**Autoencoders**

**Olga Lyudchik**

**Non-member state summer student**

**Supervisor: Dr. Jean-Roch Valery VLIMANT (EP-UCM)**

# 

# Contents List

[What are autoencoders?](#h.gv0b7uvmwhd0)

[What are autoencoders good for?](#h.643ehtcujrwr)

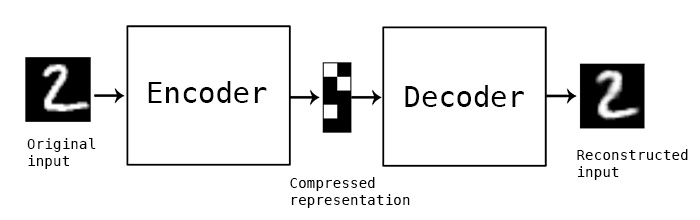
[What is the big deal with autoencoders?](#h.88uzqdim262)

[Variational autoencoder (VAE)](#h.o5dazhg3j157)

[Basic autoencoder](#h.14c9ltrw5c08)

# 

# What are autoencoders?



"Autoencoding" is a data compression algorithm where the compression and decompression functions are 1) data-specific, 2) lossy, and 3) learned automatically from examples rather than engineered by a human. Additionally, in almost all contexts where the term "autoencoder" is used, the compression and decompression functions are implemented with neural networks.

1) Autoencoders are data-specific, which means that they will only be able to compress data similar to what they have been trained on. An autoencoder trained on pictures of faces would do a rather poor job of compressing pictures of trees, because the features it would learn would be face-specific.

2) Autoencoders are lossy, which means that the decompressed outputs will be degraded compared to the original inputs. This differs from lossless arithmetic compression.

3) Autoencoders are learned automatically from data examples, which is a useful property: it means that it is easy to train specialized instances of the algorithm that will perform well on a specific type of input. It doesn't require any new engineering, just appropriate training data.

To build an autoencoder, you need three things:

1. an encoding function,
2. a decoding function,
3. a distance function between the amount of information loss between the compressed representation of your data and the decompressed representation (i.e. a "loss" function).

The encoder and decoder will be chosen to be parametric functions (typically neural networks), and to be differentiable with respect to the distance function, so the parameters of the encoding/decoding functions can be optimize to minimize the reconstruction loss, using Stochastic Gradient Descent.

# What are autoencoders good for?

Today two interesting practical applications of autoencoders are

1. data denoising,
2. dimensionality reduction for data visualization.

With appropriate dimensionality and sparsity constraints, autoencoders can learn data projections that are more interesting than PCA or other basic techniques.

For 2D visualization specifically, [t-SNE](https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding) is probably the best algorithm around, but it typically requires relatively low-dimensional data. So a good strategy for visualizing similarity relationships in high-dimensional data is to start by using an autoencoder to compress your data into a low-dimensional space (e.g. 32 dimensional), then use t-SNE for mapping the compressed data to a 2D plane.

# What is the big deal with autoencoders?

Otherwise, one reason why they have attracted so much research and attention is because they have long been thought to be a potential avenue for solving the problem of unsupervised learning, i.e. the learning of useful representations without the need for labels. Then again, autoencoders are not a true unsupervised learning technique (which would imply a different learning process altogether), they are a self-supervised technique, a specific instance of supervised learning where the targets are generated from the input data. In order to get self-supervised models to learn interesting features, you have to come up with an interesting synthetic target and loss function, and that's where problems arise: merely learning to reconstruct your input in minute detail might not be the right choice here. At this point there is significant evidence that focusing on the reconstruction of a picture at the pixel level, for instance, is not conductive to learning interesting, abstract features of the kind that label-supervized learning induces (where targets are fairly abstract concepts "invented" by humans such as "dog", "car"...). In fact, one may argue that the best features in this regard are those that are the worst at exact input reconstruction while achieving high performance on the main task that you are interested in (classification, localization, etc).

# Anomaly Detection Using Autoencoders

Automatically removing outliers from unlabeled data operates in an unsupervised mode. For this problem, methods in literature explicitly or implicitly make an assumption that inliers are located in dense areas while outliers are not. The dense areas can be estimated by statistical methods, neighbor-based methods, and reconstruction-based methods. For example, the methods in compute PCA projections on data, and those having large projection variances are determined as outliers.

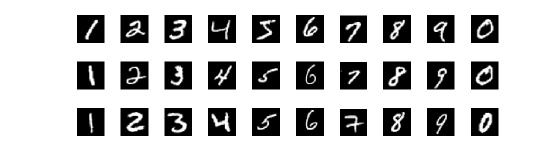
# MNIST Dataset

The problem of handwriting recognition is to interpret intelligible handwritten input auto-matically, which is of great interest in the pattern recognition research community because of its applicability to many ﬁelds towards more convenient input devices and more eﬃcient data organization and processing. As one of the fundament problems in designing practical recog-nition systems, the recognition of handwritten digits is an active research ﬁeld. Immediate applications of the digit recognition techniques include postal mail sorting, automatically address reading and mail routing, bank check processing, etc. As a benchmark for testing classiﬁcation algorithms, the MNIST dataset has been widely used to design novel handwritten digit recognition systems. There are a great amountof studies based on MNIST dataset reported in the literature, suggesting many diﬀerent methods. One of the major challenges in the recognition of handwritten digits is the within class variance, because people do not always write the same digit in exactly the same way. Many feature extraction approaches have been proposed trying to characterize the shape invariance within a class to improve the discrimination ability. Experiments have shown that by extracting direction features, local structure features or curvature features, the accuracy and eﬃciency of many classiﬁers could be improved signiﬁcantly. In this report we train and test a set of classiﬁers on the MNIST database for pattern analysis in solving the handwritten digit recognition problem.

Data Description

The MNIST database contains 60,000 digits ranging from 0 to 9 for training the digit recognition system, and another 10,000 digits as test data. Each digit is normalized and centeredin a gray-level image with size 28×28, or with 784 pixel in total as the features. Some examples are shown in Figure 1.

Figure 1:



Examples of MNIST data set

Since the original dimension is quite large, the dimensionality reduction becomes necessary. First we extract the principle components from the original data. As shown in Figure 2, the ﬁrst 50 principle components can interpret approximately 97% of total information (in termsof the total variance retained), which suﬃces to be representative and informative. We thuschoose ﬁrst 50 principle components as the extracted features.

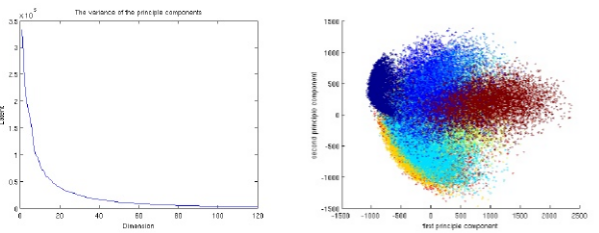


Figure 2: variance of Principle Components (left ) and scatter plot on PC Coordinate(right )