An analysis on the use of autoencoders for representation learning

V.

Source

Charte, D., Charte, F., del Jesus, M. J., & Herrera, F. (2020). An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges. *Neurocomputing, 404*, 93-107.

Abstract

In many machine learning tasks, learning a good representation of the data can be the key to building a well-performant solution. This is because most learning algorithms operate with the features in order to find models for the data. For instance, classification performance can improve if the data is mapped to a space where classes are easily separated, and regression can be facilitated by finding a manifold of data in the feature space. As a general rule, features are transformed by means of statistical methods such as principal component analysis, or manifold learning techniques such as Isomap or locally linear embedding. From a plethora of representation learning methods, one of the most versatile tools is the autoencoder. In this paper we aim to demonstrate how to influence its learned representations to achieve the desired learning behavior. To this end, we present a series of learning tasks: data embedding for visualization, image denoising, semantic hashing, detection of abnormal behaviors and instance generation. We model them from the representation learning perspective, following the state of the art methodologies in each field. A solution is proposed for each task employing autoencoders as the only learning method. The theoretical developments are put into practice using a selection of datasets for the different problems and implementing each solution, followed by a discussion of the results in each case study. Additionally, other six learning applications are shortly discussed. This helps conclude that, thanks to alterations in their structure as well as their objective function, autoencoders may be the core of a possible solution to many problems which can be modeled as a transformation of the feature space.

Keywords

representation learning - autoencoders - deep learning - feature extraction

V.1. Introduction

Creating new representations of data is a fundamental task in most machine learning tasks. The success of a classifier, a regressor or other models will greatly depend on the quality of the features it can learn from [1]. When the training set contains intact data from its collection or measurements, it may not be ready for treatment yet. Instead, it is common for data to be expressed with redundant or uninformative variables and for it to include some level of noise. These and other obstacles presented by the data [2] are the reason why most of the manual work of building machine learning models is spent in the preprocessing stage [3]. Representation learning methods [4] approach this problem by adding strategies and techniques which automatically transform the original features into a new set of variables which may be more useful. These can serve as an alternative or a complement to the more tedious preprocessing work.

However, computing alternative representations for data is not only a tool that can help build classification and regression models, but it may be an end in itself in many applications. For example, finding compact binary codes that represent text documents [5], compressing signals to a lower resolution without losing information [6], transforming the problem domain to a different one [7], or producing filtered versions of images with less distortions [8].

Learning representations usually consists in feature engineering [1] or feature extraction [9], depending on whether new features are computed manually by human intervention (either by selection [10] or simple arithmetic operations) or they are generated, evaluated and selected by the machine. Feature engineering leverages expert knowledge and human creativity in order to select features and operate with them in a way that results in a new feature set which seems appropriate for predictors to work with. Nowadays there exist many automatic approaches to feature learning, which relieve users from the tedious task of engineering new features [4]. These methods range from probabilistic to topological and from shallow to deep: principal component analysis [11], Isomap [12], locally linear embedding [13] and Laplacian eigenmaps [14], among others.

With the introduction of deep neural networks, the representation learning stage became integrated within the predictors themselves [15]. These techniques iteratively optimize the classification performance by modifying the weights in several layers of individual neurons which compute a hierarchy of abstractions over the original data. For this purpose, the backpropagation algorithm [16] allows to efficiently accumulate gradients along the network, so

that an optimizer such as Stochastic Gradient Descent [17] or one of its derivatives [18–21] may compute each weight update. Since neural networks can be structured as needed for each kind of problem, they are able to function as standalone feature learners as well. This is the case of autoencoders (AEs) [22], neural architectures whose objective is to find the best representation for the data according to the criterium defined by their loss function.

The objective of this paper is to analyze how AEs can serve as the main basis for solving a wide variety of learning tasks and demonstrate this with concrete applications and experimental results. Throughout the paper, we examine several case studies that expose the adaptability of AEs to these problems.

- First, an example of data embedding onto a very low dimensional space for visualization and exploratory analysis.
- Then, a case where noisy signals are to be repaired by the model.
- Later, a different example where very high dimensional sparse data, such as text documents, is to be compressed onto compact binary codes in a semantic way.
- Additionally, we study anomaly detection, the situation where abnormal patterns are to be detected in sequences but no anomalies are available to learn from.
- As a last case study, we propose the generation of new instances which do not belong to the training set.

Other applications are also briefly discussed: image superresolution, image compression, transfer learning, human pose recovery and recommender systems.

As a starting point, we provide the reader with the necessary background knowledge about the field of representation learning, as well as a summary of the main features of AEs that make them a good candidate model to solve the different problems later approached. The solutions to these tasks using AEs as the only automatic learner highlight their potential and flexibility as feature extraction techniques.

The rest of this paper is structured as follows. Section V.2 describes the background of the problems and techniques above introduced. Section V.3 details the inner workings of AEs. Section V.4 further develops on several case studies where AEs resolve feature learning tasks, and Section V.5 outlines other existing learning applications. Lastly, Section V.6 concludes the text.

V.2. Background

This section motivates the study of representation learning and explains some well known methods that can extract features from data. Afterwards, it introduces deep learning techniques.

Motivation

The features that are used as input conform one of the most important factors when building machine learning models, since learners operate directly with them in different ways: for instance, decision trees, regardless of whether the task is classification or regression, attempt to find the most informative variables to branch at each step [23]; support vector machines calculate the hyperplane that best separates classes in a feature space originating from specific transformations of the original [23], and k-means clustering computes distances among pairs of instances and thus depends strongly on the input domain [24].

Thus, it is of vital importance that the features provided to these learners are useful and as independent as possible. For this purpose, feature engineering techniques [1] can help select, transform and combine variables. Nevertheless, there may be situations where expert knowledge about which features are more relevant, or which operations should enrich the information provided by them, is scarce or not available. In these cases feature engineering may be a trial and error process since practitioners would be operating blindly with the variables. Instead, one can resort to feature extraction methods, which will automatically compute a set of new features according to some criteria.

Classical feature learning methods

Traditionally, feature extraction methods have been developed with linear as well as nonlinear transformations of the variables [9]. They can be considered nonconvex or convex, according to whether the objective function presents local optima or not, respectively [25]. Many of these techniques perform unsupervised learning, but others are supervised [26–28] or even semi-supervised [29]. Next, a summary of typical feature learning methods is provided.

Linear methods The most common linear feature extraction methods are the following. Principal component analysis (PCA) consists in extracting successive variables or *principal components*

with maximum variance while being uncorrelated with the previous components. It is a statistical technique developed geometrically by Pearson [30] and algebraically by Hotelling [31], but an analytical derivation can be found in [11]. Factor analysis [32] is a similar procedure to PCA which considers a set of latent variables or *factors* that are not observed but are linearly combined to produce the final variables. Linear discriminant analysis [26] is a supervised statistical technique which attempts to find linear combinations of features to project samples onto new coordinates that best discriminate classes, albeit making some assumptions about the distribution of the data.

Nonlinear methods Some well known nonlinear approaches to feature extraction are kernel PCA, restricted Boltzmann machines and manifold learning methods. Kernel PCA [33] extends PCA to nonlinear combinations of features by projecting samples onto higher-dimensional spaces and using the kernel trick [34]. Restricted Boltzmann machines are undirected graphical probabilistic models, also known as harmoniums [35], with one visible layer and one hidden layer that acts as the set of extracted features. They can be trained using the contrastive divergence algorithm [36]. Many nonlinear feature learning methods attempt to find coordinates for a lower dimensional structure embedded in the original features, namely, a manifold. Multidimensional scaling (MDS) is one of the first techniques that can be considered manifold learning, as its objective is projecting samples in a low-dimensional space while translating as much information of pairwise distances as possible. There are several variants of MDS, one of them is Sammon mapping [37], which improves on MDS by using a different cost function which stresses large distances similarly to small ones. Isomap [12] is a more recent extension of MDS which looks for the coordinates that describe the actual degrees of freedom of the data while preserving distances among neighbors and geodesic distances (the length of the shortest path that connects two points in the manifold). Locally Linear Embedding [13] also seeks a manifold which preserves neighbors but, in order to maintain the local structure, it linearly reconstructs each point from its neighbors. Laplacian eigenmaps [14] is a procedure that builds a graph based on the neighborhood structure of the data, and from it a weight matrix whose eigenvectors can be used to compute new coordinates for each point.

Deep representation learning

Deep learning architectures are hierarchies of abstractions of the input feature space and, as such, they compute several transfor-

mations of the features before reaching a response. In some cases, these can be seen as learned representations, since they must be able to capture the relevant information from each instance in order to output an accurate result. This effect can be observed especially in convolutional neural network classifiers, which are usually split into a feature extraction component formed by convolutional layers and a decision module composed by fully connected layers [38]. Apart from neural networks with other objectives such as supervised classification or regression, there have been different approaches to shallow as well as deep neural structures for unsupervised feature learning [39], such as self-organizing Kohonen maps [40, 41], predictability minimization [42], restricted Boltzmann machines [43] and AEs [44, 45]. There have been many instances of these unsupervised techniques being used to either pre-train or provide feature transformations for supervised models [46].

AEs are probably the most versatile unsupervised neural network models. They essentially combine some kind of bottleneck or restriction in the learned data representations with the objective of reconstructing and repairing the original input from that representation [22]. There are several ways to impose restrictions that produce interesting representations, and the reconstruction objective will cause the network to retain all invariant feature information along its weights, so that the representation or encoding holds mainly instance-specific traits. For example, undercomplete AEs project inputs into lower-dimensional encodings, sparse AEs obtain representations with very few activated neurons, and denoising AEs attempt to repair partially corrupted data.

Their versatility is demonstrated by the amount of applications AEs have and their diversity. Across the rest of this work, we focus on certain applications of representation learning that are solved with AEs and we analyze how each model is built and trained.

V.3. Autoencoder fundamentals

AEs are neural network structures designed with the purpose of learning new features. Throughout the following subsections, their main characteristics and differentiating aspects are outlined, and some ways to influence the encoded variables are discussed.

Origin and essentials of autoencoders

AEs were originally conceieved as a way of initializing neural networks [47] and continued fulfilling that purpose for some

time, serving as a starting point for training of deep networks as well [48]. Over the last years, other applications for AEs have been emerging and at the same time other approaches to neural network training and regularization have succeeded over AEs [49, 50]. As a consequence, the common uses for AEs have shifted from helping train other neural networks to other applications of their own.

In general, the training process required to learn an AE can be unsupervised, that is, it does not need labels or class information in order to generate a model for the data. Instead, it extracts useful information from each instance by feeding its feature vector through some transformations which impose a bottleneck or restriction on the possible representations it can compute. Then, the representation is mapped to the original feature space through a similar set of transformations, and the AE is evaluated according to the fidelity of the reconstruction. This feedback allows to modify the parameters iteratively until convergence is reached.

AEs take the form of a neural network with at least one hidden layer and two components, an encoder and a decoder, which are connected by the coding layer [22]. These components are usually symmetric in layer shapes to each other, especially if they are implemented as fully connected neural networks. In certain occasions, even the weights of each layer in the decoder are tied to the corresponding layer in the encoder. In general terms, however, it suffices with the input layer of the encoding and the output layer having the same shape. Fig. V.1 shows how the architecture of an AE may look like.

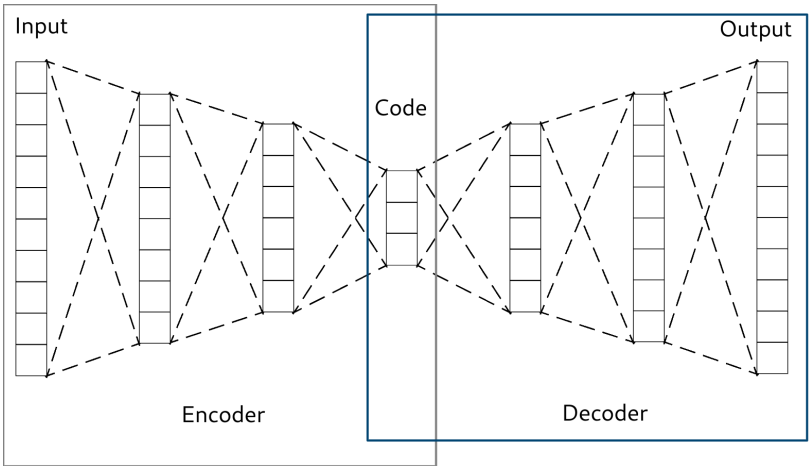

Figure V.1.: Illustration of the general structure of a basic AE: an encoder and a decoder connected by the encoding layer

In summary, an AE can be seen as the composition of an encoding map f which projects inputs onto a different feature space, and a decoding map g which operates inversely. The main objective of the AE is to recover as much information as possible of the original input, so it will attempt to minimize a distance between the inputs and the outputs:

$$\min_{\theta} \sum_{x} d(x, g_\theta(f_\theta(x))) \tag{V.1}$$

The distance function d used in the loss function is usually either the mean squared error, see Eq. (V.2), or the cross entropy, shown in Eq. (V.3). In the first case, data may not be normalized and the output units should use an unbounded activation function. For a cross entropy loss, each input and output variable is modeled as following a Bernoulli distribution, so data should be scaled to the $[0, 1]$ interval and output units could make use of a sigmoid activation.

The mean squared error for an input x and output x' of length n is defined as:

$$d(x, x') = \frac{1}{n} \sum_{i=1}^{n} (x_i - x'_i)^2 \tag{V.2}$$

Similarly, the binary cross entropy for the same input and output is computed as:

$$d(x, x') = -(x \bullet \log(x') + (1 - x) \bullet \log(1 - x')), \tag{V.3}$$

where $\bullet$ denotes element-wise product and all other operations are also performed element-wise.

Modeling the coding layer

The main objective of the AE (Eq. V.1) only promotes faithful reconstructions without explicitly considering any aspect about the codes used. This can be enough in many cases where the codes are low dimensional and they can capture only the relevant information of the instances just by training to reconstruct accurately. Notwithstanding, there are situations that require considering a more general case of the objective, which allows penalizing certain behaviors of the encoding found by the network, or even the values of the parameters themselves (Eq. V.4).

$$\min_{\theta} \sum_{x \in \mathcal{X}} d(x, g_\theta(f_\theta(x))) + r_1(f_\theta(\mathcal{X})) + r_2(\theta) \tag{V.4}$$

A straightforward example of this kind of restrictions is the sparse AE [51, 52], which adds a penalty for high activation rates in the

neurons of the code layer (Eqs. V.5 and V.6):

$$r(\mathfrak{Z}) = \sum_{j=1}^{k} (\rho - \rho_j)^2, \text{ or} \tag{V.5}$$

$$r(\mathfrak{Z}) = \sum_{j=1}^{k} \rho \log \frac{\rho}{\rho_j} + (1-\rho) \log \frac{1-\rho}{1-\rho_j}, \tag{V.6}$$

where $\rho_j = \frac{1}{|\mathfrak{Z}|} \sum_{z \in \mathfrak{Z}} z$ is the average activation vector, k is the length of the code and ρ is the desired activation rate.

Other, more sophisticated variations on the AE with different penalties are the contractive AE [53, 54], which promotes finding and preserving any local structure from the original feature space, and the variational AE [55], which uses a penalty to impose a distribution to the codes computed by the encoder.

Penalties on the codes are not, however, the only way of incentivizing a behavior on the encoder mapping. Denoising AEs [56, 57] establish a slightly different criterion to evaluate the performance of the reconstruction: the network must be able to repair any noise or corruption from the input. Robust AEs [58] use another objective function, correntropy [59], which has a similar effect in repairing several kinds of noise from the input data.

Beyond unsupervised autoencoders

Although the objective of an AE usually does not involve direct prediction of labels, it can sometimes learn from classified examples. The most straightforward way to introduce class information into the AE is to modify the loss function so it propagates different errors according to the class of each instance. For example, we could weight each class differently. Assuming the classes are binary and $\alpha \in [0, 1]$, the objective in Eq. (V.7)

$$\min_{\theta} (1-\alpha) \sum_{(x,0)} d(x, g_\theta(f_\theta(x))) + \alpha \sum_{(x,1)} d(x, g_\theta(f_\theta(x))) \tag{V.7}$$

would give more importance to reconstructing one of the classes, which may help if the aim is to find a manifold for that class and the other one is less relevant.

Several uses of label information can be found in the proposal of the adversarial AE [28]. This AE has a similar behavior to the variational AE in that it also forces the codes to follow a given distribution. Instead of using just a loss penalty, it adds a generator which samples the distribution, and a discriminator which attempts to distinguish distribution samples from codes belonging to actual instances, analogous to a generative adversarial network [**gan**].

The label information can be used then to locate each label in a region of the distribution, by feeding labels as well as codes to the discriminator. Alternatively, labels can be feeded to the decoder, which causes the codes to discard label information and instead model style in the data.

Another step forward in introducing label information in AEs would be for them to be able to predict labels as well. Some work has been already done along these lines, by training an encoder and decoder simultaneously to reconstruct and to produce codes as similar as possible to the labels in a one-hot format [60].

V.4. Learning task case studies

The following subsections detail several real examples of application of AEs: embedding data onto a very low-dimensional space for visualization purposes, reducing the noise in images, computing semantic hashes for large text documents, finding anomalous behaviors in sequences and generating new instances outside the training set. For each application, a relevant dataset has been selected and a model has been specifically designed to solve the problem. The basic traits of all chosen datasets can be found in Table V.1.

Dataset	Application	Input features	Training examples	Test examples
CPU Activity	Visualization	21	6553	1639
Satellite image	Visualization	36	5142	1288
STL10 [61]	Noise reduction	$96 \times 96 \times 3$	5000	8000
Bibtex[62]	Semantic hashing	1836	5916	1479
UNSW-NB15[63]	Anomaly detection	187	37000	175341
AT&T faces	Instance generation	64×64	400	-

Table V.1.: Main traits of datasets used for the experiments

The models described below are each associated to a diagram describing the layer structure of the corresponding AE and the purpose of each layer. Please refer to Fig. V.2 for an example of how each model is detailed.

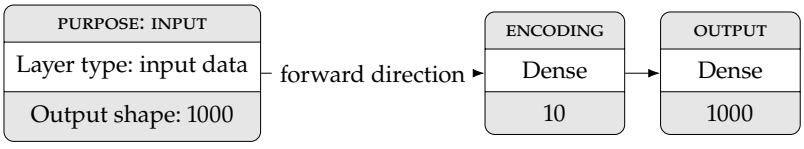

Figure V.2.: Example AE architecture. Each block represents a layer and is splitted into three parts: the meaning or purpose of the layer, the type of operation performed and its output shape (size of each dimension).

All examples have been implemented and executed employing the following setup: Tensorflow [64] 1.14.0 and Keras [65] 2.2.4 on top of Python 3.7 and R 3.6, running on an Intel Core i5-8400 CPU and a NVIDIA GeForce RTX 2060 GPU. The associated

software can be found at the following GitHub repository: https://github.com/ari-dasci/autoencoder-case-studies/.

Data visualization

Most of the data collected nowadays, either from industries or from the web, is high-dimensional. Visualization techniques can help its interpretability, but the data generally needs to be summarized for this purpose. Traditionally, an alternative representation would be a subset of its features or its principal components [66]. An AE, however, is able to automatically compute a representation that fits each dataset. This representation can be 2 or 3-dimensional if the AE is configured conveniently [67], or if another embedding technique (such as t-SNE [68]) is used after a higher-dimensional encoding.

In particular, if our dataset consists of instances (x, y) where x is a feature vector and y is its associated label, we can use a training subset to learn an autoencoder model with an encoding $f : \mathbb{R}^n \to \mathbb{R}^2$ resulting of the composition of the hidden layers up to the code layer. Then, encoded examples can be colored in a scatter plot according to their class.

Although a simple AE could fulfill the embedding task, it can be convenient to restrict or modify its behavior so as to influence the projection to the embedding space, in a way that improves how the populated regions in the original space are modeled. Along these lines, there are several approaches: denoising criteria [56], contractive regularizations [53] and embedding regularizations [67].

Denoising criterion A denoising AE [56] trains, as briefly explained in Section V.3, by reconstructing partially corrupted inputs. In order to do this, a corruption or noise function introduces alterations on the input data: for example, a Gaussian noise $\xi \sim N(0, \sigma)$ would be used to produce the input $\tilde{x} = x + \xi$. The reconstruction error is now computed as $\sum_x d(x, g(f(\tilde{x})))$. During the training process, the AE is forced to distinguish useful information from mere perturbations of the data. If the instances lie on a manifold in the original feature space, this can effectively train the AE to "push back" instances to the manifold by discarding small displacements from it. This can remove noise in the inputs as well as reconstruct some missing values if inputs are just an estimation [8, 69]. As a result, the encoding can serve as a set of coordinates for the manifold.

Contractive regularization The contractive AE [53] uses an additional penalty in the training objective which promotes local invariance to displacements in many directions around the training samples, i.e., it is less sensitive to small perturbations especially in directions that lead outside the manifold. The penalty consists in the squared Frobenius norm of the Jacobian matrix of the encoder, that is, the sum of the squares of all first-order partial derivatives applied to all inputs: $\sum_x \|J_f(x)\|^2$. This can be seen as a generalization of L2 weight decay to the case where the encoder is nonlinear. This regularization favors encodings where all dimensions are contracted, but the reconstruction error prevents the AE from contracting dimensions along the manifold.

Embedding regularization An alternative objective function for AEs can be the same loss function from other embedding techniques. This is the idea behind embeddings with AE regularization [67], which combines the reconstruction error with one of several possible embedding loss functions coming from Laplacian eigenmaps [14], multidimensional scaling [70] and margin-based embedding [71]. These loss functions evaluate the embedding by taking pairs of instances, and the AE is adapted the same way, by computing the embedding loss across all pairs of instances and the reconstruction loss across all instances.

For the purposes of demonstrating the capacity of AEs to find manifolds and appropriate embeddings, we have selected a regression dataset, CPU activity*, and a classification dataset, Satellite image†. The AE used to find embeddings is the contractive AE. AEs for both datasets have been designed using the same criteria: three hidden layers, the encoding layer having 2 variables and the rest having as much variables as needed so that the compression ratio from the input to the first hidden layer is the same than from the hidden layer to the encoding layer. The resulting architectures are detailed in Fig. V.3. The AEs have found the projections shown in Fig. V.4, where the label of each instance is used to color each point. Notice that the AEs have trained without the respective target variables, but there appears to be some degree of separability of classes and different values of the regression variable in each graph.

In order to verify to a certain degree that these embeddings, in addition to producing meaningful visualizations, contain the necessary information about the data, the mean squared error between each instance and its reconstruction through the AE can be computed. As a reference for comparison purposes, the same reconstruction error can be computed from the two first principal

* CPU activity dataset is available at https://www.openml.org/d/573.
† Satellite image dataset can be found at https://www.openml.org/d/294.

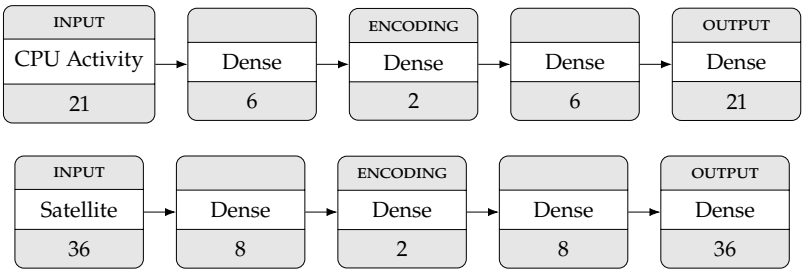

Figure V.3.: AE architectures for visualization

components of the data. Table V.2 holds these results, which are very favourable to the contractive AE, since the error is notably lower in every case. It can be deduced that the contractive penalty in the AE does not hinder the reconstruction objective, while it helps obtain useful low-dimensional embeddings.

	Mean squared error			
	CPU Activity		Satellite	
Method	train	test	train	test
PCA	0.5577	0.5097	0.1475	0.1483
Contractive AE	0.5053	0.4546	0.1132	0.1157

Table V.2.: Mean squared error comparison between the reconstructions of a contractive AE with a 2-variable encoding and the projections to the original feature space from the two principal components of the data. Lower values are better.

Noise reduction

Similar to searching for interesting representations of data in the encodings of an AE, we can look for a reconstruction that adds value to the input data. One way an AE can help with this is to remove noise from its inputs. This is especially useful when dealing with images [8], sound [72] and other kinds of signals [73], since capture methods usually may introduce some noise and it would be desirable to have a clearer and sharper output.

In general, an AE can be trained to be resilient to input perturbations with a mere random additive noise at the input. Throughout the optimization stage, the AE only takes as input partially corrupted versions of the training examples and attempts to reconstruct the original ones. Once trained, this AE does not necessarily expect more noisy data, but instead it will have learned to be robust against small changes in its inputs. This type of AE is usually called a denoising AE, and performs well in many scenarios that do not necessarily involve treatment of noisy data [57].

In this case, nonetheless, the goal is to eliminate potential perturbations in the inputs. Unlike a generic noise reduction filter, which will perform similar operations no matter what data it receives, a denoising AE can be fitted to a specific training set and may thus be more reliable with different kinds of data. More formally, we

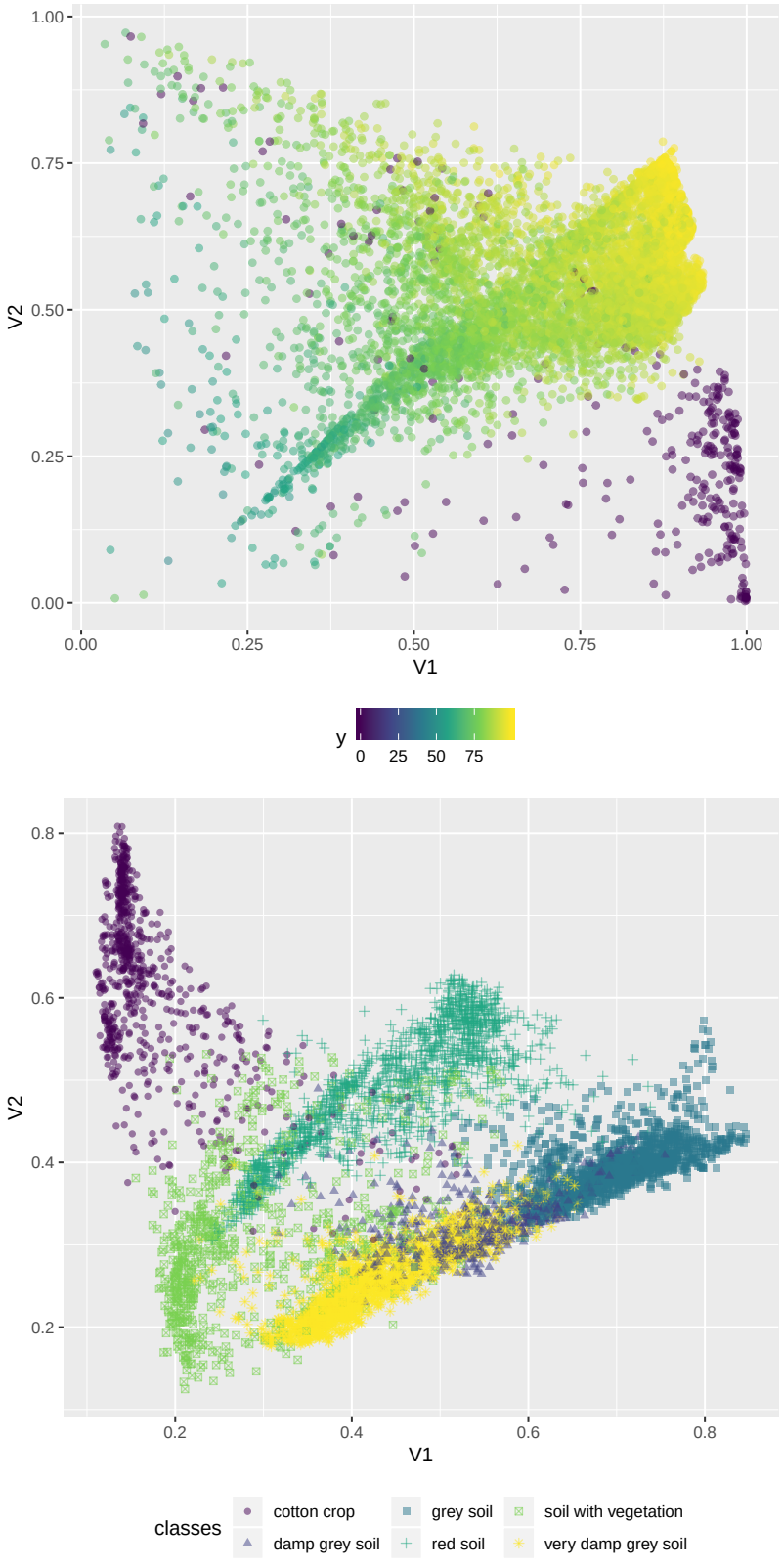

Figure V.4.: Embeddings learned by an unsupervised contractive AE. The top image shows the projection of the CPU Activity dataset where each point has been shaded according to the level of user activity. The bottom image displays the projected samples of the Satellite Image dataset, each one colored according to its class.

consider a noise function v, which generates the corrupted data that the autoencoder trains with to minimize

$$\sum_{x} d(x, g(f(v(x)))) \, .$$

The following are some possible noise functions that may be applied:

- $v(x) = x + \xi$ where ξ is sampled from a Gaussian distribution with small variance
- $v(x) = x + \xi'$ where ξ' is sampled from a Cauchy distribution with small scale
- $v(x) = \begin{cases} 0 & \text{with low probability} \\ x & \text{otherwise} \end{cases}$
- $v(x) = \begin{cases} 0 & \text{with low probability} \\ 1 & \text{with low probability} \\ x & \text{otherwise} \end{cases}$

Notice that the Gaussian and Cauchy distributions will usually induce small changes to most inputs, while the zero and zero-one noises will leave most values intact but the change in the corrupted ones will be more drastic. Thus, for a given application, a specific type of corruption function can be selected so that it fits best to the types of noise the samples could have.

When using denoising AEs, it is also convenient to adapt the type of layers used to the kind of data. For instance, a convolutional AE would be best for noisy images, and an LSTM AE for corrupted signals or sequences. Fig. V.5 details a possible encoder-decoder structure for a denoising AE which uses convolutional layers in the encoding phase as well as deconvolution operations during decodification.

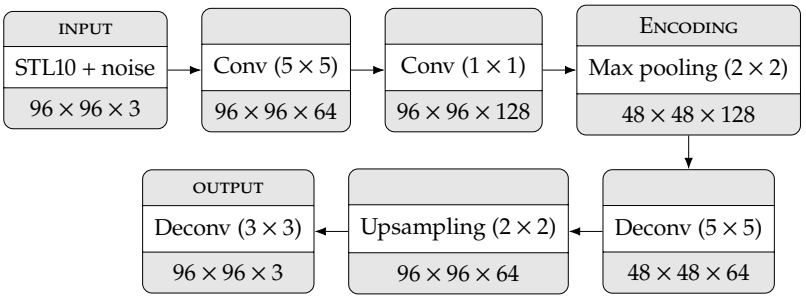

Figure V.5.: Denoising AE architecture for noise reduction

When this AE is trained with data from the STL10 dataset [61], a subset of the ImageNet dataset, the objective function will force it to configure its weights so that input noise is reduced along the network. The noise used in this case has been zeros with a probability of 0.1. The test images measure 96x96 pixels and have also been corrupted with around 10% of noisy values, which can affect any color channel, so each pixel has a 30% likelihood of having any of its 3 values altered. The AE was trained during 10 epochs with the training data using optimizer Adam.

The results can be analyzed in Table V.3, which shows the AE achieves a reduction in the mean squared error of about 89%.

Images	Mean squared error	Noise reduction
Reference	0	100%
Noisy	1656.08 ± 696.31	0%
Predicted	159.74 ± 74.55	88.94% ± 6.38

Table V.3.: Summary of results for noise reduction (average values and standard deviations are provided). Original images without noise are the reference for measuring the mean squared error, and the noise reduction is computed for each image as the percentage decrease in this error. Images are represented by their RGB values from 0 to 255.

Fig. V.6 displays some of the test inputs together with their reconstruction by the network. The resulting reconstructions remove most of the noise and appear slightly softer than the originals.

Figure V.6.: Random selection of test examples (first and third rows) and their reconstructions (second and fourth rows) via forward passes through the denoising AE.

Semantic hashing

Hashing usually refers to the process of summarizing large batches of data in smaller or simpler codes. Hashes are employed in data structures for fast search times, they can be used to find duplicates and to protect data against corruption and manipulation.

This task in particular, semantic hashing [5], involves finding binary codes which form buckets of similar data, i.e. when two data points are similar to each other, there is high probability that they will be assigned the same hash. Furthermore, if two similar data points are not hashed identically, their hashes will likely differ in only a few digits. In consequence, a way of finding instances similar to a query instance is to hash it and look for those whose hashes are the same or almost identical. This is the opposite of cryptographic hashing [74], where the likelihood of two similar entries obtaining the same hash is almost zero and there is no way of retrieving a document from its hash.

The idea of finding semantic relations between data points is especially useful in document searches: if a query document is provided, then the search method should find those documents in the dataset which match as closely as possible. It is also of application

in an image domain, where finding matching binary sequences is much more efficient than comparing two pictures [75].

The approach described in [5] uses a very simple AE architecture, with an added noise generator after the encoding which forces the encoder to polarize its outputs.

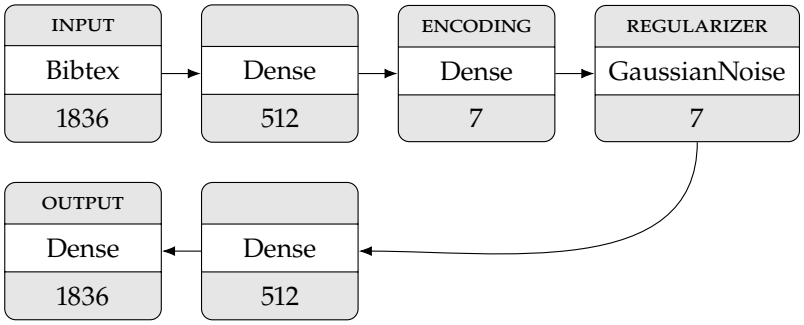

Figure V.7.: AE architecture for semantic hashing

In this case, the Bibtex dataset [62] was selected to illustrate the application. Fig. V.7 shows the AE architecture that was defined for this purpose. The input data provides 1836 binary features which are then projected onto a smaller feature space and lastly onto a 7-dimensional encoding, which is in turn slightly corrupted before decoding. The noise introduced in the encoding during training requires it to take extreme values, for the noise not to affect the reconstruction.

In order to assess whether the trained model serves the purpose of semantic hashing, we can group all possible pairs of hashes according to their Hamming distance (e.g. 0001000 and 001001 are 1 digit away from each other, while 1010101 and 0101010 are separated by a Hamming distance of 7). Then, we measure the intercluster distance between those pairs of hashes, computed as the mean cosine distance from each instance in the first cluster to each one in the second. Assuming the clusters group similar instances, the intercluster distance should increase along with the Hamming distance. The distances for this example are illustrated in Fig. V.8, which indeed shows simultaneous growth of both.

In addition to quantitatively evaluating the quality of the model, it can be qualitatively analyzed in order to verify whether semantic hashing indeed groups topics in similar hashes. One way of doing this is computing the term frequency-inverse document frequency index (tf-idf) [76] of the words for each cluster. This way, words that are frequent within a cluster but uncommon along the rest of the test set are considered the most relevant words. Table V.4 shows a truncated list of hashes used by the AE to cluster documents, along with their most relevant words ranked by tf-idf.

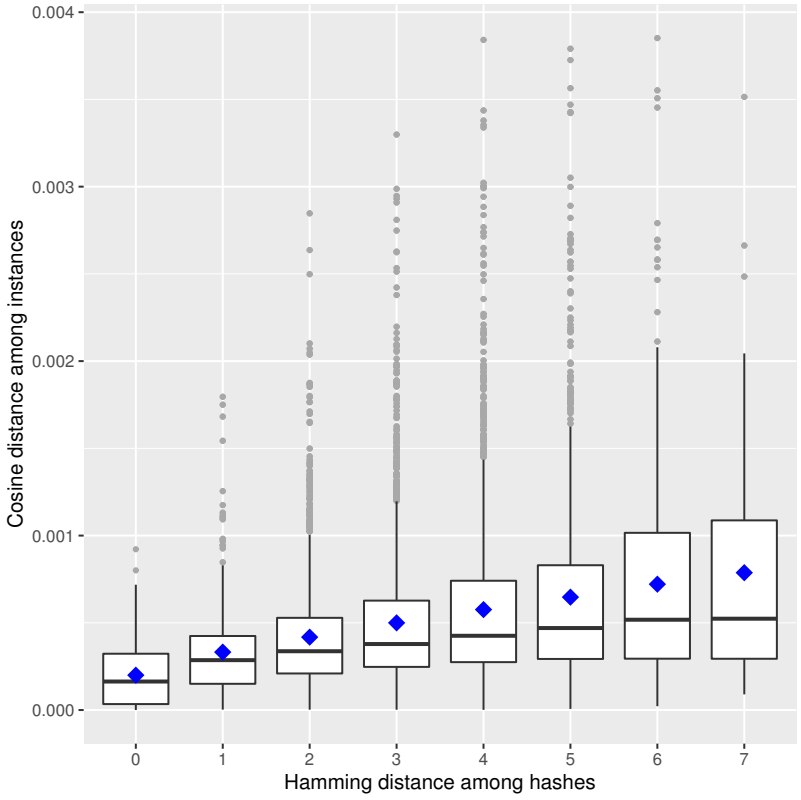

Figure V.8.: Intercluster cosine distance boxplot according to the hamming distance between hashes. Blue diamonds indicate the mean cosine distance among all pairs of clusters that differ in k digits where k is a Hamming distance. Gray dots indicate outlier cosine distances.

Anomaly detection

Sometimes the objective of a machine learning task is to find unusual behaviors or abnormalities in data, for example, detecting a possible security attack by analyzing server logs, or identifying rare patterns in medical checks. This is known as anomaly detection because the cases of interest are few in contrast to the amount of normal instances, and even in some cases there are no anomalies to train with. In this situation, a traditional classifier cannot solve the problem since it will not be able to assign a class it has not seen before.

An approach to anomaly detection without previously observed anomalous cases is to model those considered typical, and mark as anomalies those instances which do not fit the model. An AE can be used for this purpose, since it can be trained to accurately encode and reconstruct instances following a certain distribution. When the AE is feeded new instances, it is assumed that reconstruction of anomalous data will not be as accurate, since it should follow a different distribution [77–79]. More formally, the hypothesis of this methodology is that, when trained with normal data, $d(x, g(f(x)))$ will be very small when x is normal and very high when x is anomalous.

An useful application of anomaly detection where real world data will generally lack anomalies is network intrusion [80, 81], that is,

Hash	Relevant words
0000001	thermodynamic, transitions, induced, generalized, completely, interacting
0000011	relaxation, barrier, mainly, contribute, surfaces, rights
0000010	lipoproteins, capacity, oxidation, apo, receptor, recognized
0000110	identifying, amino, united, capable, matrix, region
0000111	carbon, storage, enzymes, assessed, notes, roles
0000101	infrastructure, configuration, challenge, location, qualitative, improvement
0000100	innovation, construction, ontologies, communities, 1999, located
0001100	mining, advances, bioinformatics, er, solved, intelligence
0001101	reuse, object, perspectives, intelligent, notes, logic
0001111	trans, reading, behavioral, cultural, 1997, gap
0001110	ss, siamese, betta, splendens, male, fighting
0001010	siamese, ss, fighting, male, display, fish
0001011	treated, barrier, combines, electrostatic, solvent, molecule
0001001	thermal, boltzmann, origin, bulk, fluctuations, disorder
0001000	numerically, temperatures, exact, magnetic, glass, zero

Table V.4.: The first 15 hashes used as semantic codes for clusters found by the AE, ordered in Gray code. The most relevant words are selected according to tf-idf computed for each cluster. They show some common topics between hashes 0001110 and 0001010, and between 0000001, 0001001 and 0001000.

the detection of potential security attacks and malicious accesses to a server. The straightforward approach is to continuously log server accesses, and extract data from a period of time where usage has been normal. By means of these data, an AE can be trained to recognize typical usage parameters. Then, new log accesses are constantly feeded to the AE in order to predict their reconstruction error. In the case that several successive errors are much higher than the mean, an attack may be underway.

The AE used for this purpose will work as follows: the encoding layer will perform a drastic dimensionality reduction in order for it to model the most essential information from the training data, which does not include any anomaly. This should help have low error rates on normal data, similar to training instances, but very high ones on anomalous data. In general, this may not work well for uncommon, isolated anomalies, but it is useful when anomalies are several in sequence, so this strategy is especially designed for time series data.

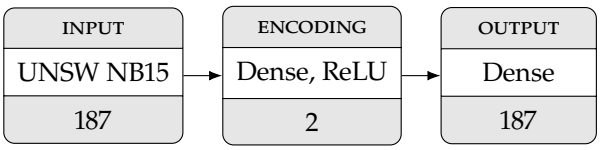

Figure V.9.: Denoising AE architecture for anomaly detection

The dataset treated in this example is UNSW-NB15 [63], which has 3 nominal variables and 42 numerical descriptors. Since AEs cannot work directly with nominal variables, these have been converted into dummy binary variables. In addition, any anomalous data from the training subset has been removed. In total, 37000 instances with 187 features are being introduced as the training input of

the AE, whose architecture is shown in V.9. The extraction of two features is sufficient to model an approximation of most of the normal data, but cannot preserve enough information for the reconstruction of most anomalies.

The results of training this model are summarized in Fig. V.10 and Fig. V.11. The first is a precision-recall curve which gives details about the fraction of detections which are actually anomalies and the ratio of detected anomalies among all of them. We find that it is possible to detect more than half the anomalies without obtaining too many false alarms. Since the test dataset contains many more anomalies than normal instances and the objective is to detect abnormal sections more than to find every individual anomaly, a recall of around 50% could be enough as long as the precision is high so that few false alarms are raised.

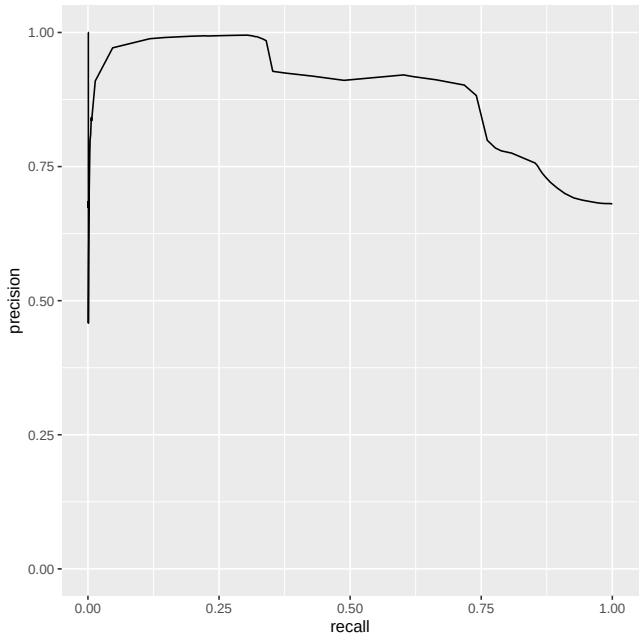

Figure V.10.: Precision-recall curve for the detection of individual anomalies in the UNSW dataset.

Indeed, Fig. V.11 graphs the reconstruction error for each test instance and shows that when an adequate threshold is chosen, anomalous sections can be easily detected with very few isolated false alarms that can be discarded. In this case, the chosen threshold is the mean reconstruction error plus 6 times its standard deviation, but it could be tuned high or low in order to adjust the sensitivity of the detection.

Instance generation

The representation learned by an AE may be useful to encode or reconstruct individual instances from a training set, but in certain cases it will be very convenient to ensure that this representation is actually attempting to perform some kind of manifold learning,

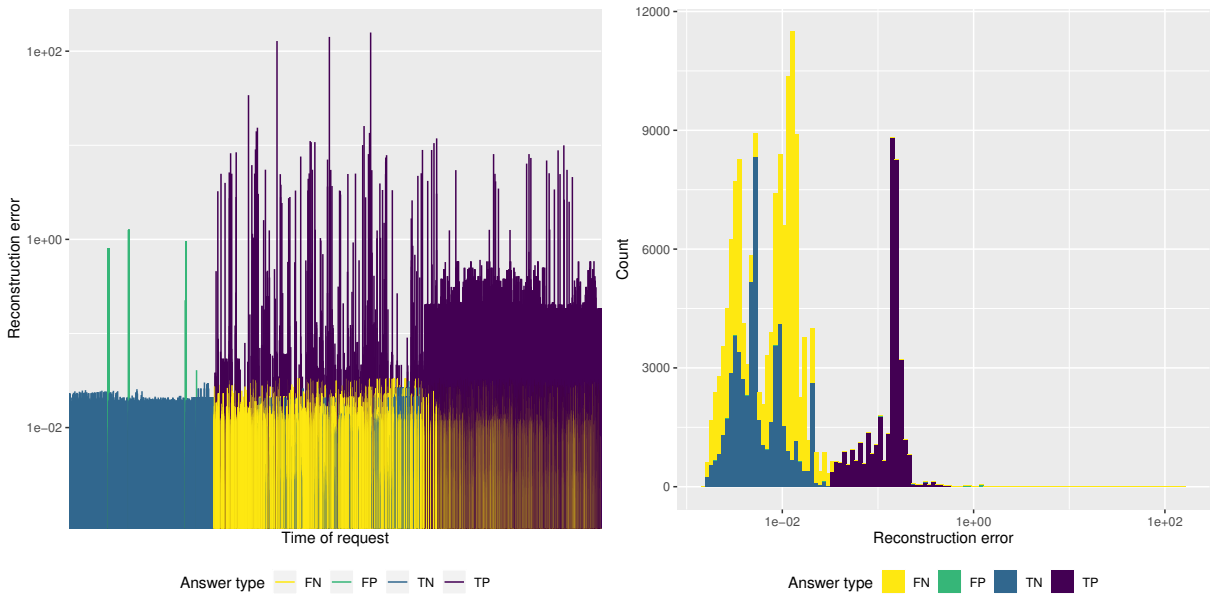

Figure V.11.: Reconstruction error of the AE during test. The graph on the left shows the reconstruction error of each request in sequence, where the detection threshold is set to the mean training error plus 6 times its standard deviation. The histogram on the right shows the amount of hits and misses according to the reconstruction error.

mapping the feature space onto a smaller space in a way that makes sense to work with the whole encoding space. This encoding space would allow to predict a reconstruction for encodings that do not come from an instance in the original feature space, and still produce a coherent result. For instance, an useful application would be to generate new images of faces similar to those in a training set but not identical to any of them. This is usually harder to achieve with simple operations such as interpolation, because they would compute many images that do not represent faces.

There are several variants of AEs that can fulfill this purpose, namely variational [55], adversarial and contractive AEs. Variational as well as adversarial AEs force a prior distribution in the encodings in different ways, which allows to sample new instances by taking points from this space and projecting them onto the original feature space via reconstruction (g). The contractive AE, on the contrary, only imposes a regularization which promotes instances to be mapped to encodings near their neighbors. This helps the autoencoder perform transformations that find manifolds in the data, since local structure is preserved. The manifold can then be traversed in order for the decoder to generate new instances.

Variational AEs are stochastic in the sense that they do not map each instance to a single point in the embedding space, but a distribution instead. This is usually a normal distribution, defined by its mean and standard deviation. Then, a reconstruction is produced by sampling that distribution and propagating the results through the decoder network. The objective function in this AE combines

the clustering behavior of the reconstruction loss function with a regularization loss which forces the distribution to be as similar as possible to, generally, a multivariate unit Gaussian. This helps the AE extract a very compact representation which only preserves the necessary information to provide a reconstruction of the input.

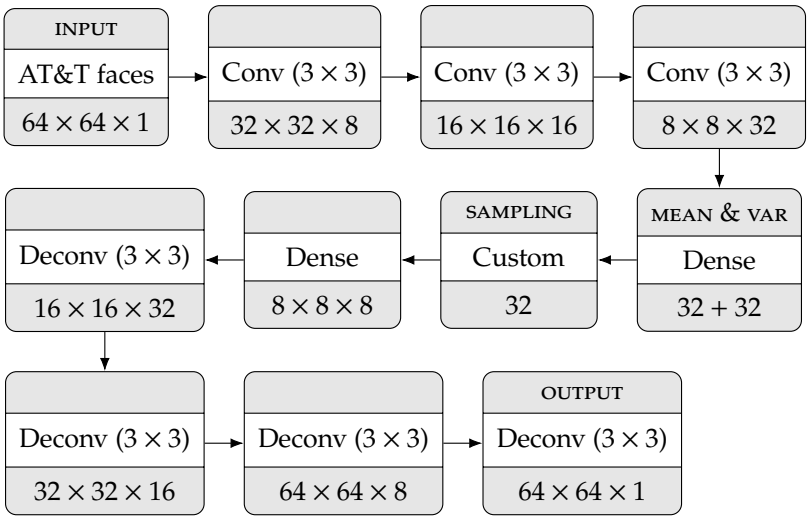

Figure V.12.: Variational AE architecture for instance generation. The sampling layer draws a sample from the vector of normal distributions with means and variances given by the previous layer.

In this example, a variational AE following the structure in Fig. V.12 is trained to generate human faces that do not belong to any person, since they will not be present in the training dataset. The input data used during training belong to the AT&T faces dataset[‡], also known as Olivetti faces dataset. The resulting model can be sampled by feeding arbitrary values to the generator component, which then outputs previously unseen images. Fig. V.13 shows some representative examples of the generated faces using this AE.

[‡] AT&T faces dataset is available at https://www.openml.org/d/41083.

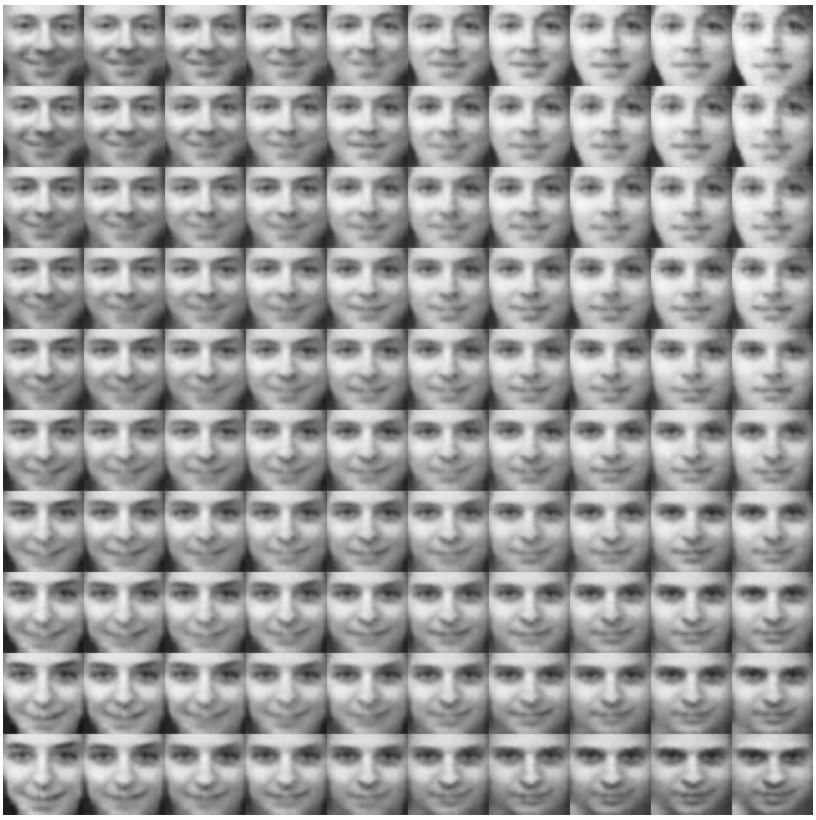

Figure V.13.: Faces sampled from the encoding space of a variational AE, using interpolations between the projections of images in the original dataset

V.5. Other applications

Apart from the previous selection of applications approached with representation learning techniques based on AEs, there are many other situations where AEs can be applied to extract features from data. The following are learning applications present in the literature that fell out of the scope of this article.

Image superresolution

This problem consists in building a high resolution image from a low resolution sample, such as a thumbnail. By using an AE trained with low resolution images and another with the high resolution ones, a map can be trained from the first encoding to the second [82]. This way, the encoder from the first AE can be connected to the decoder from the second AE and the resulting network can be fine-tuned. During prediction it suffices with feeding a low resolution image through the new network, which will encode it and decode it through the high resolution decoder, producing a higher quality image.

Image compression

Images are usually compressed with algorithms designed for this specific purpose, e.g. the JPEG standard [**jpeg**]. Since a compression mechanism must include a component which compresses the image and another which performs decompression, AEs can be trained in different ways to treat this problem as well [6, 83, 84], even surpassing the capacity of JPEG2000 especially at low bit rates.

Transfer learning

In a transfer learning task, the learner must make use of the knowledge extracted from data in a given domain to apply it to a different domain. This may consist in using pre-trained networks with a large dataset to use them with a small dataset by a fine-tuning process. However, when labels for the large dataset are not available, the first stage will necessarily be unsupervised [46], in which case an AE can be trained and its extracted features can initialize a network for a supervised problem with a dataset from other domain.

Human pose recovery

This is an application specific to image and video data where people appear and the aim is to recognize the pose of each person from the visual information, i.e., to generate a skeleton describing the position and orientation of the legs, arms and the rest of the body. One of the challenges is to model this skeleton as a 3D object while images are only 2D. AEs have been used as the core of a human pose recovery model [85] for extracting an inner pose representation which then maps onto a representation of the 3D pose and is decoded as a 3D pose. This process is, in fact, achieved with two AEs, one for each inner representation required, which are then connected through the representation mapping.

3D shape learning

Extracting features from three-dimensional shapes usually has a high computational cost but it is fundamental for tasks such as 3D object retrieval and matching. There are several AE-based models for automatic feature extraction that can help model this type of data [86–88]. These range from simple stacked AEs to combinations of convolutional AEs and extreme learning machines. In general, retrieving similar objects to a given input consists in encoding the

input and comparing the result to the codes of known objects in order to find the nearest or most similar ones.

Recommender systems and tagging systems

Recommender systems are filters that seek to predict user preferences for products, taking into account previous choices or ratings. Collaborative filters for recommendation combine the information of different users to build predictions. In [89], a collaborative variational AE for recommendation is developed. It models the implicit relationships among items and users by making use of a shared latent representation and the variational regularization. A task similar to recommendation is tagging, since tags can be ranked for an item according to its similarity to other items. AEs have been also used as the core of tagging systems [90] using denoising AEs and relational denoising AEs.

V.6. Conclusions

Throughout this text, we have summarized the traditional alternatives for learning representations, the origins and essential characteristics of AEs, including how to introduce certain behaviors into the coding layer.

Later, we have thoroughly examined several case studies of AE applications in unstructured data as well as images and sequences: data visualization, image denoising, semantic hashing, anomaly detection and instance generation. Other applications have also been briefly discussed: image superresolution, image compression, transfer learning, human pose recovery and recommender systems.

We can conclude that AEs are a versatile framework for solving a wide variety of problems where a central task is to learn representations of the data. They can adapt to a given problem in structure as well as in the objective they optimize. This way, if the solution to a problem can be modeled with a transformation of the feature space onto another space, there will be many instances where the parameters of the transformation can be learned by an AE.

Acknowledgments: D. Charte is supported by the Spanish Ministry of Science, Innovation and Universities under the FPU National Program (Ref. FPU17/04069). This paper is partially supported by the Spanish National Research Projects TIN2015-68854-R and TIN2017-89517-P and project DeepSCOP Ayudas Fundación BBVA a Equipos de Investigación Científica en Big Data 2018.

References

[1] Pedro Domingos. "A Few Useful Things to Know About Machine Learning". In: *Communnications of the ACM* 55.10 (Oct. 2012), pp. 78–87. DOI: 10.1145/2347736.2347755 (cit. on pp. 2, 4).

[2] Ana C Lorena et al. "How Complex Is Your Classification Problem?: A Survey on Measuring Classification Complexity". In: *ACM Computing Surveys (CSUR)* 52.5 (2019), p. 107. DOI: 10.1145/3347711 (cit. on p. 2).

[3] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015 (cit. on p. 2).

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828. DOI: 10.1109/TPAMI.2013.50 (cit. on p. 2).

[5] Ruslan Salakhutdinov and Geoffrey Hinton. "Semantic hashing". In: *International Journal of Approximate Reasoning* 50.7 (2009), pp. 969–978. DOI: 10.1016/j.ijar.2008.11.006 (cit. on pp. 2, 16, 17).

[6] Lucas Theis et al. "Lossy image compression with compressive autoencoders". In: *Fifth International Conference on Learning Representations*. 2017 (cit. on pp. 2, 24).

[7] Jun Deng et al. "Sparse autoencoder-based feature transfer learning for speech emotion recognition". In: *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*. IEEE. 2013, pp. 511–516. DOI: 10.1109/ACII.2013.90 (cit. on p. 2).

[8] Junyuan Xie, Linli Xu, and Enhong Chen. "Image denoising and inpainting with deep neural networks". In: *Advances in neural information processing systems*. 2012, pp. 341–349 (cit. on pp. 2, 11, 13).

[9] Isabelle Guyon et al. *Feature extraction: foundations and applications*. Vol. 207. Springer, 2008 (cit. on pp. 2, 4).

[10] Manoranjan Dash and Huan Liu. "Feature selection for classification". In: *Intelligent data analysis* 1.1-4 (1997), pp. 131–156. DOI: 10.1016/S1088-467X(97)00008-5 (cit. on p. 2).

[11] Ian T Jolliffe. "Introduction". In: *Principal component analysis*. Springer, 1986, pp. 1–7. DOI: 10.1007/978-1-4757-1904-8 (cit. on pp. 2, 5).

[12] Joshua B Tenenbaum, Vin De Silva, and John C Langford. "A global geometric framework for nonlinear dimensionality reduction". In: *Science* 290.5500 (2000), pp. 2319–2323. DOI: 10.1126/science.290.5500.2319 (cit. on pp. 2, 5).

[13] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *Science* 290.5500 (2000), pp. 2323–2326. DOI: 10.1126/science.290.5500.2323 (cit. on pp. 2, 5).

[14] Mikhail Belkin and Partha Niyogi. "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation". In: *Neural Computation* 15 (2003), pp. 1373–1396. DOI: 10.1162/089976603321780317 (cit. on pp. 2, 5, 12).

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), p. 436. DOI: 10.1038/nature14539 (cit. on p. 2).

[16] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988), p. 1. DOI: 10.1038/323533a0 (cit. on p. 2).

[17] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407. DOI: 10.1007/978-1-4612-5110-1_9 (cit. on p. 3).

[18] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159 (cit. on p. 3).

[19] Matthew D Zeiler. "ADADELTA: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012) (cit. on p. 3).

[20] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *Third International Conference on Learning Representations*. 2015 (cit. on p. 3).

[21] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-RMSProp". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31 (cit. on p. 3).

[22] David Charte et al. "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines". In: *Information Fusion* 44 (2018), pp. 78–96. DOI: 10.1016/j.inffus.2017.12.007 (cit. on pp. 3, 6, 7).

[23] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. "Supervised machine learning: A review of classification techniques". In: *Emerging artificial intelligence applications in computer engineering* 160 (2007), pp. 3–24 (cit. on p. 4).

[24] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. "Data Clustering: A Review". In: *ACM Computing Surveys* 31 (1999), pp. 264–323. DOI: 10.1145/331499.331504 (cit. on p. 4).

[25] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. *Dimensionality reduction: a comparative review*. Tech. rep. 2009 (cit. on p. 4).

[26] Ronald A. Fisher. "THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS". In: *Annals of Eugenics* 7.2 (Sept. 1936), pp. 179–188. DOI: 10.1111/j.1469-1809.1936.tb02137.x (cit. on pp. 4, 5).

[27] Haitao Zhao et al. "Local structure based supervised feature extraction". In: *Pattern Recognition* 39.8 (2006), pp. 1546–1550. DOI: 10.1016/j.patcog.2006.02.023 (cit. on p. 4).

[28] Alireza Makhzani et al. "Adversarial autoencoders". In: *Fourth International Conference on Learning Representations*. 2016 (cit. on pp. 4, 9).

[29] Jidong Zhao, Ke Lu, and Xiaofei He. "Locality sensitive semi-supervised feature selection". In: *Neurocomputing* 71.10-12 (2008), pp. 1842–1849. DOI: 10.1016/j.neucom.2007.06.014 (cit. on p. 4).

[30] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine Series 6* 2.11 (Nov. 1901), pp. 559–572. DOI: 10.1080/14786440109462720 (cit. on p. 5).

[31] Harold Hotelling. "Analysis of a complex of statistical variables into principal components". In: *Journal of educational psychology* 24.6 (1933), p. 417. DOI: 10.1037/h0071325 (cit. on p. 5).

[32] Ian T Jolliffe. "Principal Component Analysis and Factor Analysis". In: *Principal component analysis*. Springer, 1986, pp. 115–128. DOI: 10.1007/978-1-4757-1904-8 (cit. on p. 5).

[33] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. "Nonlinear component analysis as a kernel eigenvalue problem". In: *Neural computation* 10.5 (1998), pp. 1299–1319. DOI: 10.1162/089976698300017467 (cit. on p. 5).

[34] Bernhard Schölkopf. "Statistical Learning and Kernel Methods". In: *Data Fusion and Perception*. Ed. by Giacomo Della Riccia, Hans-Joachim Lenz, and Rudolf Kruse. Vienna: Springer Vienna, 2001, pp. 3–24. DOI: 10.1007/978-3-7091-2580-9_1 (cit. on p. 5).

[35] Paul Smolensky. *Information processing in dynamical systems: Foundations of harmony theory*. Tech. rep. Colorado University at Boulder, Department of Computer Science, 1986 (cit. on p. 5).

[36] Geoffrey E. Hinton. "Training Products of Experts by Minimizing Contrastive Divergence". In: *Neural Computation* 14.8 (Aug. 2002), pp. 1771–1800. DOI: 10.1162/089976602760128018 (cit. on p. 5).

[37] John W Sammon. "A nonlinear mapping for data structure analysis". In: *IEEE Transactions on computers* 100.5 (1969), pp. 401–409. DOI: 10.1109/T-C.1969.222678 (cit. on p. 5).

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems 25*. 2012, pp. 1097–1105 (cit. on p. 6).

[39] Jürgen Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *Neural networks* 61 (2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003 (cit. on p. 6).

[40] Teuvo Kohonen. "The self-organizing map". In: *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480. DOI: 10.1109/5.58325 (cit. on p. 6).

[41] Pasi Koikkalainen and Erkki Oja. "Self-organizing hierarchical feature maps". In: *1990 IJCNN international joint conference on neural networks*. IEEE. 1990, pp. 279–284. DOI: 10.1109/IJCNN.1990.137727 (cit. on p. 6).

[42] Jürgen Schmidhuber, Martin Eldracher, and Bernhard Foltin. "Semilinear predictability minimization produces well-known feature detectors". In: *Neural Computation* 8.4 (1996), pp. 773–786. DOI: 10.1162/neco.1996.8.4.773 (cit. on p. 6).

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. "Deep Learning". In: http://www.deeplearningbook.org. MIT Press, 2016. Chap. Deep generative models, pp. 651–716 (cit. on p. 6).

[44] Mark A Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE journal* 37.2 (1991), pp. 233–243. DOI: 10.1002/aic.690370209 (cit. on p. 6).

[45] Erkki Oja. "Data compression, feature extraction, and autoassociation in feedforward neural networks". In: *Artificial neural networks* 1 (1991). Ed. by Teuvo Kohonen et al., pp. 737–745 (cit. on p. 6).

[46] Yoshua Bengio. "Deep learning of representations for unsupervised and transfer learning". In: *Proceedings of ICML workshop on unsupervised and transfer learning*. 2012, pp. 17–36 (cit. on pp. 6, 24).

[47] Dana H Ballard. "Modular Learning in Neural Networks". In: *AAAI*. 1987, pp. 279–284 (cit. on p. 6).

[48] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554. DOI: 10.1162/neco.2006.18.7.1527 (cit. on p. 7).

[49] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323 (cit. on p. 7).

[50] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958 (cit. on p. 7).

[51] Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. "Sparse deep belief net model for visual area V2". In: *Advances in neural information processing systems 20*. 2008, pp. 873–880 (cit. on p. 8).

[52] Andrew Ng et al. "Sparse autoencoder". In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19 (cit. on p. 8).

[53] Salah Rifai et al. "Contractive auto-encoders: Explicit invariance during feature extraction". In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 833–840. (Visited on 08/28/2017) (cit. on pp. 9, 11, 12).

[54] Salah Rifai et al. "Higher order contractive auto-encoder". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2011, pp. 645–660. DOI: 10.1007/978-3-642-23783-6_41 (cit. on p. 9).

[55] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013) (cit. on pp. 9, 21).

[56] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294 (cit. on pp. 9, 11).

[57] Pascal Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *Journal of machine learning research* 11 (2010), pp. 3371–3408 (cit. on pp. 9, 13).

[58] Yu Qi et al. "Robust feature learning by stacked autoencoder with maximum correntropy criterion". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 6716–6720. DOI: 10.1109/ICASSP.2014.6854900 (cit. on p. 9).

[59] Weifeng Liu, Puskal P. Pokharel, and Jose C. Principe. "Correntropy: A localized similarity measure". In: *IEEE International Joint Conference on Neural Networks, 2006. IJCNN*. IEEE, 2006, pp. 4919–4924. DOI: 10.1109/IJCNN.2006.247192 (cit. on p. 9).

[60] Fuzhen Zhuang et al. "Supervised representation learning: Transfer learning with deep autoencoders". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015, pp. 4119–4125 (cit. on p. 10).

[61] Adam Coates, Andrew Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 215–223 (cit. on pp. 10, 15).

[62] I. Katakis, G. Tsoumakas, and I. Vlahavas. "Multilabel Text Classification for Automated Tag Suggestion". In: *Proceedings of the ECML/PKDD 2008*. 2008, pp. 75–83 (cit. on pp. 10, 17).

[63] N. Moustafa and J. Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)*. Nov. 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942 (cit. on pp. 10, 19).

[64] Martín Abadi et al. "Tensorflow: A system for large-scale machine learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283 (cit. on p. 10).

[65] François Chollet et al. *Keras*. https://keras.io. 2015 (cit. on p. 10).

[66] Ian Jolliffe. "Principal Component Analysis". In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1094–1096. DOI: 10.1007/978-3-642-04898-2_455 (cit. on p. 11).

[67] Wenchao Yu et al. "Embedding with autoencoder regularization". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2013, pp. 208–223. DOI: 10.1007/978-3-642-40994-3_14 (cit. on pp. 11, 12).

[68] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605 (cit. on p. 11).

[69] Junhua Li et al. "Feature learning from incomplete EEG with denoising autoencoder". In: *Neurocomputing* 165 (2015), pp. 23–31. DOI: 10.1016/j.neucom.2014.08.092 (cit. on p. 11).

[70] Warren S Torgerson. "Multidimensional scaling: I. Theory and method". In: *Psychometrika* 17.4 (1952), pp. 401–419. DOI: 10.1007/BF02288916 (cit. on p. 12).

[71] Raia Hadsell, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE. 2006, pp. 1735–1742. DOI: 10.1109/CVPR.2006.100 (cit. on p. 12).

[72] Xugang Lu et al. "Speech enhancement based on deep denoising autoencoder". In: *Interspeech*. 2013, pp. 436–440 (cit. on p. 13).

[73] Peng Xiong et al. "ECG signal enhancement based on improved denoising auto-encoder". In: *Engineering Applications of Artificial Intelligence* 52 (2016), pp. 194–202. DOI: 10.1016/j.engappai.2016.02.015 (cit. on p. 13).

[74] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman and Hall/CRC, 2014 (cit. on p. 16).

[75] Miguel A Carreira-Perpinán and Ramin Raziperchikolaei. "Hashing with binary autoencoders". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 557–566. DOI: 10.1109/CVPR.2015.7298654 (cit. on p. 17).

[76] Stephen Robertson. "Understanding inverse document frequency: on theoretical arguments for IDF". In: *Journal of documentation* 60.5 (2004), pp. 503–520. DOI: 10.1108/00220410410560582 (cit. on p. 17).

[77] Thomas Petsche et al. "A neural network autoassociator for induction motor failure prediction". In: *Advances in neural information processing systems 9*. 1996, pp. 924–930 (cit. on p. 18).

[78] Mayu Sakurada and Takehisa Yairi. "Anomaly detection using autoencoders with nonlinear dimensionality reduction". In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. ACM. 2014, pp. 4–11. DOI: 10.1145/2689746.2689747 (cit. on p. 18).

[79] Seungyoung Park, Myungjin Kim, and Seokwoo Lee. "Anomaly Detection for HTTP Using Convolutional Autoencoders". In: *IEEE Access* 6 (2018), pp. 70884–70901. DOI: 10.1109/ACCESS.2018.2881003 (cit. on p. 18).

[80] Yisroel Mirsky et al. "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection". In: *Network and Distributed Systems Security (NDSS) Symposium 2018*. Internet Society, 2018, pp. 1–15. DOI: 10.14722/ndss.2018.23204 (cit. on p. 18).

[81] N. Shone et al. "A Deep Learning Approach to Network Intrusion Detection". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.1 (Feb. 2018), pp. 41–50. DOI: 10.1109/TETCI.2017.2772792 (cit. on p. 18).

[82] Kun Zeng et al. "Coupled deep autoencoder for single image super-resolution". In: *IEEE transactions on cybernetics* 47.1 (2015), pp. 27–37. DOI: 10.1109/TCYB.2015.2501373 (cit. on p. 23).

[83] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. "End-to-end optimized image compression". In: *Fifth International Conference on Learning Representations*. 2017 (cit. on p. 24).

[84] Zhengxue Cheng et al. "Deep convolutional autoencoder-based lossy image compression". In: *2018 Picture Coding Symposium (PCS)*. IEEE. 2018, pp. 253–257. DOI: 10.1109/PCS.2018.8456308 (cit. on p. 24).

[85] Chaoqun Hong et al. "Multimodal deep autoencoder for human pose recovery". In: *IEEE Transactions on Image Processing* 24.12 (2015), pp. 5659–5670. DOI: 10.1109/TIP.2015.2487860 (cit. on p. 24).

[86] Zhuotun Zhu et al. "Deep learning representation using autoencoder for 3D shape retrieval". In: *Neurocomputing* 204 (2016), pp. 41–50. DOI: 10.1016/j.neucom.2015.08.127 (cit. on p. 24).

[87] Yueqing Wang et al. "An efficient and effective convolutional auto-encoder extreme learning machine network for 3d feature learning". In: *Neurocomputing* 174 (2016), pp. 988–998. DOI: 10.1016/j.neucom.2015.10.035 (cit. on p. 24).

[88] Biao Leng et al. "3D object retrieval with stacked local convolutional autoencoder". In: *Signal Processing* 112 (2015), pp. 119–128. DOI: 10.1016/j.sigpro.2014.09.005 (cit. on p. 24).

[89] Xiaopeng Li and James She. "Collaborative variational autoencoder for recommender systems". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2017, pp. 305–314. DOI: 10.1145/3097983.3098077 (cit. on p. 25).

[90] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. "Relational stacked denoising autoencoder for tag recommendation". In: *Twenty-ninth AAAI conference on artificial intelligence*. 2015 (cit. on p. 25).