Autoencoders

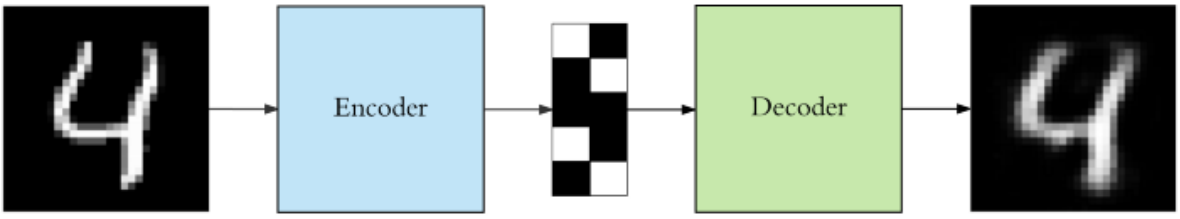
Topics Covered

1. Autoencoder
2. Applications of Autoencoders
3. Types of Autoencoders
4. Autoencoders for Dimensionality Reduction
5. Using Autoencoders for Dimensionality Reduction Using Tensorflow
6. Summary

Autoencoder

Autoencoders are a specific type of feedforward neural networks where the input is the same as the output. They compress the input into a lower-dimensional *code* and then reconstruct the output from this representation. The code is a compact "summary" or "compression" of the input, also called the *latent*-space representation.

An autoencoder consists of 3 components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code.



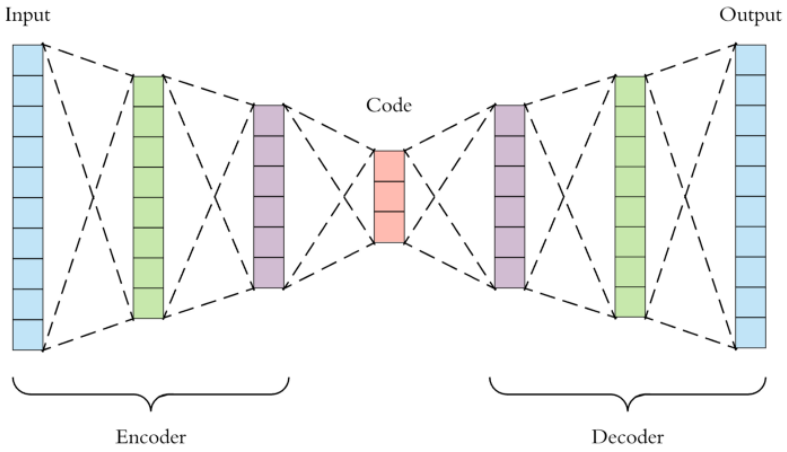
To build an autoencoder we need 3 things: an encoding method, decoding method, and a loss function to compare the output with the target.

The auto-encoder itself is unsupervised. There is no target feature you're trying to predict, what you're trying to do is find a low-dimensional representation of the inputs instead.

It is worth noting that auto-encoders are often used in supervised learning; you train an auto-encoder on the inputs alone, and then use the low-dimensional encoding as inputs to the supervised learning network. This is not dissimilar to first using PCA to reduce the dimensionality, and then training a prediction model on the reduced dimensionality inputs.

Architecture of Autoencoders

Let's explore the details of the encoder, code and decoder. Both the encoder and decoder are fully-connected feedforward neural networks. Code is a single layer of an ANN with the dimensionality of our choice. The number of nodes in the code layer (code size) is a hyperparameter that we set before training the autoencoder.



This is a more detailed visualization of an autoencoder. First the input passes through the encoder, which is a fully-connected ANN, to produce the code. The decoder, which has the similar ANN structure, then produces the output only using the code. The goal is to get an output identical with the input. Note that the decoder architecture is the mirror image of the encoder. This is not a requirement but it's typically the case. The only requirement is the dimensionality of the input and output needs to be the same. Anything in the middle can be played with.

There are 4 hyperparameters that we need to set before training an autoencoder:

- Code size: number of nodes in the middle layer. Smaller size results in more compression.
- Number of layers: the autoencoder can be as deep as we like. In the figure above we have 2 layers in both the encoder and decoder, without considering the input and output.
- Number of nodes per layer: the autoencoder architecture we're working on is called a *stacked autoencoder* since the layers are stacked one after another. Usually stacked autoencoders look like a "sandwitch". The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. Also the decoder is symmetric to the encoder in terms of layer structure. As noted above this is not necessary and we have total control over these parameters.
- Loss function: we either use *mean squared error (mse)* or *binary crossentropy*. If the input values are in the range [0, 1] then we typically use crossentropy, otherwise we use the mean squared error.
- Autoencoders are trained the same way as ANNs via backpropagation.

Applications 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.

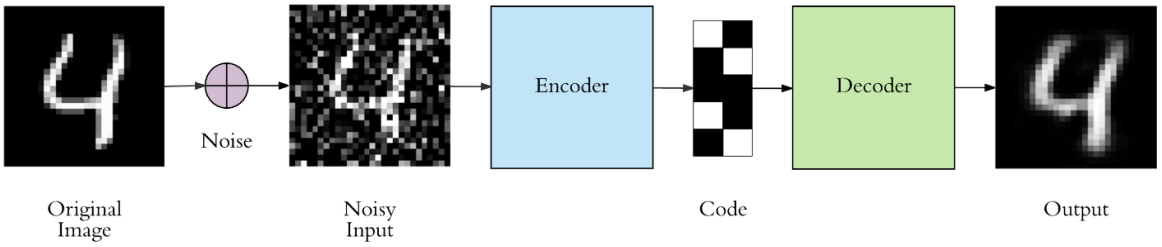
Furthermore, while dimensionality reduction procedures like PCA can only perform linear dimensionality reductions, undercomplete autoencoders can perform large-scale non-linear dimensionality reductions.

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.

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.

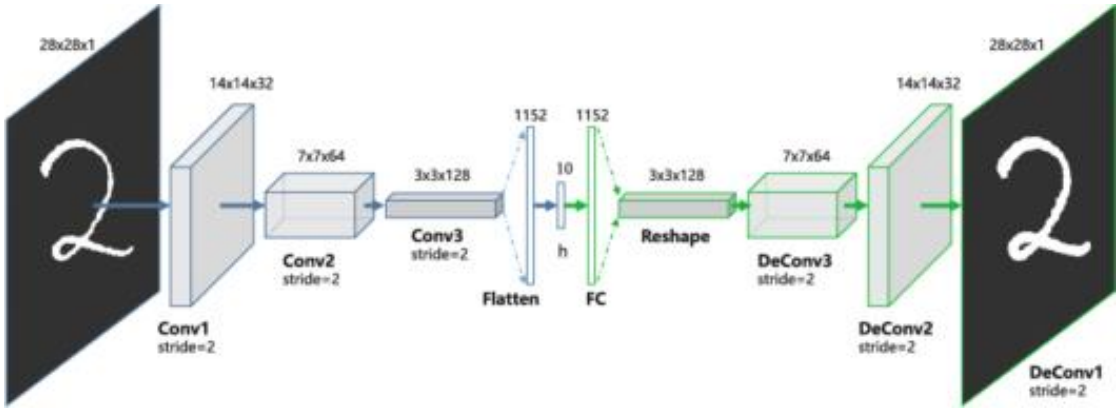
As a result, the outlier image gives off a very high reconstruction loss and can easily be identified as an anomaly with the help of a proper threshold.

Types of Autoencoders

There are many types of autoencoders and some of them are mentioned below with a brief description

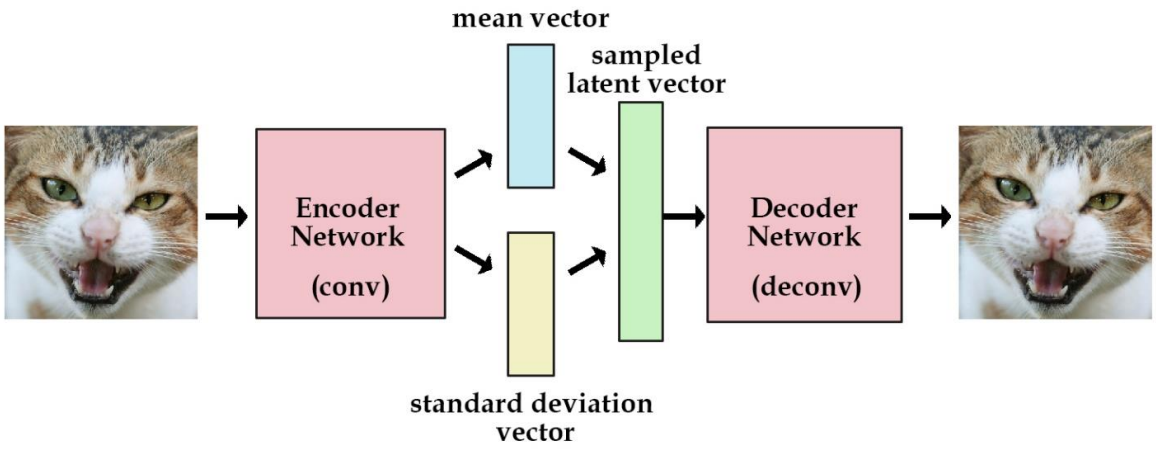
Convolutional Autoencoder: Convolutional Autoencoders(CAE) learn to encode the input in a set of simple signals and then reconstruct the input from them.

In addition, we can modify the geometry or generate the reflectance of the image by using CAE. In this type of autoencoder, encoder layers are known as convolution layers and decoder layers are also called deconvolution layers. The deconvolution side is also known as upsampling or transpose convolution.



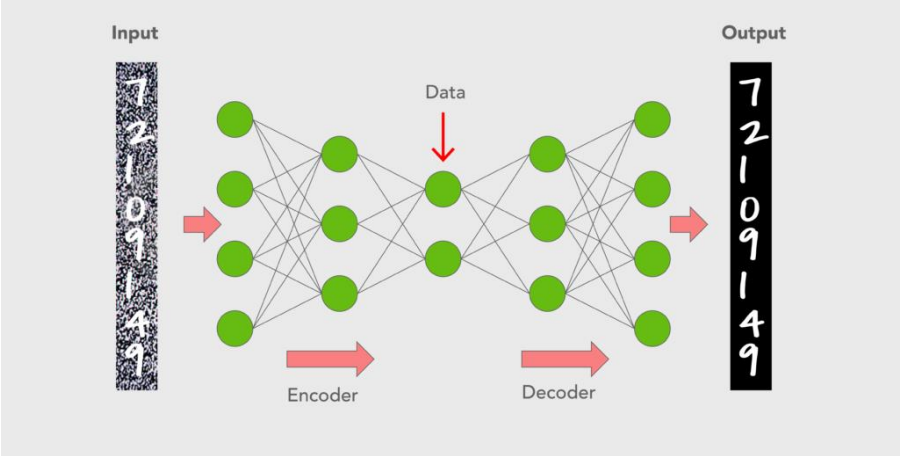
Variational Autoencoders: This type of autoencoder can generate new images just like GANs. Variational autoencoder models tend to make strong assumptions related to the distribution of latent variables.

They use a variational approach for latent representation learning, which results in an additional loss component and a specific estimator for the training algorithm called the Stochastic Gradient Variational Bayes estimator. The probability distribution of the latent vector of a variational autoencoder typically matches the training data much closer than a standard autoencoder. As VAEs are much more flexible and customisable in their generation behaviour than GANs, they are suitable for art generation of any kind.



Denoising autoencoders: Denoising autoencoders add some noise to the input image and learn to remove it.

Thus avoiding to copy the input to the output without learning features about the data. These autoencoders take a partially corrupted input while training to recover the original undistorted input. The model learns a vector field for mapping the input data towards a lower-dimensional manifold which describes the natural data to cancel out the added noise. By this means, the encoder will extract the most important features and learn a more robust representation of the data.



Deep autoencoders: A deep autoencoder is composed of two symmetrical deep-belief networks having four to five shallow layers.

One of the networks represents the encoding half of the net and the second network makes up the decoding half. They have more layers than a simple autoencoder and thus are able to learn more complex features. The layers are restricted Boltzmann machines, the building blocks of deep-belief networks.

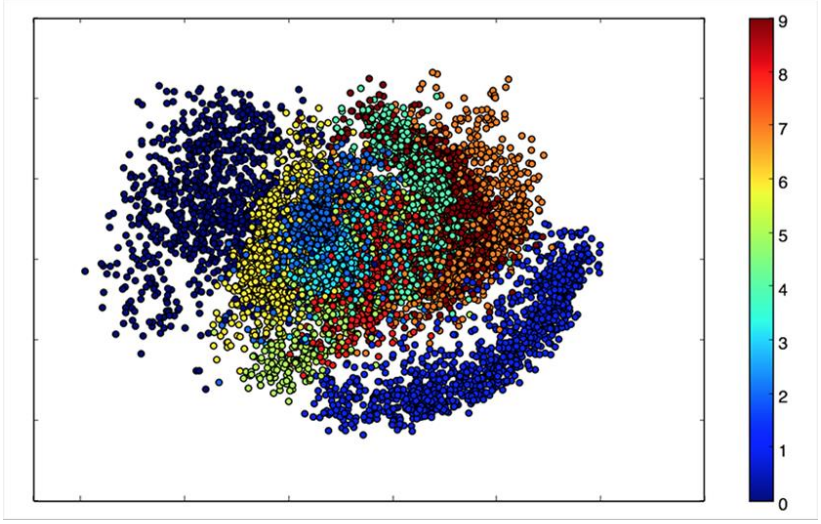
Autoencoders for Dimensionality Reduction

Autoencoders are mainly a dimensionality reduction (or compression) algorithm with a couple of important properties:

- Data-specific: Autoencoders are only able to meaningfully compress data similar to what they have been trained on. Since they learn features specific for the given training data, they are different than a standard data compression algorithm like gzip. So we can't expect an autoencoder trained on handwritten digits to compress landscape photos.

- Lossy: The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation. If you want lossless compression, they are not the way to go.

- Unsupervised: To train an autoencoder we don't need to do anything fancy, just throw the raw input data at it. Autoencoders are considered an unsupervised learning technique since they don't need explicit labels to train on. But to be more precise they are self-supervised because they generate their own labels from the training data.



This is latent space for hand written numbers.

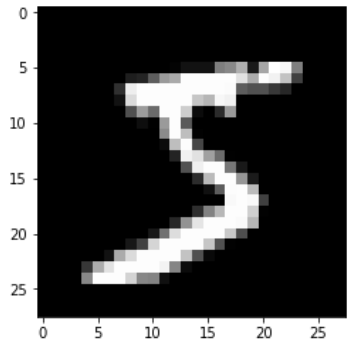
Using Autoencoders for Dimensionality Reduction Using Tensorflow

We will use the MNIST dataset of tensorflow, where the images are 28 x 28 dimensions, in other words, if we flatten the dimensions, we are dealing with **784 dimensions**. Our goal is to reduce the dimensions, from **784** to **2**, by including as much information as possible.

```
### Encoder
encoder = Sequential()
encoder.add(Flatten(input_shape=[28,28]))
encoder.add(Dense(400,activation="relu"))
encoder.add(Dense(200,activation="relu"))
encoder.add(Dense(100,activation="relu"))
encoder.add(Dense(50,activation="relu"))
encoder.add(Dense(2,activation="relu"))



### Decoder
decoder = Sequential()
decoder.add(Dense(50,input_shape=[2],activation='relu'))
decoder.add(Dense(100,activation='relu'))
decoder.add(Dense(200,activation='relu'))
decoder.add(Dense(400,activation='relu'))
decoder.add(Dense(28 * 28, activation="relu"))
decoder.add(Reshape([28, 28]))

### Autoencoder
autoencoder = Sequential([encoder,decoder])
autoencoder.compile(loss="mse")
autoencoder.fit(X_train,X_train,epochs=50)


encoded_2dim = encoder.predict(X_train)


# The 2D
AE = pd.DataFrame(encoded_2dim, columns = ['X1', 'X2'])

AE['target'] = y_train

sns.lmplot(x='X1', y='X2', data=AE, hue='target', fit_reg=False, size=10)
```

Every image in the MNSIT Dataset is a “gray scale” image of 28 x 28 dimensions. Let’s have a look at the first image.

```
plt.imshow(X_train[0], cmap='gray')
```



This is one example of the number 5 and the corresponding 28 x 28 array is the:

```
X_train[0].shape
```

(28, 28)

```
X_train[0]
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
```

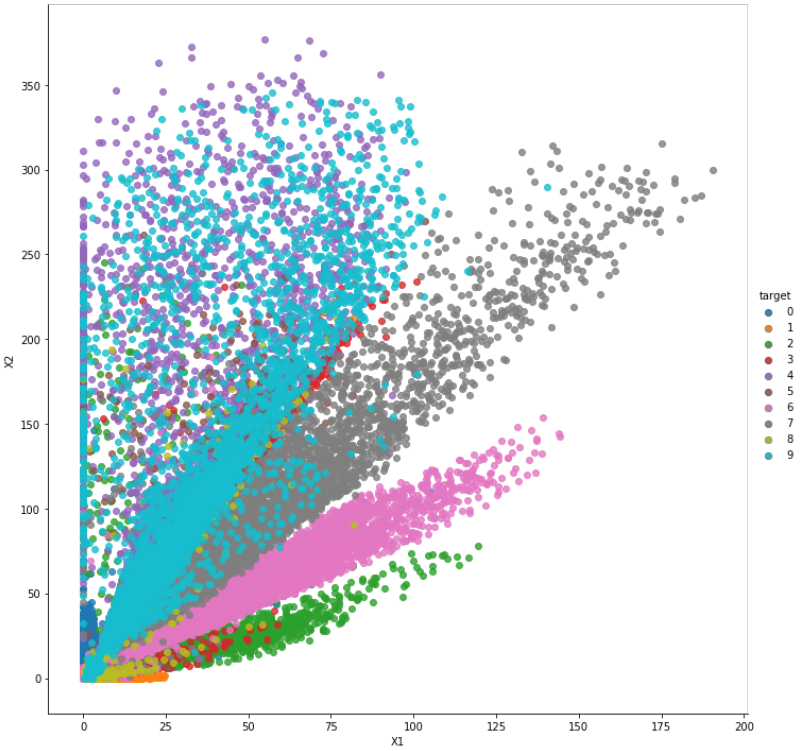
```
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
      ...,
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

Our goal is to reduce the dimensions of MNIST images from **784** to **2** and to represent them in a scatter plot.

Results of Autoencoders

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,8))
sns.lmplot(x='X1', y='X2', data=AE, hue='target', fit_reg=False, size=10)
```

We ended up with two dimensions and we can see the corresponding scatterplot below, using as labels the digits.



As we can see from the plot above, only by taking into account 2 dimensions out of 784, we were able somehow to distinguish between the different images (digits). Hence, keep in mind, that apart from PCA and t-SNE, we can also apply Autoencoders for Dimensionality Reduction

Summary

- Autoencoders are a specific type of feedforward neural networks where the input is the same as the output.
- An autoencoder consists of 3 components: encoder, code and decoder.
- The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code.
- Autoencoders are mainly a dimensionality reduction (or compression) algorithm.
- Autoencoders are considered an unsupervised learning technique since they don't need explicit labels to train on.
- Autoencoders are only able to meaningfully compress data similar to what they have been trained on.
- The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation.
- A deep autoencoder is composed of two symmetrical deep-belief networks having four to five shallow layers.
- Convolutional Autoencoders(CAE) learn to encode the input in a set of simple signals and then reconstruct the input from them.
- Variational autoencoder models tend to make strong assumptions related to the distribution of latent variables.
- Denoising autoencoders add some noise to the input image and learn to remove it.