**Autoencoders** are type of artificial neural network that aims to copy their inputs to their outputs. They work by compressing the input into a **latent-space** **representation**also known as **bottleneck**, and then reconstructing the output from this representation. Autoencoder is an unsupervised machine learning algorithm. We can define autoencoder as **feature extraction algorithm** through **dimensionality reduction**.

***1. Feature extraction***

**Feature projection** (also called **feature extraction**) transforms the data from the high-dimensional space to a space of fewer dimensions. The data transformation may be linear, as in principal component analysis (**PCA**), but many **nonlinear dimensionality reduction** techniques also exist (this is the one that **autoencoders** use). For multidimensional data, **tensor representation** can be used in dimensionality reduction through **multilinear subspace learning**.

***2. PCA***

The main **linear** technique for dimensionality reduction, **principal component analysis**, performs a linear mapping of the data to a lower-dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized.

***3. ICA***

***4. Dimensionality reduction***

**Dimensionality reduction** is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension (The **intrinsic dimension** for a data set can be thought of as the number of variables needed in a minimal representation of the data. Similarly, in signal processing of multidimensional signals, the **intrinsic dimension** of the signal describes how many variables are needed to generate a good approximation of the signal).

**Encoder**: This part of the network encodes or compresses the input data into a latent-space representation. The compressed data typically looks garbled, nothing like the original data.

**Decoder:** This part of network decodes or reconstructs the encoded data (latent space representation) back to original dimension. The decoded data is a lossy reconstruction of the original data.

Purpose of autoencoders in not to copy inputs to outputs, but to train autoencoders to copy inputs to outputs in such a way that **bottleneck** will learn useful information or properties.

The **bottleneck**(or **“code”**) contains the most compressed representation of the input: it is both the output layer of the encoder network and the input layer of the *decoder* network. A fundamental goal of the design and training of an autoencoder is discovering the minimum number of important features (or *dimensions*) needed for effective reconstruction of the input data. The latent space representation–that is, the *code*–emerging from this layer is then fed into the decoder.

*We need an* ***undercomplete*** *autoencoder:*

Undercomplete autoencoders are a simple autoencoder structure used primarily for **dimensionality reduction**. Their hidden layers contain fewer nodes than their input and output layers, and the capacity of its bottleneck is fixed.

The goal of this bottleneck is to prevent the autoencoder from overfitting to its training data. Without sufficiently limiting the capacity of the bottleneck, the network tends toward learning the **identity function** between the input and output: in other words, it may learn to minimize reconstruction loss by simply copying the input directly. By forcing the data to be significantly compressed, the neural network must learn to retain only the features most essential to reconstruction.

***Autoencoder hyperparameters***

In addition to selecting the appropriate type of neural network — for example, a **CNN-based** architecture, an **RNN-based** architecture (like **long short-term memory**), a **transformer architecture**, or a simple **vanilla feed-forward neural network** — the design of an autoencoder entails multiple hyperparameters:

* **Code size**: The size of the bottleneck determines how much the data is to be compressed. The code size can also be used a regularization term: adjustments to code size are one way to counter overfitting or underfitting.
* **Number of layers**: The depth of the autoencoder is measured by the number of layers in the encoder and decoder. More depth provides greater complexity, while less depth provides greater processing speed.
* **Number of nodes per layer**: Generally, the number of nodes (or “neurons”) decreases with each encoder layer, reaches a minimum at the bottleneck, and increases with each layer of the decoder layer—though in certain variants, like sparse autoencoders, this is not always the case. The number of neurons may also vary per the nature of input data: for example, an autoencoder dealing with large images would require more neurons than one dealing with smaller images.
* **Loss function**: When training an autoencoder, the loss function—which measures reconstruction loss between the output and input—is used to optimize model weights through gradient descent during backpropagation. The ideal algorithm(s) for the loss function depends on the task the autoencoder will be used for.

***Long short-term memory (LSTM)***

Long short-term memory (LSTM) is a type of recurrent neural network (**RNN**) aimed at dealing with the vanishing gradient problem present in traditional RNNs. LSTM excels in sequence prediction tasks, capturing long-term dependencies. It is ideal for time series.

A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTM models address this problem by introducing a **memory cell**, which is a container that can hold information for an extended period.

The LSTM architecture involves the memory cell which is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell.

- The **input** gate controls what information is **added** to the memory cell.

- The **forget** gate controls what information is **removed** from the memory cell.

- The **output** gate controls what information is **output** from the memory cell.

***Denoising Autoencoders (DAE)***

The objective of DAE is to minimize the difference between the original input (clean input without the noise) and the reconstructed output. This is quantified using a **reconstruction loss function**.

It quantifies the discrepancy between the original input and its reconstructed version, often using measures like mean squared error (MSE) for continuous data or mean absolute error (MAE). The aim is to minimize this discrepancy, leading to a model that can capture and replicate the essential features of the input data.

During model training, the reconstruction error of the denoised output is not measured against the corrupted input data, but against the original image.

**Error Metrics**

**Many error metrics mentioned here:** [**https://arxiv.org/pdf/2006.00887**](https://arxiv.org/pdf/2006.00887)

**1. RMSE**

**2. MSE**

**3. MAE**

**4. PSNR (Peak Signal-to-Noise Ration):** (used here: <https://par.nsf.gov/servlets/purl/10195190>): a widely used metric for assessing the quality of reconstruction in signal processing, especially for images and other types of signals. It provides a measure of how much the reconstructed signal deviates from the original signal by **comparing the maximum possible signal strength to the amount of noise or error introduced during the reconstruction process**. Essentially, it quantifies how much noise is present relative to the strength of the signal. The **higher** the value of PSNR, the **better** will be the quality of the output image.

Formula:

A close up of a number

Description automatically generated

where **MAXf**is the maximum signal value that exists in our original “known to be good” image and **MSE** is the Mean Squared Error**.**

**Sensitivity to Noise**: PSNR is sensitive to small differences between the original and reconstructed signals, especially in the higher values of the signal. This makes it particularly useful in applications where preserving high-frequency details (like sharp edges in images or precise spectral features) is important.

**Activation Functions**

**Sigmoid (logistic) function:** The sigmoid activation function outputs values between 0 and 1. It introduces non-linearity and is particularly useful for making sure the output remains within a specific range.

**Tanh function:** Outputs values between -1 and 1.