Autoencoders - an Introduction

What they are and what you can do with them

Dr. Umberto Michelucci[1]

[1] Presenting author, umberto.michelucci@hslu.ch

March 13, 2023

Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Table of Contents

Introduction

Reference / Paper / GitHub

You can find a complete discussion at [1]
`https://arxiv.org/pdf/2201.03898.pdf`

GitHub:
`https://github.com/toelt-llc/ETH-ZURICH-GDSC-WORKSHOPS-2022`

Notation

Normally in Machine learning we use the following notation for **supervised learning**:

$\mathbf{x}_i$ indicates the input observations (e.g. images)
y_i indicate the expected value or label.

Now suppose we have only unlabelled observations. We have only a training dataset S_T with M observations:

$$S_T = \{\mathbf{x}_i \mid i = 1, ..., M\} \quad \text{with } \mathbf{x}_i \in \mathbb{R}^n$$

Notation

Normally in Machine learning we use the following notation for **supervised learning**:

$\mathbf{x}_i$ indicates the input observations (e.g. images)
y_i indicate the expected value or label.

Now suppose we have only unlabelled observations. We have only a training dataset S_T with M observations:

$$S_T = \{\mathbf{x}_i \mid i = 1, ..., M\} \quad \text{with } \mathbf{x}_i \in \mathbb{R}^n$$

Notation

Normally in Machine learning we use the following notation for **supervised learning**:

$\mathbf{x}_i$ indicates the input observations (e.g. images)
y_i indicate the expected value or label.

Now suppose we have only unlabelled observations. We have only a training dataset S_T with M observations:

$$S_T = \{\mathbf{x}_i \mid i = 1, ..., M\} \quad \text{with } \mathbf{x}_i \in \mathbb{R}^n$$

Notation

Normally in Machine learning we use the following notation for **supervised learning**:

$\mathbf{x}_i$ indicates the input observations (e.g. images)
y_i indicate the expected value or label.

Now suppose we have only unlabelled observations. We have only a training dataset S_T with M observations:

$$S_T = \{\mathbf{x}_i \mid i = 1, ..., M\} \quad \text{with } \mathbf{x}_i \in \mathbb{R}^n$$

History

Autoencoders were first introduced[1] by Rumelhart, Hinton, and Williams [2] in 1986 with the goal of learning to reconstruct the input observations $\mathbf{x}_i$ with the lowest error possible.

If you have problems imagining what that means, think of having a dataset made of images. An autoencoder would be an algorithm that can give as output an image that is as similar as possible to the input one.

[1] https://web.stanford.edu/class/psych209a/ReadingsByDate/02_06/PDPVolIChapter8.pdf

History

Autoencoders were first introduced[1] by Rumelhart, Hinton, and Williams [2] in 1986 with the goal of learning to reconstruct the input observations $\mathbf{x}_i$ with the lowest error possible.

If you have problems imagining what that means, think of having a dataset made of images. An autoencoder would be an algorithm that can give as output an image that is as similar as possible to the input one.

[1] https://web.stanford.edu/class/psych209a/ReadingsByDate/02_06/PDPVolIChapter8.pdf

Definition

Definition

An autoencoder is a type of algorithm with the primary purpose of learning an "informative" representation of the data that can be used for different applications [3] by learning to reconstruct a set of input observations well enough.

What does it mean **well enough**?

What does it mean **informative**?

Definition

Definition

An autoencoder is a type of algorithm with the primary purpose of learning an ”informative” representation of the data that can be used for different applications [3] by learning to reconstruct a set of input observations well enough.

What does it mean **well enough**?

What does it mean **informative**?

Definition

Definition

An autoencoder is a type of algorithm with the primary purpose of learning an "informative" representation of the data that can be used for different applications [3] by learning to reconstruct a set of input observations well enough.

What does it mean **well enough**?

What does it mean **informative**?

The *informative* aspect is important

Building an algorithm that reconstruct perfectly the input is very easy...

It is enough to use the identify function $I(x) = x$

This is not useful at all (and dumb)

The *informative* aspect is important

Building an algorithm that reconstruct perfectly the input is very easy...

It is enough to use the identify function $I(x) = x$

This is not useful at all (and dumb)

The *informative* aspect is important

Building an algorithm that reconstruct perfectly the input is very easy...

It is enough to use the identify function $I(x) = x$

This is not useful at all (and dumb)

The *informative* representation

For example, an informative representation of hand-written digits could be the number of lines required to write each number or the angle of each line and how they connect.

Learning how to write numbers certainly does not require to learn the gray values of each pixel in the input image.

We humans do not certainly learn to write by filling pixels with gray values.

The *informative* representation

For example, an informative representation of hand-written digits could be the number of lines required to write each number or the angle of each line and how they connect.

Learning how to write numbers certainly does not require to learn the gray values of each pixel in the input image.

We humans do not certainly learn to write by filling pixels with gray values.

The *informative* representation

For example, an informative representation of hand-written digits could be the number of lines required to write each number or the angle of each line and how they connect.

Learning how to write numbers certainly does not require to learn the gray values of each pixel in the input image.

We humans do not certainly learn to write by filling pixels with gray values.

Structure of an autoencoder

Structure of an autoencoder

The structure of an autoencoder looks typically like this

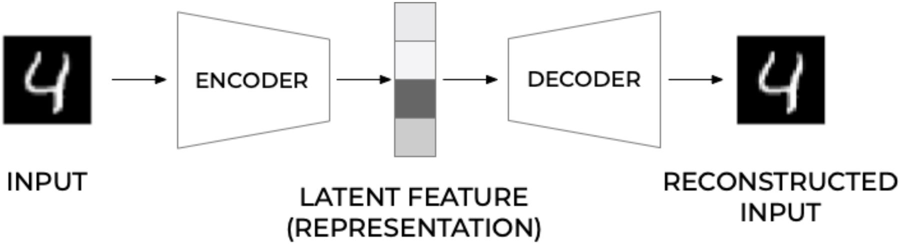

Figure: The typical structure of an autoencoder.

Elements of an autoencoder

The main elements of an autoencoder are the following:

1. **Encoder**: generally speaking is a function $\mathbf{h}_i = g(\mathbf{x}_i)$ that depends on some parameters;
2. **Latent Features**: the $\mathbf{h}_i$ are normally just an array of numbers (in 1 or 2 dimension depending on the function $g()$);
3. **Decoder**: a second function that has as output the reconstructed image $\tilde{\mathbf{x}}_i = f(\mathbf{x}_i) = f(g(\mathbf{x}_i))$;
4. The functions $f()$ and $g()$ are typically neural networks.

Elements of an autoencoder

Lucerne University of Applied Sciences and Arts
HOCHSCHULE LUZERN

The main elements of an autoencoder are the following:

1. **Encoder**: generally speaking is a function $\mathbf{h}_i = g(\mathbf{x}_i)$ that depends on some parameters;
2. **Latent Features**: the $\mathbf{h}_i$ are normally just an array of numbers (in 1 or 2 dimension depending on the function $g()$);
3. **Decoder**: a second function that has as output the reconstructed image $\tilde{\mathbf{x}}_i = f(\mathbf{x}_i) = f(g(\mathbf{x}_i))$;
4. The functions $f()$ and $g()$ are typically neural networks.

Elements of an autoencoder

The main elements of an autoencoder are the following:

1. **Encoder**: generally speaking is a function $\mathbf{h}_i = g(\mathbf{x}_i)$ that depends on some parameters;
2. **Latent Features**: the $\mathbf{h}_i$ are normally just an array of numbers (in 1 or 2 dimension depending on the function $g()$);
3. **Decoder**: a second function that has as output the reconstructed image $\tilde{\mathbf{x}}_i = f(\mathbf{x}_i) = f(g(\mathbf{x}_i))$;
4. The functions $f()$ and $g()$ are typically neural networks.

Elements of an autoencoder

The main elements of an autoencoder are the following:

1. **Encoder**: generally speaking is a function $\mathbf{h}_i = g(\mathbf{x}_i)$ that depends on some parameters;
2. **Latent Features**: the $\mathbf{h}_i$ are normally just an array of numbers (in 1 or 2 dimension depending on the function $g()$);
3. **Decoder**: a second function that has as output the reconstructed image $\tilde{\mathbf{x}}_i = f(\mathbf{x}_i) = f(g(\mathbf{x}_i))$;
4. The functions $f()$ and $g()$ are typically neural networks.

Another take on the *informative* aspect

An **informative representation** means to obtain a latent representation $\mathbf{h}_i$ that is useful for other purposes.

Bottleneck in autoencoders

1. To enforce $\mathbf{h}_i$ to be useful, we impose that it must be of lower dimension than $\mathbf{x}_i$;
2. The fact that the dimension of $\mathbf{h}_i$ is lower than that of $\mathbf{x}_i$ is called a **bottleneck**.

Bottleneck in autoencoders

1. To enforce $\mathbf{h}_i$ to be useful, we impose that it must be of lower dimension than $\mathbf{x}_i$;
2. The fact that the dimension of $\mathbf{h}_i$ is lower than that of $\mathbf{x}_i$ is called a **bottleneck.**

Example of bottleneck

A diagram of a bottleneck with Feed Forward Neural Networks is the following

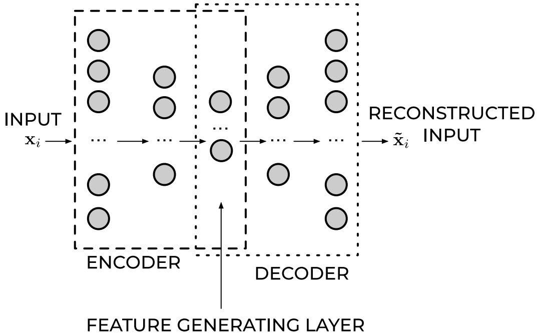

Figure: A typical architecture of a Feed-Forward Autoencoder. The number of neurons in the layers at first goes down as we move through the network until it reaches the middle and then starts to grow again until the last layer has the same number of neurons as the input dimensions.

Bottleneck II

Information

The encoder can reduce the number of dimensions of the input observation (n) and create a learned representation ($\mathbf{h}_i$) of the input that has a smaller dimension $q < n$. This learned representation is enough for the decoder to reconstruct the input accurately (if the autoencoder training was successful as intended).

Activation Function of the Output Layer I

The most used ones are ReLU and sigmoid.

$$\mathrm{ReLU}(x) = \max(0, x) \tag{1}$$

and

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

Warning

ReLU activation function for the output layer is well suited for cases when the input observations $\mathbf{x}_i$ assume a wide range of positive real values.

Warning

sigmoid activation function for the output layer is well suited for cases when the input observations $\mathbf{x}_i$ assume a a range of values in $[0, 1]$

Loss Functions

For autoencoders one can use both typical loss functions:

1. MSE
2. Cross-entropy

Warning

An essential prerequisite for using the binary cross-entropy loss function is that the inputs must be normalized between 0 and 1 and that the activation function for the last layer must be a sigmoid or softmax function.

Reconstruction Error

The typical reconstruction error (RE) is a metric that gives an indication of how good (or bad) the autoencoder was able to reconstruct the input observation.

$$\mathrm{RE} = \mathrm{MSE} = \frac{1}{M}\sum_{i=1}^{M} |\mathbf{x}_i - \tilde{\mathbf{x}}_i|^2 \tag{3}$$

Reconstruction Error - an example

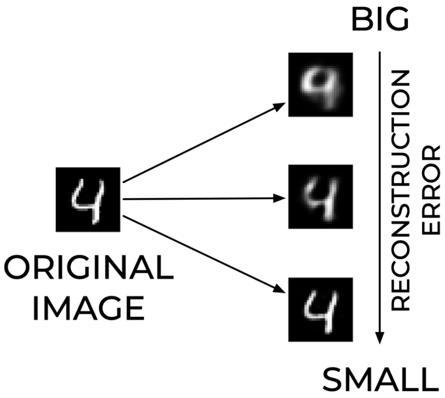

Figure: An example of big and small reconstruction error when an autoencoder tries to reconstruct an image.

Classification with latent features

An example

Dataset: MNIST (handwritten digits, 28x28 gray levels, 60000 training images, 10000 test images)

Model: With kNN with $k = 7$ on MNIST (60000 images, $\mathbf{x}_i \in \mathbb{R}^{784}$) takes ca. 16.6 minutes (ca. 1000 sec.) with an accuracy of 96.4%.

Model with latent features: if we consider an FFA with 8 neurons in the middle layer and again train a kNN algorithm on the latent features $g(\mathbf{x}_i) \in \mathbb{R}^8$ we get an accuracy of 89% in 1.1 sec

An example

Dataset: MNIST (handwritten digits, 28x28 gray levels, 60000 training images, 10000 test images)

Model: With kNN with $k = 7$ on MNIST (60000 images, $\mathbf{x}_i \in \mathbb{R}^{784}$) takes ca. 16.6 minutes (ca. 1000 sec.) with an accuracy of 96.4%.

Model with latent features: if we consider an FFA with 8 neurons in the middle layer and again train a kNN algorithm on the latent features $g(\mathbf{x}_i) \in \mathbb{R}^8$ we get an accuracy of 89% in 1.1 sec

An example

Dataset: MNIST (handwritten digits, 28x28 gray levels, 60000 training images, 10000 test images)

Model: With kNN with $k = 7$ on MNIST (60000 images, $\mathbf{x}_i \in \mathbb{R}^{784}$) takes ca. 16.6 minutes (ca. 1000 sec.) with an accuracy of 96.4%.

Model with latent features: if we consider an FFA with 8 neurons in the middle layer and again train a kNN algorithm on the latent features $g(\mathbf{x}_i) \in \mathbb{R}^8$ we get an accuracy of 89% in 1.1 sec

An example

Input Data	Accuracy	Running Time
Original data $\mathbf{x}_i \in \mathbb{R}^{784}$	96.4%	1000 sec. $\approx$ 16.6 min.
Latent Features $\mathrm{g}(\mathbf{x}_i) \in \mathbb{R}^8$	89%	1.1 sec.

Table: the different in accuracy and running time when applying the kNN algorithm to the original 784 features or the 8 latent features for the MNIST dataset.

An example

Input Data	Accuracy	Running Time
Original data$\mathbf{x}_i \in \mathbb{R}^{784}$	85.4$\%$	1040 sec. $\approx$ 16.6 min.
Latent Features $enc(\mathbf{x}_i) \in \mathbb{R}^8$	79.9$\%$	1.2 sec.
Latent Features $enc(\mathbf{x}_i) \in \mathbb{R}^{16}$	83.6$\%$	3.0 sec.

Table: the difference in accuracy and running time when applying the kNN algorithm to the original 784 features with a FFA with 8 neurons and with a FFA with 16 neurons for the Fashion MNIST dataset.

Data Compression

Data Compression

You can do also data compression by using instead of the original dataset the latent features (16 neurons).

```
np.save('temp_orig', mnist_x_test)
! ls -al temp_orig*
-rw-r--r--  1 umberto  staff  31360128 May  5 13:45 temp_orig.npy
```

and with the latent features

```
np.save('temp_encoded', encoded_imgs)
! ls -al temp_encoded*
-rw-r--r--  1 umberto  staff  320128 May  5 13:45 temp_encoded.npy
```

Anomaly Detection

Anomaly Detection

Lucerne University of Applied Sciences and Arts
HOCHSCHULE LUZERN

1. We consider n autoencoder with only three layers with 784 neurons in the first, 64 in the latent feature generation layer, and again 784 neurons in the output layer;
2. We will train it with the MNIST dataset and in particular with the 60000 training portion of it;
3. Let us choose an image of a shoe from this dataset and add it to the testing portion of the MNIST dataset (that now will have 10001 images).

Anomaly Detection

1. We consider n autoencoder with only three layers with 784 neurons in the first, 64 in the latent feature generation layer, and again 784 neurons in the output layer;
2. We will train it with the MNIST dataset and in particular with the 60000 training portion of it;
3. Let us choose an image of a shoe from this dataset and add it to the testing portion of the MNIST dataset (that now will have 10001 images).

Anomaly Detection

1. We consider n autoencoder with only three layers with 784 neurons in the first, 64 in the latent feature generation layer, and again 784 neurons in the output layer;
2. We will train it with the MNIST dataset and in particular with the 60000 training portion of it;
3. Let us choose an image of a shoe from this dataset and add it to the testing portion of the MNIST dataset (that now will have 10001 images).

Anomaly Detection - how the autoencoder reconstruct the shoe

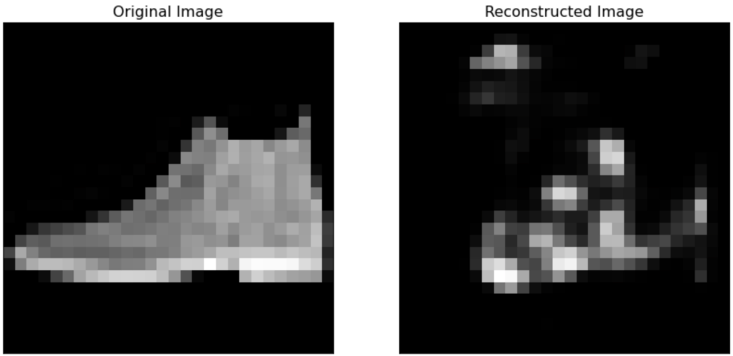

Figure: The shoe and the autoencoder's reconstruction trained on the 60000 hand-written images of the MNIST dataset. This image has the biggest RE in the entire 10001 test dataset we built with a value of 0.062.

Anomaly Detection - Algorithm

1. One train an autoencoder on the entire dataset (or if possible, on a portion of the dataset known not to have any outlier)
2. For each observation (or input) of the portion of the dataset known to have the wanted outliers one calculates the RE
3. One sorts the observations by the RE.
4. One classifies the observations with the highest RE as outliers. Note that how many observations are outliers will depend on the problem at hand and require an analysis of the results and usually lot of knowledge of the data and the problem

Anomaly Detection - Algorithm

1. One train an autoencoder on the entire dataset (or if possible, on a portion of the dataset known not to have any outlier)
2. For each observation (or input) of the portion of the dataset known to have the wanted outliers one calculates the RE
3. One sorts the observations by the RE.
4. One classifies the observations with the highest RE as outliers. Note that how many observations are outliers will depend on the problem at hand and require an analysis of the results and usually lot of knowledge of the data and the problem

Anomaly Detection - Algorithm

1. One train an autoencoder on the entire dataset (or if possible, on a portion of the dataset known not to have any outlier)
2. For each observation (or input) of the portion of the dataset known to have the wanted outliers one calculates the RE
3. One sorts the observations by the RE.
4. One classifies the observations with the highest RE as outliers. Note that how many observations are outliers will depend on the problem at hand and require an analysis of the results and usually lot of knowledge of the data and the problem

Anomaly Detection - Algorithm

1. One train an autoencoder on the entire dataset (or if possible, on a portion of the dataset known not to have any outlier)
2. For each observation (or input) of the portion of the dataset known to have the wanted outliers one calculates the RE
3. One sorts the observations by the RE.
4. One classifies the observations with the highest RE as outliers. Note that how many observations are outliers will depend on the problem at hand and require an analysis of the results and usually lot of knowledge of the data and the problem

Assumption in anomaly detection

Warning

If one train the autoencoder on the entire dataset at disposal, there is an essential assumption: the outliers are a negligible part of the dataset and their presence will not influence (or will influence in an insignificant way) how the autoencoder learns to reconstruct the observations.

References I

- U. Michelucci, “An introduction to autoencoders,” *arXiv preprint arXiv:2201.03898*, 2022.
- D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation, parallel distributed processing, vol. 1,” *Foundations. MIT Press, Cambridge*, 1986.
- D. Bank, N. Koenigstein, and R. Giryes, “Autoencoders,” *arXiv preprint arXiv:2003.05991*, 2020.