Autoencoders -Machine Learning

Last Updated : 06 Dec, 2023

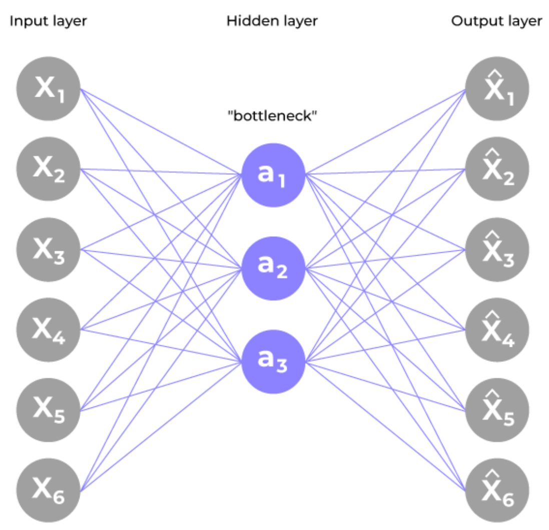
At the heart of **deep learning** lies the neural network, an intricate interconnected system of nodes that mimics the human brain's neural architecture. Neural networks excel at discerning intricate patterns and representations within vast datasets, allowing them to make predictions, classify information, and generate novel insights. **Autoencoders** emerge as a fascinating subset of neural networks, offering a unique approach to unsupervised learning. Autoencoders are an adaptable and strong class of architectures for the dynamic field of deep learning, where neural networks develop constantly to identify complicated patterns and representations. With their ability to learn effective representations of data, these unsupervised learning models have received considerable attention and are useful in a wide variety of areas, from image processing to anomaly detection.

What are Autoencoders?

Autoencoders are a specialized class of algorithms that can learn efficient representations of input data with no need for labels. It is a class of artificial neural networks designed for unsupervised learning. Learning to compress and effectively represent input data without specific labels is the essential principle of an automatic decoder. This is accomplished using a two-fold structure that consists of an encoder and a decoder. The encoder transforms the input data into a reduced-dimensional representation, which is often referred to as "latent space" or "encoding". From that representation, a decoder rebuilds the initial input. For the network to gain meaningful patterns in data, a process of encoding and decoding facilitates the definition of essential features.

Architecture of Autoencoder in Deep Learning

The general architecture of an autoencoder includes an encoder, decoder, and bottleneck layer.



1. Encoder
   - Input layer take raw input data
   - The hidden layers progressively reduce the dimensionality of the input, capturing important features and patterns. These layer compose the encoder.
   - The bottleneck layer (latent space) is the final hidden layer, where the dimensionality is significantly reduced. This layer represents the compressed encoding of the input data.

2. Decoder
   - The bottleneck layer takes the encoded representation and expands it back to the dimensionality of the original input.
   - The hidden layers progressively increase the dimensionality and aim to reconstruct the original input.
   - The output layer produces the reconstructed output, which ideally should be as close as possible to the input data.

3. The loss function used during training is typically a reconstruction loss, measuring the difference between the input and the reconstructed output. Common choices include mean squared error (MSE) for continuous data or binary cross-entropy for binary data.
4. During training, the autoencoder learns to minimize the reconstruction loss, forcing the network to capture the most important features of the

input data in the bottleneck layer.

After the training process, only the encoder part of the autoencoder is retained to encode a similar type of data used in the training process. The different ways to constrain the network are: –

- **Keep small Hidden Layers:** If the size of each hidden layer is kept as small as possible, then the network will be forced to pick up only the representative features of the data thus encoding the data.
- **Regularization:** In this method, a loss term is added to the cost function which encourages the network to train in ways other than copying the input.
- **Denoising:** Another way of constraining the network is to add noise to the input and teach the network how to remove the noise from the data.
- **Tuning the Activation Functions:** This method involves changing the activation functions of various nodes so that a majority of the nodes are dormant thus, effectively reducing the size of the hidden layers.

Types of Autoencoders

There are diverse types of autoencoders and analyze the advantages and disadvantages associated with different variation:

Denoising Autoencoder

Denoising autoencoder works on a partially corrupted input and trains to recover the original undistorted image. As mentioned above, this method is an effective way to constrain the network from simply copying the input and thus learn the underlying structure and important features of the data.

Advantages

1. This type of autoencoder can extract important features and reduce the noise or the useless features.
2. Denoising autoencoders can be used as a form of data augmentation, the restored images can be used as augmented data thus generating additional training samples.

Disadvantages

1. Selecting the right type and level of noise to introduce can be challenging and may require domain knowledge.
2. Denoising process can result into loss of some information that is needed from the original input. This loss can impact accuracy of the output.

Sparse Autoencoder

This type of autoencoder typically contains more hidden units than the input but only a few are allowed to be active at once. This property is called the sparsity of the network. The sparsity of the network can be controlled by either manually zeroing the required hidden units, tuning the activation functions or by adding a loss term to the cost function.

Advantages

1. The sparsity constraint in sparse autoencoders helps in filtering out noise and irrelevant features during the encoding process.
2. These autoencoders often learn important and meaningful features due to their emphasis on sparse activations.

Disadvantages

1. The choice of hyperparameters play a significant role in the performance of this autoencoder. Different inputs should result in the activation of different nodes of the network.
2. The application of sparsity constraint increases computational complexity.

Variational Autoencoder

Variational autoencoder makes strong assumptions about the distribution of latent variables and uses the **Stochastic Gradient Variational Bayes** estimator in the training process. It assumes that the data is generated by a **Directed Graphical Model** and tries to learn an approximation to $q_\phi(z|x)$to the conditional property $q_\theta(z|x)$where $\phi$and $\theta$are the parameters of the encoder and the decoder respectively.

Advantages

1. Variational Autoencoders are used to generate new data points that resemble the original training data. These samples are learned from the latent space.
2. Variational Autoencoder is probabilistic framework that is used to learn a compressed representation of the data that captures its underlying structure and variations, so it is useful in detecting anomalies and data exploration.

Disadvantages

1. Variational Autoencoder use approximations to estimate the true distribution of the latent variables. This approximation introduces some level of error, which can affect the quality of generated samples.
2. The generated samples may only cover a limited subset of the true data distribution. This can result in a lack of diversity in generated samples.

Convolutional Autoencoder

Convolutional autoencoders are a type of autoencoder that use convolutional neural networks (CNNs) as their building blocks. The encoder consists of multiple layers that take a image or a grid as input and pass it through different convolution layers thus forming a compressed representation of the input. The decoder is the mirror image of the encoder it deconvolves the compressed representation and tries to reconstruct the original image.

Advantages

1. Convolutional autoencoder can compress high-dimensional image data into a lower-dimensional data. This improves storage efficiency and transmission of image data.
2. Convolutional autoencoder can reconstruct missing parts of an image. It can also handle images with slight variations in object position or orientation.

Disadvantages

1. These autoencoder are prone to overfitting. Proper regularization techniques should be used to tackle this issue.

2. Compression of data can cause data loss which can result in reconstruction of a lower quality image.

Implementation of Autoencoders

We've created an autoencoder comprising two Dense layers: an encoder responsible for condensing the images into a 64-dimensional latent vector, and a decoder tasked with reconstructing the initial image based on this latent space.

Import necessary libraries

For the implementation, we are going to import matplotlib, numpy, pandas, sklearn and keras.

Python3

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score, precision_score, recall_sco
from sklearn.model_selection import train_test_split
from keras import layers, losses
from keras.datasets import mnist
from keras.models import Model
```

Load the MNIST dataset

Python3

```
# Loading the MNIST dataset and extracting training and testing data
(x_train, _), (x_test, _) = mnist.load_data()

# Normalizing pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Displaying the shapes of the training and testing datasets
print("Shape of the training data:", x_train.shape)
print("Shape of the testing data:", x_test.shape)
```

Output:

```
Shape of the training data: (60000, 28, 28)
Shape of the testing data: (10000, 28, 28)
```

Define a basic Autoencoder

In the following code snippet,

- SimpleAutoencoder class is defined.
- Constructor initializes the autoencoder with specified latent dimensions and data shape
- The encoder and decoder architectures is defined using Sequential model
- The call method defines the forward pass of the autoencoder, where input data is passed through the encoder to obtain encoded data and then through the decoder to obtain the decoded data.

Python3

```
# Definition of the Autoencoder model as a subclass of the TensorFlow Mc

class SimpleAutoencoder(Model):
	def __init__(self,latent_dimensions , data_shape):
		super(SimpleAutoencoder, self).__init__()
		self.latent_dimensions = latent_dimensions
		self.data_shape = data_shape

		# Encoder architecture using a Sequential model
		self.encoder = tf.keras.Sequential([
			layers.Flatten(),
			layers.Dense(latent_dimensions, activation='relu'),
		])

		# Decoder architecture using another Sequential model
		self.decoder = tf.keras.Sequential([
			layers.Dense(tf.math.reduce_prod(data_shape), activation='si
			layers.Reshape(data_shape)
		])

	# Forward pass method defining the encoding and decoding steps
	def call(self, input_data):
		encoded_data = self.encoder(input_data)
		decoded_data = self.decoder(encoded_data)
```

```
            return decoded_data

    # Extracting shape information from the testing dataset
    input_data_shape = x_test.shape[1:]

    # Specifying the dimensionality of the latent space
    latent_dimensions = 64

    # Creating an instance of the SimpleAutoencoder model
    simple_autoencoder = SimpleAutoencoder(latent_dimensions, input_data_sha
```

Compile and Fit Autoencoder

Python3

```
simple_autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredErro

simple_autoencoder.fit(x_train, x_train,
                epochs=1,
                shuffle=True,
                validation_data=(x_test, x_test))
```

Output:

```
Epoch 1/10
1875/1875 [==============================] - 12s 6ms/step - loss:
0.0243 - val_loss: 0.0091
Epoch 2/10
1875/1875 [==============================] - 16s 9ms/step - loss:
0.0069 - val_loss: 0.0054
Epoch 3/10
1875/1875 [==============================] - 15s 8ms/step - loss:
0.0051 - val_loss: 0.0046
Epoch 4/10
1875/1875 [==============================] - 8s 5ms/step - loss:
0.0045 - val_loss: 0.0043
Epoch 5/10
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0043 - val_loss: 0.0041
Epoch 6/10
```

```
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0042 - val_loss: 0.0041
Epoch 7/10
1875/1875 [==============================] - 7s 4ms/step - loss:
0.0041 - val_loss: 0.0040
```

Visualize the original and reconstructed data

Python3

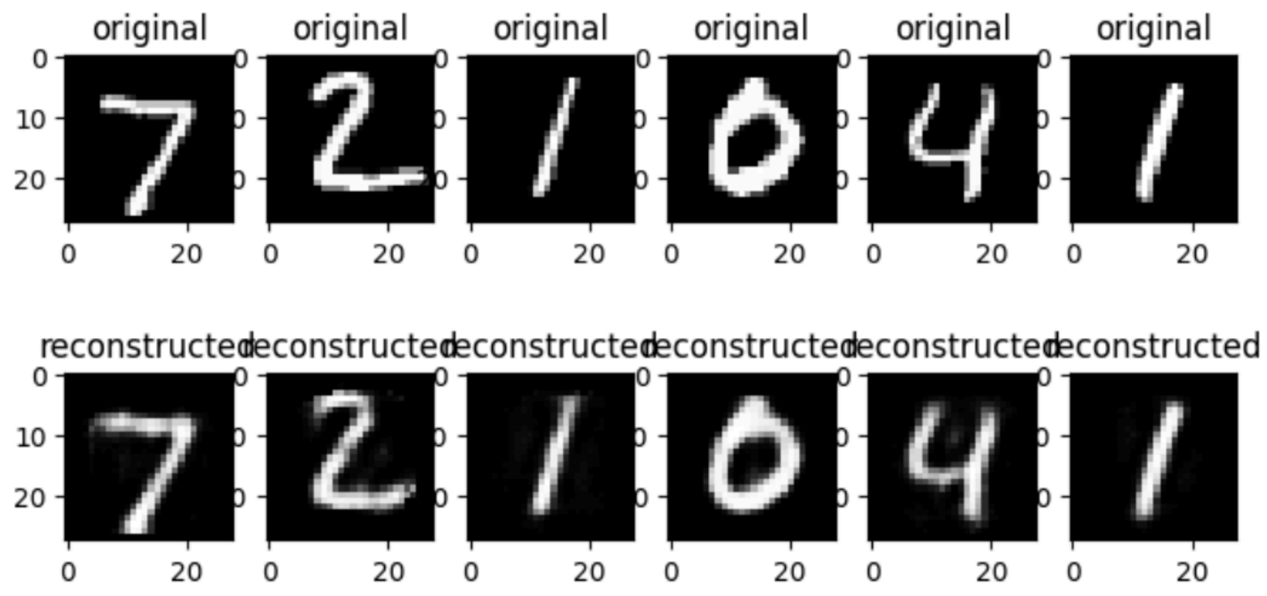
```python
encoded_imgs = simple_autoencoder.encoder(x_test).numpy()
decoded_imgs = simple_autoencoder.decoder(encoded_imgs).numpy()

n = 6
plt.figure(figsize=(8, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i])
  plt.title("reconstructed")
  plt.gray()

plt.show()
```

Output:



Conclusion

In conclusion, we have defined Autoencoders as robust and flexible classes of architectures in the dynamic land of deep learning. Their significance in tasks such as image compression and anomaly detection is underlined by the fact that they are capable of learning effective data representations, together with their ability to adapt to different autoencoder types. Autoencoders remain a major player in the area of deep learning, playing an increasingly important role as we explore and develop new approaches for understanding data patterns and representations.

Also Check:

- How Autoencoders works?
- Implementing an Autoencoder in PyTorch
- Building an Auto-Encoder using Keras
- How is Autoencoder different from PCA

Frequently Asked Questions (FAQs)

1. What is CNN autoencoder?

> *A CNN autoencoder is an autoencoder architecture that incorporates convolutional layers in both the encoder and decoder parts of the network. They are suited for handling high-dimensional input data with spatial structure, such as images.*

2. How does an Autoencoder work?

> *The autoencoder works by encoding the input data into a lower-dimensional representation, often called the latent space or bottleneck, using the encoder. The decoder then reconstructs the input data from this lower-dimensional representation. The network is trained to minimize the difference between the input and the reconstructed output.*

3. Is autoencoder supervised or unsupervised?

> *Autoencoders are primarily used for unsupervised learning, as they do not require labeled data during the training phase. However, they can*