**Auto Encoder and Decoder**

An autoencoder is a type of artificial neural network used for unsupervised learning. It is designed to learn efficient representations or encodings of data, typically by reducing the dimensionality of the input and capturing the most important features. The architecture consists of an encoder and a decoder.

1. Encoder:

- The encoder takes in the input data and transforms it into a lower-dimensional representation. This process involves mapping the input data to a set of latent variables or features.

- The encoder's goal is to capture the essential features of the input in a compressed form, forcing the network to learn a more efficient representation.

2. Decoder:

- The decoder takes the compressed representation generated by the encoder and reconstructs the original input data.

- It aims to produce an output that closely resembles the input, based on the information encoded by the encoder.

- The decoder helps the autoencoder learn to preserve the important information during the encoding process.

The training of an autoencoder involves minimizing the difference between the input data and the reconstructed output. This process encourages the network to learn a compressed representation that retains meaningful information about the input. Autoencoders have applications in data denoising, dimensionality reduction, and feature learning.

It's worth noting that autoencoders are not limited to a specific type of neural network architecture. Common variants include stacked autoencoders, convolutional autoencoders (for image data), and recurrent autoencoders (for sequential data).

Diagram:

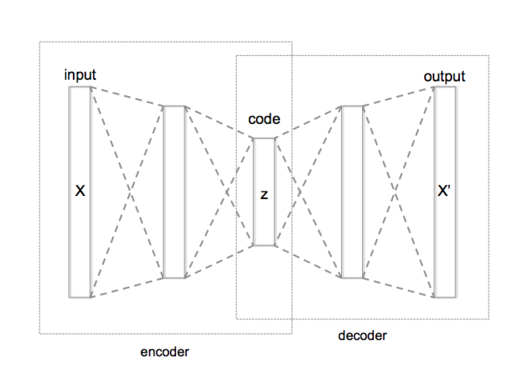


Fig 1: Schematic structure of an autoencoder with 3 fully connected hidden layers. The code (z, or h for reference in the text) is the most internal layer.

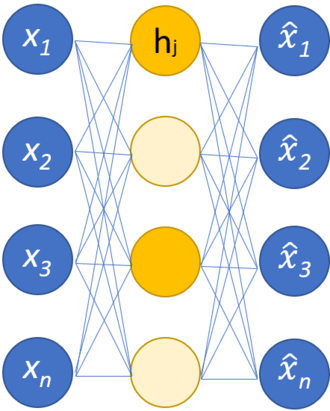


Fig 2: Simple schema of a single-layer sparse autoencoder. The hidden nodes in bright yellow are activated, while the light yellow ones are inactive. The activation depends on the input.

The steps involved in the processing of an autoencoder:

1. Input Data:

- Provide the raw input data that the autoencoder will learn to encode and decode. This data could be images, text, numerical values, or any other type of information.

2. Encoder:

- The input data is fed into the encoder, which consists of one or more layers of neurons. The encoder transforms the input into a compressed representation, reducing its dimensionality.

3. Latent Representation:

- The output of the encoder is the compressed or latent representation of the input data. This representation captures the essential features of the input in a lower-dimensional space.

4. Decoder:

- The compressed representation is then fed into the decoder, which is another set of layers in the neural network. The decoder's role is to reconstruct the original input data from the compressed representation.

5. Reconstructed Output:

- The output of the decoder is the reconstructed data, which ideally should closely resemble the original input. This reconstructed output is compared to the actual input during training.

6. Loss Calculation:

- A loss function is used to measure the difference between the input data and the reconstructed output. Common loss functions include mean squared error or binary cross-entropy, depending on the nature of the data.

7. Backpropagation:

- The gradient of the loss with respect to the model parameters is calculated using backpropagation. This gradient is then used to update the weights of the neural network, minimizing the difference between the input and the reconstructed output.

8. Training Iterations:

- Steps 2 to 7 are repeated for multiple iterations or epochs until the autoencoder learns to encode and decode the input data effectively. The goal is to converge to a set of weights that produces a minimal reconstruction error.

9. Testing and Inference:

- Once trained, the autoencoder can be used for testing and inference. New data can be encoded and decoded using the learned representations, allowing the model to generate meaningful reconstructions or compress the input data.

This process is a basic overview, and the specific architecture and parameters of the autoencoder can vary depending on the application and the nature of the data being processed.

\*\*\*\*\*\*\*\*\*\*\*

Activation functions play a crucial role in introducing non-linearity to neural networks, including autoencoders. Here's a list of activation functions commonly used in autoencoders for key feature extraction, and output generation:

1. ReLU (Rectified Linear Unit):

- Description:

- Purpose: Introduces non-linearity by outputting the input for positive values and zero for negative values. Efficient and helps with the vanishing gradient problem.

2. Sigmoid:

- Description:

- Purpose: Squashes input values between 0 and 1, making it suitable for binary classification tasks. Commonly used in the output layer for binary data.

3. Tanh:

- Description:

- Purpose: Similar to the sigmoid but outputs values between -1 and 1. It helps mitigate issues related to the zero-centered nature of the data.

4. Leaky ReLU:

- Description: where is a small positive constant

- Purpose: Addresses the "dying ReLU" problem by allowing a small, non-zero gradient for negative input values.

5. Softmax (Output Layer):

- Description: Converts a vector of raw scores into probabilities that sum to 1.

- Purpose: Often used in the output layer for multi-class classification tasks, ensuring the network's output represents a valid probability distribution.

6. Swish:

- Description: where is the sigmoid function.

- Purpose:Designed to capture non-linearities while maintaining smooth gradients. It has shown promising performance in some scenarios.

7. GELU (Gaussian Error Linear Unit):

- Description:

- Purpose: A smooth activation function with a non-monotonic behavior, GELU has been used in transformer models and has shown good performance in certain contexts.

8. Exponential Linear Unit (ELU):

- Description: , where is a small positive constant

- Purpose: Smoothens the transition for negative input values, helping with the vanishing gradient problem.

The choice of activation function depends on factors such as the nature of the data, the architecture of the autoencoder, and the specific requirements of the task at hand. Experimentation is often necessary to determine the most effective activation functions for a particular scenario.

A list of main formulas for commonly used **loss functions** in autoencoder-decoder architectures:

1. Mean Squared Error (MSE) Loss:

Formula:

Where : Input Data, : Reconstructed output, and : Number of data points.

1. Binary Cross-Entropy Loss:

Formula:

Where : Ground truth binary value,

:Predicted probability of being in class 1

:Number of data points

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*