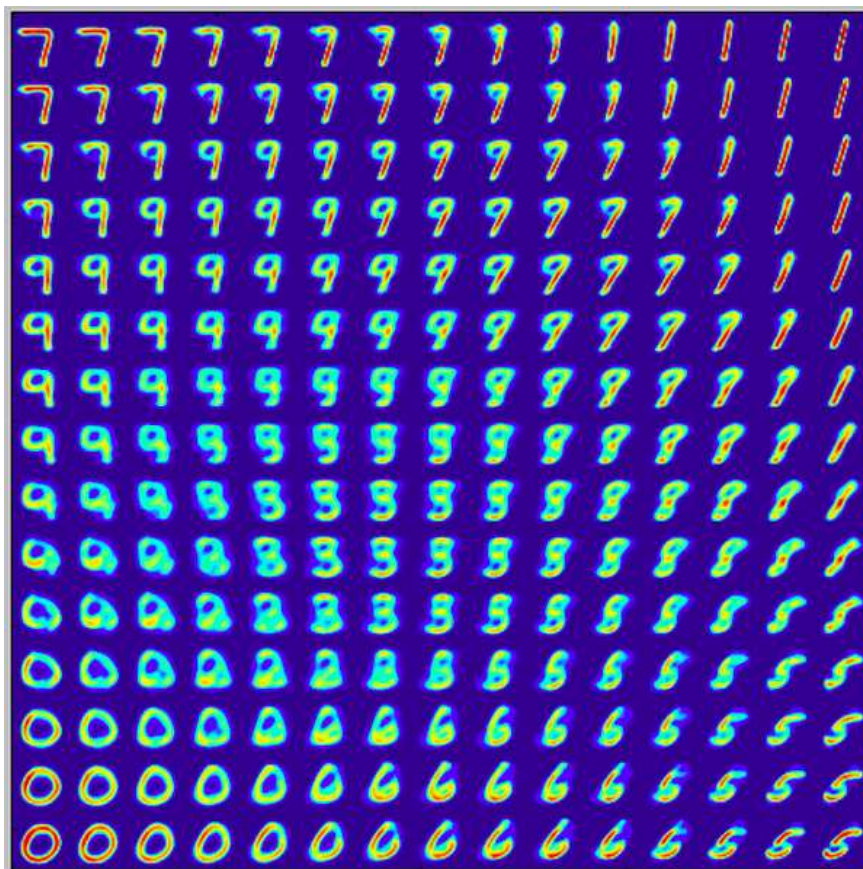
$$L = x - x^{\wedge} + \beta \sum_i \text{KL}(q_j(z|x) \| N(0,1))$$

Where  $N$  denotes the normal unit distribution and  $\beta$  denotes a weighting factor.

The primary use of variational autoencoders can be seen in generative modeling.

Sampling from the latent distribution trained and feeding the result to the decoder can lead to data being generated in the autoencoder.

A sample of MNIST digits generated by training a variational autoencoder is shown below:







## Module 3

# Applications of autoencoders

Now that you understand various types of autoencoders, let's summarize some of their most common use cases.

## 1. Dimensionality reduction

Undercomplete autoencoders are those that are used for dimensionality reduction.

These can be used as a pre-processing step for dimensionality reduction as they can perform fast and accurate dimensionality reductions without losing much information.

Furthermore, while dimensionality reduction procedures like PCA can only perform linear dimensionality reductions, undercomplete autoencoders can perform large-scale non-linear dimensionality reductions.

## 2. Image denoising

Autoencoders like the denoising autoencoder can be used for performing efficient and highly accurate image denoising.

Unlike traditional methods of denoising, autoencoders do not search for noise, they extract the image from the noisy data that has been fed to them via learning a representation of it. The representation is then decompressed to form a noise-free image.

Denoising autoencoders thus can denoise complex images that cannot be denoised via traditional methods.

## 3. Generation of image and time series data

Variational Autoencoders can be used to generate both image and time series data.





---

## Module 3

The parameterized distribution at the bottleneck of the autoencoder can be randomly sampled to generate discrete values for latent attributes, which can then be forwarded to the decoder, leading to generation of image data. VAEs can also be used to model time series data like music.

### 4. Anomaly detection

Undercomplete autoencoders can also be used for anomaly detection.

For example—consider an autoencoder that has been trained on a specific dataset  $P$ . For any image sampled for the training dataset, the autoencoder is bound to give a low reconstruction loss and is supposed to reconstruct the image as is.

For any image which is not present in the training dataset, however, the autoencoder cannot perform the reconstruction, as the latent attributes are not adapted for the specific image that has never been seen by the network.

As a result, the outlier image gives off a very high reconstruction loss and can easily be identified as an anomaly with the help of a proper threshold.