A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines

David Charte[a,*], Francisco Charte[b], Salvador García[a], María J. del Jesus[b], Francisco Herrera[a,c]

[a] *Department of Computer Science and A.I., University of Granada, 18071 Granada, Spain*
[b] *Department of Computer Science, University of Jaén, 23071 Jaén, Spain*
[c] *Faculty of Computing and Information Technology, King Abdulaziz University, 21589, Jeddah, Saudi Arabia*

Abstract

Many of the existing machine learning algorithms, both supervised and unsupervised, depend on the quality of the input characteristics to generate a good model. The amount of these variables is also important, since performance tends to decline as the input dimensionality increases, hence the interest in using feature fusion techniques, able to produce feature sets that are more compact and higher level. A plethora of procedures to fuse original variables for producing new ones has been developed in the past decades. The most basic ones use linear combinations of the original variables, such as PCA (*Principal Component Analysis*) and LDA (*Linear Discriminant Analysis*), while others find manifold embeddings of lower dimensionality based on non-linear combinations, such as Isomap or LLE (*Linear Locally Embedding*) techniques.

More recently, autoencoders (AEs) have emerged as an alternative to manifold learning for conducting nonlinear feature fusion. Dozens of AE models have been proposed lately, each with its own specific traits. Although many of them can be used to generate reduced feature sets through the fusion of the original ones, there also AEs designed with other applications in mind.

The goal of this paper is to provide the reader with a broad view of what an AE is, how they are used for feature fusion, a taxonomy gathering a broad range of models, and how they relate to other classical techniques. In addition, a set of didactic guidelines on how to choose the proper AE for a given task is supplied, together with a discussion of the software tools available. Finally, two case studies illustrate the usage of AEs with datasets of handwritten digits and breast cancer.

Keywords: autoencoders, feature fusion, feature extraction, representation learning, deep learning, machine learning

1. Introduction

The development of the first machine learning techniques dates back to the middle of the 20th century, supported mainly by previously established statistical methods. By then, early research on how to emulate the functioning of the human brain through a machine was underway. McCulloch and Pitts cell [1] was proposed back in 1943, and the Hebb rule [2] that the Perceptron [3] is founded on was stated in 1949. Therefore, it is not surprising that artificial neural networks (ANNs), especially since the backpropagation algorithm was rediscovered in 1986 by Rumelhart, Hinton and Willians [4], have become one of the essential models.

ANNs have been applied to several machine learning tasks, mostly following a supervised approach. As was mathematically demonstrated [5] in 1989, a multilayer feed-forward ANN (MLP) is an universal approximator, hence their usefulness in classification and regression problems. However, a proper algorithm able to train an MLP with several hidden layers was not available, due to the vanishing gradient [6] problem. The gradient descent algorithm, firstly used for convolutional neural networks [7] and later for unsupervised learning [8], was one of the foundations of modern deep learning [9] methods.

Under the umbrella of deep learning, multiple techniques have emerged and evolved. These include DBNs (*Deep Belief Networks*) [10], CNNs (*Convolutional Neural Networks*) [11], RNNs (*Recurrent Neural Networks*) [12] as well as LSTMs (*Long Short-Term Memory*) [13] or AEs (*autoencoders*).

The most common architecture in unsupervised deep learning is that of the *encoder-decoder* [14]. Some techniques lack the encoder or the decoder and have to compute costly optimization algorithms to find a code or sampling methods to reach a reconstruction, respectively. Unlike those, AEs capture both parts in their structure, with the aim that training them becomes easier and faster. In

*Corresponding author. Tel.: +34 953 213 016
Email addresses: fdavidcl@correo.ugr.es (David Charte), fcharte@ujaen.es (Francisco Charte), salvagl@decsai.ugr.es (Salvador García), mjjesus@ujaen.es (María J. del Jesus), herrera@decsai.ugr.es (Francisco Herrera)

general terms, AEs are ANNs which produce codifications for input data and are trained so that their decodifications resemble the inputs as closely as possible.

AEs were firstly introduced [15] as a way of conducting pretraining in ANNs. Although mainly developed inside the context of deep learning, not all AE models are necessarily ANNs with multiple hidden layers. As explained below, an AE can be a deep ANN, i.e. in the stacked AEs configuration, or it can be a shallow ANN with a single hidden layer. See Section 2 for a more detailed introduction to AEs.

While many machine learning algorithms are able to work with raw input features, it is also true that, for the most part, their behavior is degraded as the number of variables grows. This is mainly due to the problem known as the *curse of dimensionality* [16], as well as the justification for a field of study called feature engineering. Engineering of features started as a manual process, relying in an expert able to decide by observation which variables were better for the task at hand. Notwithstanding, automated feature selection [17] methods were soon available.

Feature selection is only one of the approaches to reduce input space dimensionality. Selecting the best subset of input variables is an NP-hard combinatorial problem. Moreover, feature selection techniques usually evaluate each variable independently, but it is known that variables that separately do not provide useful information may do so when they are used together. For this reason other alternatives, primarily feature construction or extraction [18], emerged. In addition to these two denominations, feature selection and feature extraction, when dealing with dimensionality reduction it is also frequent to use other terms. The most common are as follows:

Feature engineering [19]. This is probably the broadest term, encompassing most of the others. Feature engineering can be carried out by manual or automated means, and be based on the selection of original characteristics or the construction of new ones through transformations.

Feature learning [20]. It is the denomination used when the process to select among the existing features or construct new ones is automated. Thus, we can perform both feature selection and feature extraction through algorithms such as the ones mentioned below. Despite the use of automatic methods, sometimes an expert is needed to decide which algorithm is the most appropriate depending on data traits, to evaluate the optimum amount of variables to extract, etc.

Representation learning [20]. Although this term is sometimes interchangeably used with the previous one, it is mostly used to refer to the use of ANNs to fully automate the feature generation process. Applying ANNs to learn distributed representations of concepts was proposed by Hinton in [21]. Today, learning representations is mainly linked to processing natural language, images and other signals with specific kinds of ANNs, such as CNNs [11].

Feature selection [22]. Picking the most informative subset of variables started as a manual process usually in charge of domain experts. It can be considered a special case of feature weighting, as discussed in [23]. Although in certain fields the expert is still an important factor, nowadays the selection of variables is usually carried out using computer algorithms. These can operate in supervised or unsupervised manner. The former approach usually relies on correlation or mutual information between input and output variables [24, 25], while the latter tends to avoid redundancy among features [26]. Feature selection is overall an essential strategy in the data preprocessing [27, 22] phase.

Feature extraction [28]. The goal of this technique is to find a better data representation for the machine learning algorithm intended to use, since the original representation might not be the best one. It can be faced both manually, in which case the *feature construction* term is of common use, and automatically. Some elemental techniques such as normalization, discretization or scaling of variables, as well as basic transformations applied to certain data types[1], are also considered within this field. New features can be extracted by finding linear combinations of the original ones, as in PCA (*Principal Component Analysis*) [29, 30] or LDA (*Linear Discriminant Analysis*) [31], as well as nonlinear combinations, like Kernel PCA [32] or Isomap [33]. The latter ones are usually known as *manifold learning* [34] algorithms, and fall in the scope of nonlinear dimensionality reduction techniques [35]. Feature extraction methods can also be categorized as supervised (e.g. LDA) or non-supervised (e.g. PCA).

Feature fusion [36]. This more recent term has emerged with the growth of multimedia data processing by machine learning algorithms, especially images, text and sound. As stated in [36], feature fusion methods aim to combine variables to remove redundant and irrelevant information. Manifold learning algorithms, and especially those based on ANNs, fall into this category.

Among the existing AE models there are several that are useful to perform feature fusion. This is the aim of the most basic one, which can be extended via several regularizations and adaptations to different kinds of data. These options will be explored through the present work, whose aim is to provide the reader with a didactic review on the inner workings of these distinct AE models and the ways they can be used to learn new representations of data.

The following are the main contributions of this paper:

- A proposal of a global taxonomy of AEs dedicated to feature fusion.

[1] e.g. Take the original field containing a date and divide it into three new variables, year, month and day.

- Descriptions of these AE models including the necessary mathematical formulation and explanations.
- A theoretical comparison between AEs and other popular feature fusion techniques.
- A comprehensive review of other AE models as well as their applications.
- A set of guidelines on how to design an AE, and several examples on how an AE may behave when its architecture and parameters are altered.
- A summary of the available software for creating deep learning models and specifically AEs.

Additionally, we provide a case study with the well known dataset MNIST [37], which gives the reader some intuitions on the results provided by an AE with different architectures and parameters. The scrips to reproduce these experiments are provided in a repository, and their use will be further described in Section 6.

The rest of this paper is structured as follows. The foundations and essential aspects of AEs are introduced in Section 2, including the proposal of a global taxonomy. Section 3 is devoted to thoroughly describing the AE models able to operate as feature fusion mechanisms and several models which have further applications. The relationship between these AE models and other feature fusion methods is portrayed in Section 4, while applications of different kinds of AEs are described in Section 5. Section 6 provides a set of guidelines on how to design an AE for the task at hand, followed by the software pieces where it can be implemented, as well as the case study with MNIST data. Concluding remarks can be found in Section 7. Lastly, an Appendix briefly describes the datasets used through the present work.

2. Autoencoder essentials

AEs are ANNs[2] with a symmetric structure, where the middle layer represents an *encoding* of the input data. AEs are trained to reconstruct their input onto the output layer, while verifying certain restrictions which prevent them from simply copying the data along the network. Although the term *autoencoder* is the most popular nowadays, they were also known as autoassociative neural networks [38], diabolo networks [39] and replicator neural networks [40].

In this section the foundations of AEs are introduced, describing their basic architecture as ANNs as well as the activation functions regularly applied in their layers. Next, AEs are grouped into four types according to their architecture. This is followed by our proposed taxonomy for AEs, which takes into account the properties these induce in codifications. Lastly, a summary of their habitual applications is provided.

[2]Strictly speaking not all AEs are ANNs, but here our interest is in those since they are the most common ones.

2.1. General structure

The basic structure of an AE, as shown in Fig. 1, includes an input x which is mapped onto the encoding y via an encoder, represented as function f. This encoding is in turn mapped to the reconstruction r by means of a decoder, represented as function g.

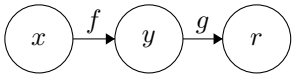


Figure 1: General autoencoder structure

This structure is captured in a feedforward neural network. Since the objective is to reproduce the input data on the output layer, both x and r have the same dimension. y, however, can be higher-dimensional or lower-dimensional, depending on the properties desired. The AE can also have as many layers as needed, usually placed symmetrically in the encoder and decoder. Such a neural architecture can be observed in Fig. 2.

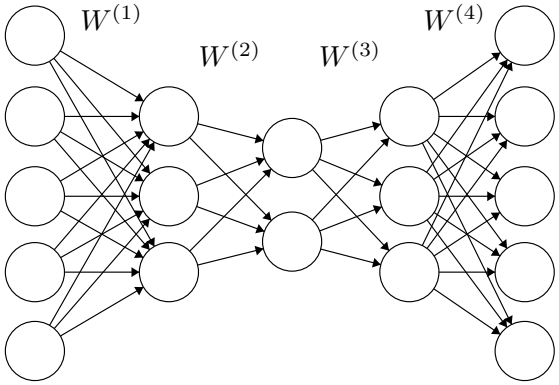


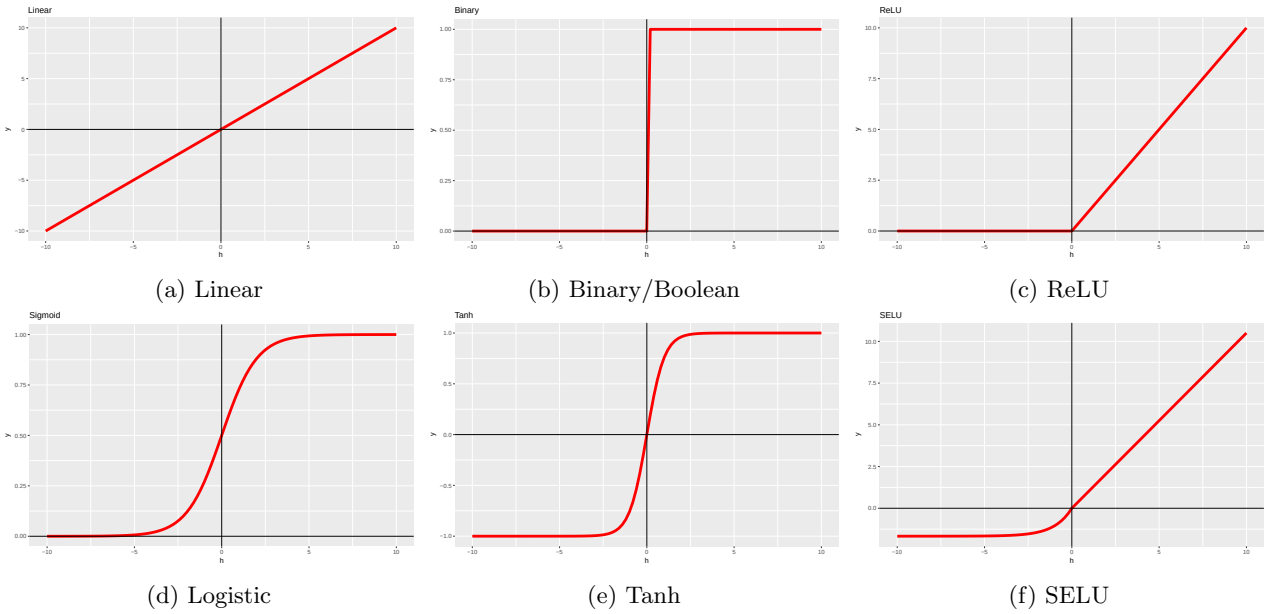
Figure 2: A possible neural architecture for an autoencoder with a 2-variable encoding layer. W denotes weight matrices.

In this case the encoder is made up of three layers, including the middle encoding one, while the decoder starts in the middle one and also spans three layers.

2.2. Activation functions of common use in autoencoders

A unit located in any of the hidden layers of an ANN receives several inputs from the preceding layer. The unit computes the weighted sum of these inputs and eventually applies a certain operation, the so-called *activation function*, to produce the output. The nonlinearity behavior of most ANNs is founded on the selection of the activation function to be used. Fig. 3 shows the graphical appearance of six of the most popular ones.

The activation functions shown in the first row are rarely used on AEs when it comes to learning higher level features, since they rarely provide useful representations. An undercomplete AE having one hidden layer made up of k linear activation units (Eq. 1) and that minimizes the sum of squared errors is known to be equivalent to obtaining the k principal components of the feature space



(a) Linear (b) Binary/Boolean (c) ReLU

(d) Logistic (e) Tanh (f) SELU

Figure 3: Common activation functions in ANNs.

via PCA [41–43]. AEs using binary/boolean activations (Eq. 2) [44, 45] are mostly adopted for educational uses, as McCulloch and Pitts [1] cells are still used in this context. However, they also have some specific applications, such as data hashing as described in subsection 5.4.

$$s_{\text{linear}}(x) = x \tag{1}$$

$$s_{\text{binary}}(x) = [x > 0] \tag{2}$$

Note that square brackets denote Iverson's convention [46] and evaluate to 0 or 1 according to the truthiness of the proposition.

Rectified linear units (ReLU, Fig. 3c, Eq. 3) are popular in many deep learning models, but it is an activation function that tends to degrade the AE performance. Since it always outputs 0 for negative inputs, it weakens the process of reconstructing the input features onto the outputs. Although they have been successfully used in [47, 48], the authors had to resort to a few detours. A recent alternative which combines the benefits of ReLU while circumventing these obstacles is the SELU function (*Scaled Exponential Linear Units*, Fig. 3f, Eq. 4) [49]. There are already some proposals of deep AEs based on SELU such as [50].

$$s_{\text{relu}}(x) = x[x > 0] \tag{3}$$

$$s_{\text{selu}}(x) = \lambda \begin{cases} \alpha e^x - \alpha & x \leq 0 \\ x & x > 0 \end{cases}, \text{ where } \lambda > 1 \tag{4}$$

Sigmoid functions are undoubtedly the most common activations in AEs. The standard logistic function, popularly known simply as sigmoid (Fig. 3d, Eq. 5), is probably the most frequently used. The hyperbolic tangent (Fig. 3e, Eq. 6) is also a sigmoid function, but it is symmetric about the origin and presents a steeper slope. According to LeCun [51] the latter should be preferred, since its derivative produces stronger gradients that the former.

$$s_{\text{sigm}}(x) = \sigma(x) = \frac{1}{1 + e^x} \tag{5}$$

$$s_{\text{tanh}}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{6}$$

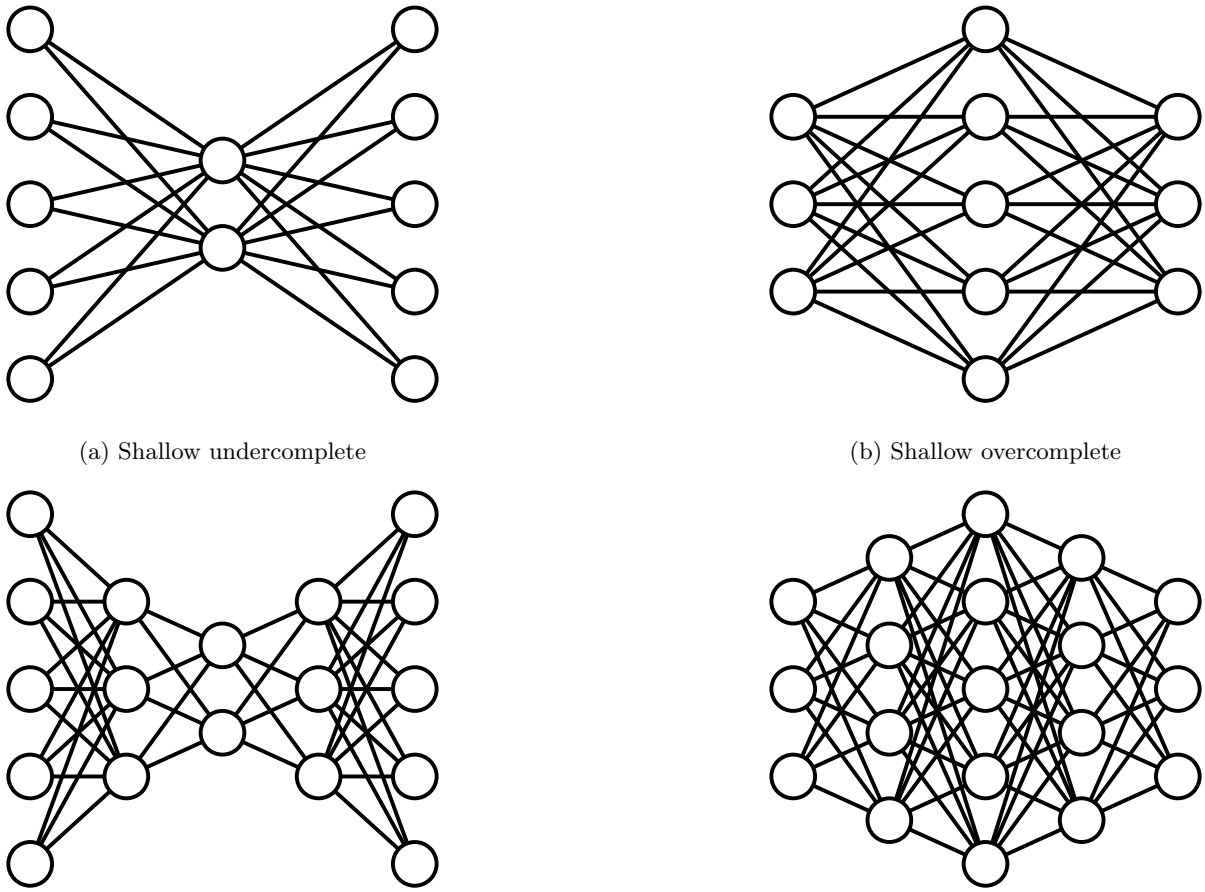
When designing AEs with multiple hidden layers, it is possible to use different activation functions in some of them. This would result in AEs combining the characteristics of several of these functions.

2.3. Autoencoder groups according to network structure

AEs could be grouped according to disparate principles, such as their structure, the learning algorithm they use, the loss function that guides the update of weights, their activation function or the field they are applied. In this section we focus on the first criterion, while the others will be further covered in following sections.

As explained above, AEs are ANNs with a symmetrical structure. The decoder and the encoder have the same number of layers, with the number of units per layer in reverse order. The encoding layer is shared by both parts. Depending on the dimensionality of the encoding layer, AEs are said to be:

- *Undercomplete*, if the encoding layer has a lower dimensionality than the input. The smaller number of units imposes a restriction, so during training the AE is forced to learn a more compact representation. This is achieved by fusing the original features according to the weights assigned through the learning process.



(a) Shallow undercomplete

(b) Shallow overcomplete

(c) Deep undercomplete

(d) Deep overcomplete

Figure 4: Autoencoder models according to their structure.

- *Overcomplete*, otherwise. An encoding layer having the same or more units than the input could allow the AE to simply learn the identity function, copying the input onto the output. To avoid this behavior, usually other restrictions are applied as will be explained later.

Although the more popular AE configuration for dimensionality reduction is undercomplete, an overcomplete AE with the proper restrictions can also produce a compact encoding as explained in subsection 3.2.1.

In addition to the number of units per layer, the structure of an AE is also dependent of the number of layers. According to this factor, an AE can be:

- *Shallow*, when it only comprises three layers (input, encoding and output). It is the simplest AE model, since there is only one hidden layer (the encoding).
- *Deep*, when it has more than one hidden layer. This kind of AE can be trained either layer by layer, as several shallow stacked AEs, or as a deep ANN [52].

These four types of AEs are visually summarized in Fig. 4. Shallow AEs are on the top row and deep ones in the bottom, while undercomplete AEs are on the left column and overcomplete on the right one.

2.4. Autoencoder taxonomy

As stated before, a taxonomy of AEs can be built according to different criteria. Here the interest is mainly on the properties of the inferred model regarding the feature fusion task. Conforming to this principle, we have elaborated the taxonomy shown in Fig. 5. As can be seen, there are four main categories in this taxonomy:

Lower dimensionality. High-dimensional data can be an issue when using most classifiers and especially shallow neural networks, since they do not perform any kind of high-level feature learning and are then forced to optimize a notable amount of parameters. This task may be eased by just lowering the dimensionality of the data, and this is the aim of the basic AE, which is thoroughly explained in Section 3.1. Decreasing the dimensionality of specific types of data, such as images or sequences, can be treated by domain specific AEs detailed in Section 3.4.

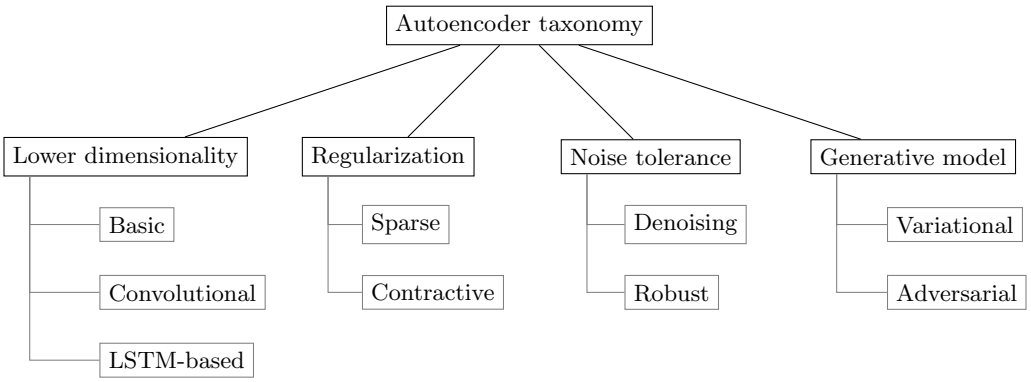


Figure 5: Taxonomy: most popular autoencoders classified according to the charasteristics they induce in their encodings

Regularization. Sometimes, learned features are required to present special mathematical properties. AEs can be easily modified in order to reach encodings that verify them. The main regularizations that can be applied to AEs are portrayed in Section 3.2.

Noise tolerance. In addition to different properties, a desirable trait for the encoded space may be robustness in the face of noisy data. Two distinct approaches to this problem using AEs are gathered in Section 3.3.

Generative model. The transformation from the original feature space onto the encoded space may not be the main objective of an AE. Occasionally it will be useful to map new samples in the encoded space onto the original features. In this case, a generative model is needed. Those based in AEs are specified in Section 3.5.

2.5. Usual applications

The term autoencoder is very broad, referring to multiple learning models based on both fully-connected feed-forward ANNs and other types of ANNs, and even models completely unrelated to that structure. Similarly, the application fields of AEs are also varied. In this work we pay attention specifically to AEs whose basic model is that of an ANN. In addition, we are especially interested in those whose objective is the fusion of characteristics by means of nonlinear techniques.

Reducing the dimensionality of a feature space using AEs can be achieved following disparate approaches. Most of them are reviewed in Section 3, starting with the basic AE model, then advancing to those that include a regularization, that present noise tolerance, etc. The goal is to provide a broad view of the techniques that AEs rely on to perform feature fusion.

Besides feature extraction, which is our main focus, there are AE models designed for other applications such as outlier detection, hashing, data compression or data generation. In Sections 3.5 and 3.6 some of these AEs will be briefly portrayed, and in Section 5 many of their applications will be shortly reviewed.

3. Autoencoders for feature fusion

As has been already established, AEs are tools originally designed for finding useful representations of data by learning nonlinear ways to combine their features. Usually, this leads to a lower-dimensional space, but different modifications can be applied in order to discover features which satisfy certain requirements. All of these possibilities are discussed in this section, which begins by establishing the foundations of the most basic AE, and later encompasses several diverse variants, following the proposed taxonomy: those that provide regularizations are followed by AEs presenting noise tolerance, generative models are explained afterwards, then some domain specific AEs and finally two variations which do not fit into any previous category.

3.1. Basic autoencoder

The main objective of most AEs is to perform a feature fusion process where learned features present some desired traits, such as lower dimensionality, higher sparsity or desirable analytical properties. The resulting model is able to map new instances onto the latent feature space. All AEs thus share a common origin, which may be called the basic AE [53].

The following subsections define the structure of a basic AE, establish their objective function, describe the training process while enumerating the necessary algorithms for this task, and depict how a deep AE can be initialized by stacking several shallow ones.

3.1.1. Structure

The structure of a basic AE, as shown in the previous section, is that of a feed forward ANN where layers are of symmetrical amount of units. Layers need not be symmetrical in the sense of activation functions or weight matrices.

The simplest AE consists of just one hidden layer, and is defined by two weight matrices and two bias vectors:

$$y = f(x) = s_1(W^{(1)}x + b^{(1)}), \tag{7}$$

$$r = g(y) = s_2(W^{(2)}y + b^{(2)}), \tag{8}$$

where s_1 and s_2 denote the activation functions, which usually are nonlinear.

Deep AEs are the natural extension of this definition to a higher number of layers. We will call the composition of functions in the encoder f, and the composition of functions in the decoder g.

3.1.2. Objective function

AEs generally base their objective function on a per-instance loss function $\mathcal{L} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$:

$$\mathcal{J}(W, b; S) = \sum_{x \in S} \mathcal{L}(x, (g \circ f)(x)) \tag{9}$$

where f and g are the encoding and decoding functions determined by the weights W and biases b, assuming activation functions are fixed, and S is a set of samples. The objective of an AE is thus to optimize W and b in order to minimize $\mathcal{J}$.

For example, a typical loss function is the mean squared error (MSE):

$$\mathcal{L}_{\text{MSE}}(u, v) = \|u - v\|_2^2 \ . \tag{10}$$

Notice that multiplying by constants or performing the square root of the error induced by each instance does not alter the process, since these operations preserve numerical order. As a consequence, the root mean squared error (RMSE) is an equivalent loss metric.

When a probabilistic model is assumed for the input samples, the loss function is chosen as the negative log-likelihood for the example x given the output $(g \circ f)(x)$ [54]. For instance, when input values are binary or modeled as bits, cross-entropy is usually the preferred alternative for the loss function:

$$\mathcal{L}_{\text{CE}}(u, v) = -\sum_{k=1}^{d} u_k \log v_k + (1 - u_k) \log(1 - v_k) \ . \tag{11}$$

3.1.3. Training

Usual algorithms for optimizing weights and biases in AEs are stochastic gradient descent (SGD) [55] and some of its variants, such as AdaGrad [56], RMSProp [57] and Adam [58]. Other applicable algorithms which are not based on SGD are L-BFGS and conjugate gradient [59].

The foundation of these algorithms is the technique of gradient descent [60]. Intuitively, at each step, the gradient of the objective function with respect to the parameters shows the direction of steepest slope, and allows the algorithm to modify the parameters in order to search for a minimum of the function.

In order to compute the necessary gradients, the backpropagation algorithm [4] is applied. Backpropagation performs this computation by calculating several intermediate terms from the last layer to the first.

AEs, like many other machine learning techniques, are susceptible to overfitting of the training data. To avoid this issue, a regularization term can be added to the objective function which causes a *weight decay* [61]. This improves the generalization ability and encourages smaller weights that produce good reconstructions. Weight decay can be introduced in several ways, but essentially consists in a term depending on weight sizes that will attempt to limit their growth. For example, the resulting objetive function could be

$$\mathcal{J}(W, b; S) = \sum_{x \in S} \mathcal{L}(x, (g \circ f)(x)) + \lambda \sum_i w_i^2 \tag{12}$$

where w_i traverses all the weights in W and λ is a parameter determining the magnitude of the decay.

Further restrictions and regularizations can be applied. A specific constraint that can be imposed is to tie the weight matrices symmetrically, that is, in a shallow AE, to set $W^{(1)} = (W^{(2)})^T$, and the natural extension to deep AEs. This allows to optimize a lower amount of parameters, so the AE can be trained faster, while maintaining the desired architecture.

3.1.4. Stacking

When AEs are deep, the success of the training process relies heavily on a good weight initialization, since there can be from tens to hundreds of thousands of them. This weight initialization can be performed by *stacking* successive shallow AEs [54], that is, training the AE layer by layer in a greedy fashion.

The training process begins by training only the first hidden layer as the encoding of a shallow AE, as shown by the network on the left of Fig. 6. After this step, the second hidden layer is trained, using the first hidden layer as input layer, as displayed on the right. Inputs are computed via a forward pass of the original inputs through the first layer, with the weights determined during the previous stage. Each successive layer up to the encoding is trained the same way.

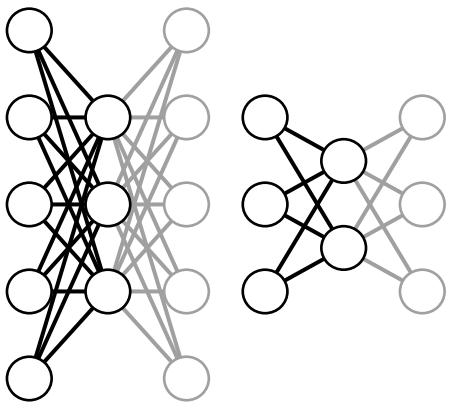


Figure 6: Greedy layer-wise training of a deep AE with the architecture shown in Fig 4c. Units drawn in black designate layers of the final AE, and gray ones indicate layers that are not part of the unrolled AE during the fine-tuning phase.

After this layer-wise training, initial weights for all layers preceding and including the encoding layer will have been computed. The AE is now "unrolled", i.e. the rest of layers are added symetrically with weight matrices resulting from transposing the ones from each corresponding layer. For instance, for the AE trained in Fig. 6, the unrolled AE would have the structure shown in Fig. 4c.

Finally, a fine-tuning phase can be performed, optimizing the weights by backpropagating gradients through the complete structure with training data.

3.2. Regularization

Encodings produced by basic AEs do not generally present any special properties. When learned features are required to verify some desirable traits, some regularizations may be achieved by adding a penalization for certain behaviors Ω to the objetive function:

$$\mathcal{J}(W, b; S) = \sum_{x \in S} \mathcal{L}(x, (g \circ f)(x)) + \lambda \Omega(W, b; S) \ . \quad (13)$$

3.2.1. Sparse autoencoder

Sparsity in a representation means most values for a given sample are zero or close to zero. Sparse representations are resembling of the behavior of simple cells in the mammalian primary visual cortex, which is believed to have evolved to discover efficient coding strategies [62]. This motivates the use of transformations of samples into sparse codes in machine learning. A model of sparse coding based on this behavior was first proposed in [63].

Sparse codes can also be overcomplete and meaningful. This was not necessarily the case in basic AEs, where an overcomplete code would be trained to just copy inputs onto outputs.

When sparsity is desired in the encoding generated by an AE, activations of the encoding layer need to have low values in average, which means units in the hidden layer usually do not fire. The activation function used in those units will determine this low value: in the case of sigmoid and ReLU activations, low values will be close to 0; this value will be -1 in the case of tanh, and $-\lambda\alpha$ in the case of a SELU.

The common way to introduce sparsity in an AE is to add a penalty to the loss function, as proposed in [64] for Deep Belief Networks. In order to compare the desired activations for a given unit to the actual ones, these can be modeled as a Bernoulli random variable, assuming a unit can only either fire or not. For a specific input x, let

$$\hat{\rho}_i = \frac{1}{|S|} \sum_{x \in S} f_i(x) \quad (14)$$

be the average activation value of an unit in the hidden layer, where $f = (f_1, f_2, \dots f_c)$ and c is the number of units in the encoding. $\hat{\rho}_i$ will be the mean of the associated Bernoulli distribution.

Let ρ be the desired average activation. The Kullback-Leibler divergence [65] between the random variable defined by unit i and the one corresponding to the desired activations will measure how different both distributions are [66]:

$$\mathrm{KL}(\rho \| \hat{\rho}_i) = \rho \log \frac{\rho}{\hat{\rho}_i} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_i} \ . \quad (15)$$

Fig. 7 shows the penalty caused by Kullback-Leibler divergence for a hidden unit when the desired average activation is $\rho = 0.2$. Notice that the penalty is very low when the average activation is near the desired one, but grows rapidly as it moves away and tends to infinity at 0 and 1.

The resulting penalization term for the objective function is

$$\Omega_{\mathrm{SAE}}(W, b; S) = \sum_{i=1}^{c} \mathrm{KL}(\rho \| \hat{\rho}_i) \ , \quad (16)$$

where the average activation value $\hat{\rho}_i$ depends on the parameters of the encoder and the training set S.

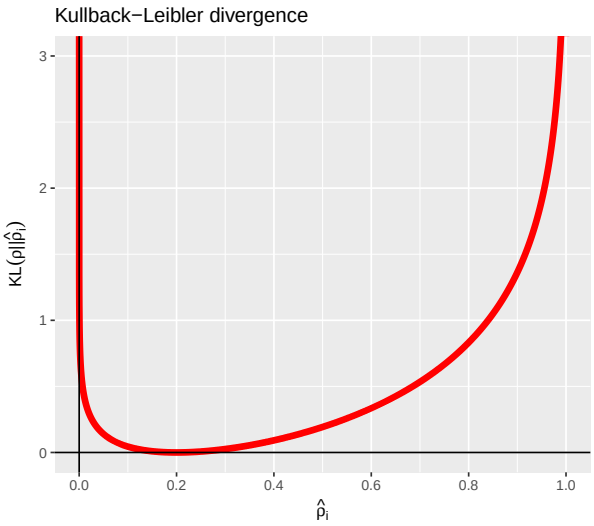


Figure 7: Values of Kullback-Leibler divergence for a unit with average activation $\hat{\rho}_i$.

There are other modifications that can lead an encoder-decoder architecture to produce a sparse code. For example, applying a sparsifying logistic activation function in the encoding layer of a similar energy-based model, which forces a low activation average [67], or using a Sparse Encoding Symmetric Machine [68] which optimizes a loss function with a different sparsifying penalty.

3.2.2. Contractive autoencoder

High sensitivity to perturbations in input samples could lead an AE to generate very different encodings. This is usually inconvenient, which is the motivation behind the contractive AE. It achieves local invariance to changes in many directions around the training samples, and is able to more easily discover lower-dimensional manifold structures in the data.

Sensitivity for small changes in the input can be measured as the Frobenius norm $\|\cdot\|_F$ of the Jacobian matrix of the encoder J_f:

$$\|J_f(x)\|_F^2 = \sum_{j=1}^{d} \sum_{i=1}^{c} \left(\frac{\partial f_i}{\partial x_j}(x) \right)^2 . \quad (17)$$

The higher this value is, the more unstable the encodings will be to perturbations on the inputs.

A regularization is built from this measure into the objective function of the contractive AE:

$$\Omega_{\text{CAE}}(W, b; S) = \sum_{x \in S} \|J_f(x)\|_F^2 . \quad (18)$$

A particular case of this induced contraction is the usage of L2 weight decay with a linear encoder: in this situation, the only way to produce a contraction is to maintain small weights. In the nonlinear case, however, contraction can be encouraged by pushing hidden units to the saturated region of the activation function.

The contractive AE can be sampled [69], that is, it can generate new instances from the learned model, by using the Jacobian of the encoder to add a small noise to another point and computing its codification. Intuitively, this can be seen as moving small steps along the tangent plane defined by the encoder in a point on the manifold modeled.

3.3. Noise tolerance

A standard AE can learn a latent feature space from a set of samples, but it does not guarantee stability in the presence of noisy instances, nor it is able to remove noise when reconstructing new samples. In this section, two variants that tackle this problem are discussed: denoising and robust AEs.

3.3.1. Denoising autoencoder

A denoising AE or DAE [70] learns to generate robust features from inputs by reconstructing partially destroyed samples. The use of AEs for denoising had been introduced earlier [71], but this technique leverages the denoising ability of the AE to build a latent feature space which is more resistant to corrupted inputs, thus its applications are broader than just denoising.

The structure and parameters of a denoising AE are identical to those of a basic AE. The difference here lies in a stochastic corruption of the inputs which is applied during the training phase. The corrupting technique proposed in [70], as illustrated by Fig. 8, is to randomly choose a fixed amount of features for each training sample and set them to 0. The reconstructions of the AE are however compared to the original, uncorrupted inputs. The AE will be thus be trained to guess the missing values.

Formally, let $q(\tilde{x}|x)$ be a stochastic mapping performing the partial destruction of values described above, the denoising AE recieves $\tilde{x} \sim q(\tilde{x}|x)$ as input and minimizes

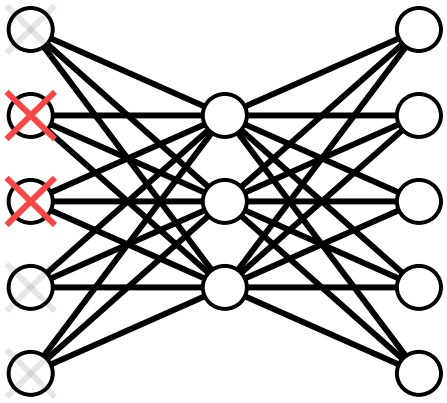


Figure 8: Illustration of the training phase of a denoising AE. For each input sample, some of its components are randomly selected and set to 0, but the reconstruction error is computed by comparing to the original, non-corrupted data.

$$\mathcal{J}_{\text{DAE}}(W, b; S) = \sum_{x \in S} \mathbb{E}_{\tilde{x} \sim q(\tilde{x}|x)} \left[\mathcal{L}(x, (g \circ f)(\tilde{x})) \right] . \quad (19)$$

A denoising AE does not need further restrictions or regularizations in order to learn a meaningful coding from the data, which means it can be overcomplete if desired. When it has more than one hidden layer, it can be trained layer-wise. For this to be done, uncorrupted inputs are computed as outputs of the previous layers, these are then corrupted and provided to the network. Note that after the denoising AE is trained, it is used to compute higher-level representations without corrupting the input data.

The training technique allows for other possible corruption processes, apart from forcing some values to 0 [72]. For instance, additive Gaussian noise

$$\tilde{x} \sim \mathcal{N}(x, \sigma^2 \text{I}) , \quad (20)$$

which randomly offsets each component of x with the same variance, or *salt-and-pepper* noise, which sets a fraction of the elements of the input to their minimum or maximum value, according to a uniform distribution.

3.3.2. Robust autoencoder

Training an AE to recover from corrupted data is not the only way to induce noise tolerance in the generated model. An alternative is to modify the loss function used to minimize the reconstruction error in order to dampen the sensitivity to different types of noise.

Robust stacked AEs [73] apply this idea, and manage to be less affected by non-Gaussian noise than standard AEs. They achieve this by using a different loss function based on *correntropy*, a localized similarity measure defined in [74].

$$\mathcal{L}_{\mathrm{MCC}}(u,v) = -\sum_{k=1}^{d} \mathcal{K}_\sigma(u_k - v_k), \tag{21}$$

$$\text{where } \mathcal{K}_\sigma(\alpha) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\alpha^2}{2\sigma^2}\right), \tag{22}$$

and σ is a parameter for the kernel $\mathcal{K}$.

Correntropy specifically measures the probability density that two events are equal. An advantage of this metric is it being less affected by outliers than MSE. Robust AEs attempt to maximize this measure (equivalently, minimize negative correntropy), which translates in a higher resilience to non-Gaussian noise.

3.4. Domain specific autoencoders

The following two AEs are based on the standard type, but are designed to model very specific kinds of data, such as images and sequences.

Convolutional autoencoder [75]. Standard AEs do not explicitly consider the 2-dimensional structure when processing image data. Convolutional AEs solve this by making use of convolutional layers instead of fully connected ones. In these, a global weight matrix is used and the convolution operation is applied in order to forward pass values from one layer to the next. The same matrix is flipped over both dimensions and used for the reconstruction phase. Convolutional AEs can also be stacked and used to initialize CNNs [76], which are able to perform classification of images.

LSTM autoencoder [77]. A basic AE is not designed to model sequential data, an LSTM AE achieves this by placing Long-Short-Term Memory (LSTM) [78] units as encoder and decoder of the network. The encoder LSTM reads and compresses a sequence into a fixed-size representation, from which the decoder attempts to extract the original sequence in inverse order. This is especially useful when data is sequential and large, for example video data. A further possible task is to predict the future of the sequence from the representation, which can be achieved by attaching an additional decoder trained for this purpose.

3.5. Generative models

In addition to the models already described, which essentially provide different mechanisms to reduce the dimensionality of a set of variables, the following ones also produce a generative model from the training data. Generative models learn a distribution in order to be able to draw new samples, different from those observed. AEs can generally reconstruct encoded data, but are not necessarily able to build meaningful outputs from arbitrary encodings. Variational and adversarial AEs learn a model of the data from which new instances can be generated.

Variational autoencoder [79]. This kind of AE applies a variational Bayesian [80] approach to encoding. It assumes that a latent, unobserved random variable $\mathbf{y}$ exists, which by some random process leads to the observations, $\mathbf{x}$. Its objective is thus to approximate the distribution of the latent variable given the observations. Variational AEs replace deterministic functions in the encoder and decoder by stochastic mappings, and compute the objective function in virtue of the density functions of the random variables:

$$\mathcal{L}_{\mathrm{VAE}}(\theta,\phi;\mathbf{x}) = \\ \mathrm{KL}(q_\phi(\mathbf{y}|\mathbf{x})\|p_\theta(\mathbf{y})) - \mathbb{E}_{q_\phi(\mathbf{y}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{y})\right] \ , \tag{23}$$

where q is the distribution approximating the true latent distribution of $\mathbf{y}$, and θ, ϕ are the parameters of each distribution. Since variational AEs allow sampling from the learned distribution, applications usually involve generating new instances [81, 82].

Adversarial autoencoder [83]. It brings the concept of Generative Adversarial Networks [84] to the field of AEs. It models the encoding by imposing a prior distribution, then training a standard AE and, concurrently, a discriminative network trying to distinguish codifications from samples from the imposed prior. Since the generator (the encoder) is trained to fool the discriminator as well, encodings will tend to follow the imposed distribution. Therefore, adversarial AEs are also able generate new meaningful samples.

Other generative models based on similar principles are Variational Recurrent AEs [85], PixelGAN AEs [86] and Adversarial Symmetric Variational AEs [87].

3.6. Other autoencoders farther from feature fusion

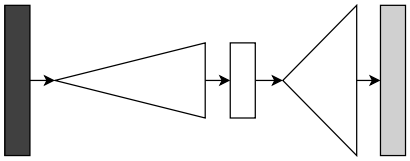
As can be seen, AEs can be easily altered to achieve different properties in their encoding. The following are some AEs which do not fall into any previous category.

Relational autoencoder. Basic AEs do not explicitly consider the possible relations among instances. The relational AE [88] modifies the objective function to take into account the fidelity of the reconstruction of relationships among samples. Instead of just adding a penalty term, the authors propose a weighted sum of the sample reconstruction error and the relation reconstruction error. Notice that this is not the only variation named "relational autoencoder" by its authors, different but identically named models are commented in sections 3.7 and 5.

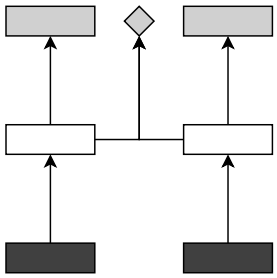
Discriminative autoencoder. Introduced in [89], the discriminative AE uses the class information of instances in order to build a manifold where positive samples are gathered and negative samples are pushed away. As a consequence, this AE performs better reconstruction of positive instances than negative ones. It achieves this by optimizing a different loss function, specifically the hinge loss function used in metric learning. The main objective of this model is object detection.

3.7. Autoencoder-based architectures for feature learning

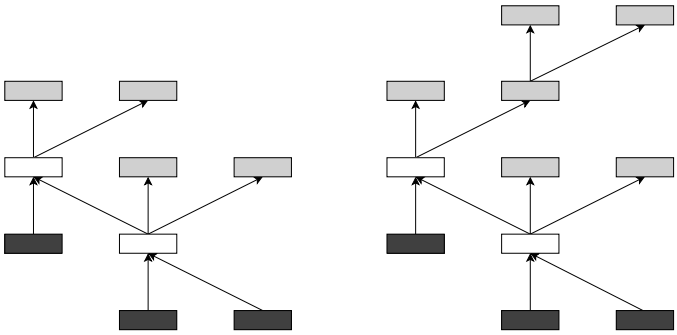
The basic AE can also be used as building block or inspiration for other, more complex architectures dedicated to feature fusion. This section enumerates and briefly introduces the most relevant ones.



(a) AE tree. Triangles represent decision trees.



(b) Dual AE. The encodings are coupled by an additional penalty term, represented as a diamond.



(c) Recursive AE (left), unfolded recursive AE (right)

Figure 9: Illustrations of autoencoder-based architectures. Each rectangle represents a layer, dark gray fill represents an input, light gray represents output layers and white objects represent hidden layers.

Autoencoder trees [90] (Fig. 9a) are encoder-decoder architectures, inspired by neural AEs, where the encoder as well as the decoder are actually decision trees. These trees use soft decision nodes, which means they propagate instances to all their children with different probabilities.

A dual-autoencoder architecture [91] (Fig. 9b) attempts to learn two latent representations for problems where variables can be treated as instances and viceversa, e.g. predicting customers' recommendations of items. These two representations are linked by an additional term in the objective function which minimizes their deviation from the training data.

The relational or "cross-correlation" AE defined in [92] incorporates layers where units are combined by multiplication instead of by a weighted sum. This allows it to represent co-ocurrences among components of its inputs.

A recursive AE [93] (Fig. 9c) is a tree-like architecture built from AEs, in which new pieces of input are introduced as the model gets deeper. A standard recursive AE attempts to reconstruct only the direct inputs of each encoding layer, whereas an unfolding recursive AE [94] reconstructs all previous inputs from each encoding layer. This architecture is designed to model sentiment in sentences.

4. Comparison to other feature fusion techniques

AEs are only several of a very diverse range of feature fusion methods [36]. These can be grouped according to whether they perform supervised or unsupervised learning. In the first case, they are usually known as *distance metric learning* techniques [95]. Some adversarial AEs, as well as AEs preserving class neighborhood structure [96], can be sorted into this category, since they are able to make use of the class information. However, this section focuses on the latter case, since most AEs are unsupervised and therefore share more similarities with this kind of methods.

A dimensionality reduction technique is said to be *convex* if it optimizes a function which does not have any local optima, and it is *nonconvex* otherwise [97]. Therefore, a different classification of these techniques is into convex and nonconvex approaches. AEs fall into the nonconvex group, since they can attempt to optimize disparate objective functions, and these may present more than one optimum. AEs are also not restrained to the dimensionality reduction domain, since they can produce sparse codes and other meaningful overcomplete representations.

Lastly, feature fusion procedures can be carried out by means of linear or nonlinear transformations. In this section, we aim to summarize the main traits of the most relevant approaches in both of these situations, and compare them to AEs.

4.1. Linear approaches

Principal component analysis is a statistical technique developed geometrically by Pearson [29] and algebraically by Hotelling [30]. It consists in the extraction of the *principal components* of a vector of random variables. Principal components are linear combinations of the original variables in a specific order, so that the first one has maximum variance, the second one has maximum possible variance while being uncorrelated to the first (equivalently, orthogonal), the third has maximum possible variance while being uncorrelated to the first and second, and so on. A modern analytical derivation of principal components can be found in [98].

The use of PCA for dimensionality reduction is very common, and can lead to reasonably good results. It is known that AEs with linear activations that minimize the mean quadratic error learn the principal components of the data [42]. From this perspective, AEs can be regarded as generalizations of PCA. However, as opposed to PCA, AEs can learn nonlinear combinations of the variables and even overcomplete representations of data.

Fig. 10 shows a particular occurrence of these facts in the case of the MNIST dataset [37]. Row 1 shows several test samples and the rest display reconstructions built by PCA and some AEs. As can be inferred from rows 2 and 3, linear AEs which optimize MSE learn an approximation of PCA. However, just by adjusting the activation functions and the objective function of the neural network one can obtain superior results (row 4). Improvements over the standard AE such as the robust AE (row 5) also provide higher quality in their reconstructions.

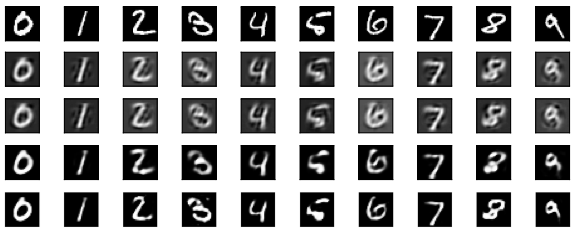


Figure 10: Row 1 shows test samples, second row corresponds to PCA reconstructions, the third one shows those from a linear AE optimizing MSE, row 4 displays reconstructions from a basic AE with tanh activation and cross-entropy as loss function, and last row corresponds to a robust AE.

A procedure similar to PCA but from a different theoretical perspective is Factor Analysis (FA) [99], which assumes a set of latent variables or *factors* which are not observable but are linearly combined to produce the observed variables. The difference between PCA and FA is similar to that between the basic AE and the variational AE: the latter assumes that hypothetical, underlying variables exist and cause the observed data. Variational AEs and FA attempt to find the model that best describes these variables, whereas the basic AE and PCA only aim for a lower-dimensional representation.

Linear Discriminant Analysis (LDA) [100] is a supervised statistical method to find linear combinations of features that achieve good separation of classes. It makes some assumptions of normality and homoscedasticity over the data, and projects samples onto new coordinates that best discriminate classes. It can be easily seen that AEs are very different in theory to this method: they usually perform unsupervised learning, and they do not necessarily make previous assumptions of the data. In contrast, AEs may not find the best separation of classes but they might encode further meaningful information from the data. Therefore, these techniques may be convenient, each in very different types of problems.

4.2. Nonlinear approaches

Kernel PCA [32] is an extension of PCA which applies kernel methods in order to extract nonlinear combinations of variables. Since principal components can be computed by projecting samples onto the eigenvectors of the covariance matrix, the kernel trick can be applied in order to calculate the covariance matrix of the data in a higher-dimensional space, given by the kernel function. Therefore, kernel PCA can compute nonlinear combinations of variables and overcomplete representations. The choice of kernel, however, can determine the success of the method and may behave differently with each problem, and hence AEs are a more general and easily applicable framework for nonlinear feature fusion.

Multidimensional Scaling (MDS) [101] is a well known technique and a foundation for other algorithms. It consists in finding new coordinates in a lower-dimensional space, while maintaining relative distances among data points as accurately as possible. For this to be achieved, it computes pairwise distances among points and then estimates an origin or zero point for these, which allows to transform relative distances into absolute distances that can be fitted into a real Euclidean space. Sammon mapping [102] modifies the classical cost function of MDS, in an attempt to similarly weigh retaining large distances as well as small ones. It achieves better preservation of local structure than classic MDS, at the cost of giving more importance to very small distances than large ones.

The approach of MDS to nonlinear feature fusion is opposite to that of AEs, which generally do not directly take into account distances among pairs of samples, and instead optimize a global measure of fitness. However, the objective function of an AE can be combined with that of MDS in order to produce a nonlinear embedding which considers pairwise distances among points [103].

Isomap [33] is a manifold learning method which extends MDS in order to find coordinates that describe the actual degrees of freedom of the data while preserving distances among neighbors and geodesic distances between the rest of points. In addition, Locally Linear Embedding (LLE) [104] has a similar goal, to learn a manifold which preserves neighbors, but a very different approach: it linearly reconstructs each point from its neighbors in order to maintain the local structure of the manifold.

Both of these techniques can be compared to the contractive AE, as it also attempts to preserve the local behavior of the data in its encoding. Denoising AEs may also be indirectly forced to learn manifolds, when they exist, and corrupted examples will be projected back onto their surface [72]. However, AEs are able to map new instances onto the latent space after they have been trained, a task Isomap and LLE are not designed for.

Laplacian Eigenmaps [105] is a framework aiming to retain local properties as well. It consists in constructing an adjacency graph where instances are nodes and neighbors are connected by edges. Then, a weight matrix similar to an adjacency matrix is built. Last, eigenvalues and eigenvectors are obtained for the Laplacian matrix associated to the weight matrix, and those eigenvectors (except 0) are used to compute new coordinates for each point. As previously mentioned, AEs do not usually consider the local structure of the data, except for contractive AEs and further regularizations which incorporate measures of local

properties into the objective function, such as Laplacian AEs [106].

A Restricted Boltzmann Machine (RBM) [107], introduced originally as *harmonium* in [108], is an undirected graphical model, with one visible layer and one hidden layer. They are defined by a joint probability distribution determined by an energy function. However, computing probabilities is unfeasible since the distribution is intractable, and they have been proved to be hard to simulate [109]. Instead, Contrastive Divergence [110] is used to train an RBM. RBMs are an alternative to AEs for greedy layer-wise initialization of weights in ANNs including AEs. AEs, however, are trained with more classical methods and are more easily adaptable to different tasks than RBMs.

5. Applications in feature learning and beyond

The ability of AEs to perform feature fusion is useful for easing the learning of predictive models, improving classification and regression results, and also for facilitating unsupervised tasks that are harder to conduct in high-dimensional spaces, such as clustering. Some specific cases of these applications are portrayed within the following subsections, including:

- Classification: reducing or transforming the training data in order to achieve better performance in a classifier.
- Data compression: training AEs for specific types of data to learn efficient compressions.
- Detection of abnormal patterns: identification of discordant instances by analyzing generated encodings.
- Hashing: summarizing input data onto a binary vector for faster search.
- Visualization: projecting data onto 2 or 3 dimensions with an AE for graphical representation.
- Other purposes: further applications of AEs.

5.1. Classification

Using any of the AE models described in Section 3 to improve the output of a classifier is something very common nowadays. Here only a few but very representative case studies are referenced.

Classifying tissue images to detect cancer nuclei is a very complicated accomplishment, due to the large size of high-resolution pathological images and the high variance of the fundamental traits of these nuclei, e.g. its shape, size, etc. The authors of [111] introduce a method, based on stacked DAEs to produce higher level and more compact features, which eases this task.

Multimodal/Multiview learning [112] is a rising technique which also found considerable support in AEs. The authors of [113] present a general procedure named *Orthogonal Autoencoder for Multi-View*. It is founded on DAEs to extract private and shared latent feature spaces, with an added orthogonality constraint to remove unnecessary connections. In [114] the authors propose the MSCAE (*Multimodal Stacked Contractive Autoencoder*), an application-specific model fed with text, audio and image data to perform multimodal video classification.

Multilabel classification [115] (MLC) is another growing machine learning field. MLC algorithms have to predict several outputs (labels) linked to each input pattern. These are usually defined by a high-dimensional feature vector and a set of labels as output which tend to be quite large as well. In [116] the authors propose an AE-based method named C2AE (*Canonical Correlated AutoEncoder*), aimed to learn a compressed feature space while establishing relationships between the input and output spaces.

Text classification following a semi-supervised approach by means of AEs is introduced in [117]. A model called SSVAE (*Semi-supervised Sequential Variational Autoencoder*) is presented, mixing a Seq2Seq [118] structure and a sequential classifier. The authors state that their method outperforms fully supervised methods.

Classifiers based on AEs can be grouped in ensembles in order to gain expressive power, but some diversity needs to be introduced. Several means of doing so, as well as a proposal for Stacked Denoising Autoencoding (SDAE) classifiers can be found in [119]. This method has set a new performance record in MNIST classification.

5.2. Data compression

Since AEs are able to reconstruct the inputs given to them, an obvious application would be compressing large amounts of data. However, as we already know, the AE output is not perfect, but an approximate reconstruction of the input. Therefore, it is useful only when lossy compression is permissible.

This is the usual scenario while working with images, hence the popularity of the JPEG [120] graphic file format. It is therefore not surprising that AEs have been successfully applied to this task. This is the case of [121], where the performance of several AE models compressing mammogram image patches is analyzed. A less specific goal can be found in [122]. It proposes a model of AE named SWTA AE (*Stochastic Winner-Take-All Auto-Encoder*), a variation of the sparse AE model, aimed to work as a general method able to achieve a variable ratio of image compression.

Although images could be the most popular data compressed by means of AEs, these have also demonstrated their capacity to work with other types of information as well. For instance:

- In [123] the authors suggest the use of AEs to compress biometric data, such as blood pressure or heart rate, retrieved by wearable devices. This way battery life can be extended while time transmission of data is reduced.

- Language compression is the goal of ASC (*Autoencoding Sentence Compression*), a model introduced in [124]. It is founded on a variational AE, used to draw sentences from a language modeled with a certain distribution.

- High-resolution time series of data, such as measurements taken from service grids (electricity, water, gas, etc.), tend to need a lot of space. In [125] the APRA (*Adaptive Pairwise Recurrent Encoder*) model is presented, combining an AE and a LSTM to successfully compress this kind of information.

Lossy compression is assumed to be tolerable in all these scenarios, so the approximate reconstruction process of the AE does not hinder the main objective in each case.

5.3. Detection of abnormal patterns

Abnormal patterns are samples present in the dataset that clearly differ from the remaining ones. The distinction between *anomalies* and *outliers* is usually found in the literature, although according to Aggarwal [126] these terms, along with *deviants*, *discordants* or *abnormalities*, refer to the same concept.

The telemetry obtained from spacecrafts is quite complex, made up of hundreds of variables. The authors of [127] propose the use of basic and denoising AEs for facing anomaly detection taking advantage of the nonlinear dimensionality reduction ability of these models. The comparison with both PCA and Kernel PCA demonstrates the superiority of AEs in this task.

The technique introduced in [128] aims to improve the detection of outliers. To do so, the authors propose to create ensembles of AEs with random connections instead of fully connected layers. Their model, named RandNet (*Randomized Neural Network for Outlier Detection*), is compared against four classic outlier detection methods achieving an outstanding performance.

A practical application of abnormal pattern detection with AEs is the one proposed in [129]. The authors of this work used a DAE, trained with a benchmark dataset, to identify fake twitter accounts. This way legitimate followers can be separated of those that are not.

5.4. Hashing

Hashing [130] is a very common technique in computing, mainly to create data structures able to offer constant access time to any element (*hash tables*) and to provide certain guarantees in cryptography (*hash values*). A special family of hash functions are those known as *Locality Sensitive Hashing* (LSH) [131]. They have the ability to map data patterns to lower dimensional spaces while maintaining some topological traits, such as the relative distance between these patterns. This technique is very useful for some applications, such as similar document retrieval. AEs can be also applied in these same fields.

Salakhutdinov and Hinton demonstrated in [132] how to perform what they call *semantic hashing* through a multi-layer AE. The fundamental idea is to restrict the values of the encoding layer units so that they are binary. In the example proposed in this study that layer has 128 or 20 units, sequences of ones and zeroes that are interpreted as an address. The aim is to facilitate the retrieval of documents, as noted above. The authors show how this technique offers better performance than the classic TF-IDF [133] or LSH.

Although the approach to generate the binary AE is different from the previous one, since they achieve hashing with binary AEs helped by MAC (*Method of Auxiliary Coordinates*) [134], the proposal in [135] is quite similar. The encoding layer produces a string of zeroes and ones, used in this case to conduct fast search of similar images in databases.

5.5. Data visualization

Understanding the nature of a given dataset can be a complex task when it posesses many dimensions. Data visualization techniques [136] can help analyze the structure of the data. One way of visualizing all instances in a dataset is to project it onto a lower-dimensional space which can be represented graphically.

A particular useful case of AEs are those with a 2 or 3-variable encoding [137]. This allows the generated codifications of samples to be displayed in a graphical representation such as the one in Fig. 11.

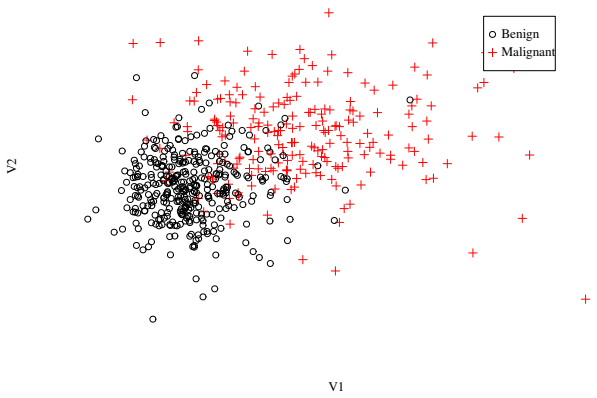


Figure 11: Example visualization of the codifications of a Cancer dataset generated with a basic AE with weight decay.

The original data [138] has 30 variables describing each pattern. Each data point is linked to one of two potential cancer diagnosis (classes), Benign and Malignant. These have been used in Fig. 11 to better show the separation between the two classes, but the V1 and V2 variables have been produced by the AE in an unsupervised fashion. Different projections could be obtained by adjusting the AE parameters.

5.6. Other applications of autoencoders

Beyond the specific applications within the four previous categories, which can be considered as usual in terms of the use of AEs, these find to be useful in many other cases. The following are just a few specific examples.

Holographic images [139] are a useful resource to store information in a fast way. However, retrieval of data has to face a common obstacle as is image degradation by the presence of speckle noise. In [140] an AE is trained with original holographic images as well as with degraded images, aiming to have a decoder able to reconstruct deteriorated examples. The denoising of images is also the goal of the method introduced in [141], although in this case they are medical images and the AE method is founded on convolutional denosing AEs.

The use of AEs to improve automatic speech recognition (ASR) systems has been also studied in late years. The authors of [142] rely on a DAE to reduce the noise and thus perform speech recognition enhancement. Essentially, the method gives the deep DAE noisy speech samples as inputs while the reference outputs are clean. A similar procedure is followed in [143], although in this case the problem present in the speech samples is reverberation. ASR is specially challenging when faced with whispered speech, as described in [144]. Once more, a deep DAE is the tool to improve results from classical approaches.

The procedure to curate biological databases is very expensive, so usually machine learning methods such as SVD (*Singular Value Decomposition*) [145] are applied to help in the process. In [41] this classical approach is compared with the use of deep AEs, reaching as conclusion that the latter is able to improve the results.

The authors of [146] aim to perform multimodal fusion by means of deep AEs, specifically proposing a Multimodal Deep Autoencoder (MDA). The goal is to perform human pose recovery from video [147]. To do so, two separate AEs are used to obtain high-level representations of 2D images and 3D human poses. Connecting these two AEs, a two-layer ANN carries out the mapping between the two representations.

Tagging digital resources, such as movies and products [148] or even questions in forums [149], helps the users in finding the information they are interested in, hence the importance in designing tag recommendation systems. The foundation of the approach in [150] is an AE variation named RSDAE (*Relational Stacked Denoising Autoencoder*). This AE works as a graphical model, combining the learning of high-level features with relationships among items.

AEs are also scalable to diverse applications with big data, where the stacking of networks acquires notable importance [151]. Multi-modal AEs and Tensor AEs are some examples of variants developed in this field.

6. Guidelines, software and examples on autoencoder design

This section attempts to guide the user along the process of designing an AE for a given problem, reviewing the range of choices the user has and their utility, then summarizing the available software for deep learning and outlining the steps needed to implement an AE. It also provides a case study with the MNIST dataset where the impact of several parameters of AEs is explored, as well as different AE types with identical parameter settings.

6.1. Guidelines

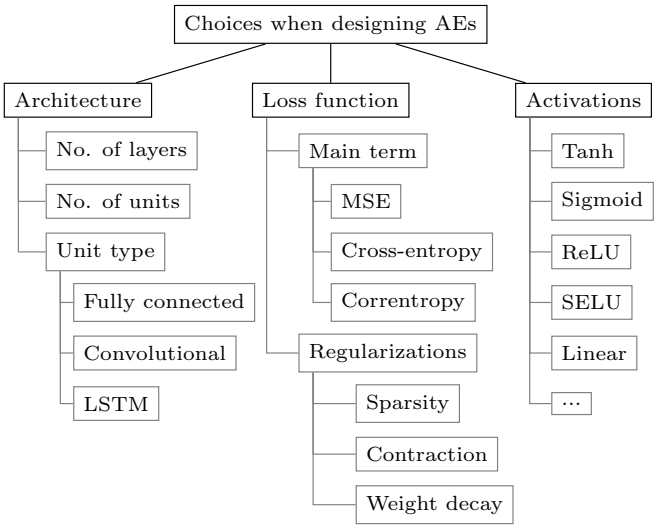


Figure 12: Summary of choices when designing an AE

When building an AE for a specific task, it is convenient to take into consideration the modifications studied in Section 3. There is no need to choose just one of those, most of them can actually be combined in the same AE. For instance, one could have a stacked denoising AE with weight decay and sparsity regularizations. A schematic summary of these can be viewed in Fig. 12.

Architecture. Firstly, one must define the structure of the AE, especially the length of the encoding layer. This is a fundamental step that will determine whether the training process can lead it to a good codification. If the length of the encoding is proportionally very low with respect to the number of original variables, training a deep stacked AE should be considered. In addition, convolutional layers are generally better performant with image data, whereas LSTM encoders and decoders would be preferable when modeling sequences. Otherwise, fully connected layers should be chosen.

Activations and loss function. Activation functions that will be applied within each layer have to be decided according to the loss function which will be optimized. For example, a sigmoid-like function such as the logistic or

tanh is generally a reasonable choice for the encoding layer, the latter being usually preferred due to its greater gradients. This does not need to coincide with the activation in the output layer. Placing a linear activation or ReLU at the output can be sensible when using mean squared error as reconstruction error, while a logistic activation would be better combined with the cross-entropy error and normalized data, since it outputs values between 0 and 1.

Regularizations. On top of that, diverse regularizations may be applied that will lead the AE to improve its encoding following certain criteria. It is generally advisable to add a small weight decay in order to prevent it from overfitting the training data. A sparse codification is useful in many cases and adds more flexibility to the choice of structure. Additionally, a contraction regularization may be valuable if the data forms a lower-dimensional manifold.

As seen in previous sections, AEs provide high flexibility and can be further modified for very different applications. In the case that the standard components do not fit the desired behavior, one must study which of those can be replaced and how, in order to achieve it.

6.2. Software

There exists a large spectrum of cross-platform, open source implementations of deep learning methods which allow for the construction and training of AEs. This section summarizes the most popular frameworks, enumerates some specific implementations of AEs, and provides an example of use where an AE is implemented on top of one of these frameworks.

6.2.1. Available frameworks and packages

Tensorflow [152]. Developed by Google, Tensorflow has been the most influential deep learning framework. It is based on the concept of *data flow graphs*, where nodes represent mathematical operations and multidimensional data arrays travel through the edges. Its core is written in C++ and interfaces mainly with Python, although there are APIs for Java, C and Go as well.

Caffe [153]. Originating at UC Berkeley, Caffe is built in C++ with speed and modularity in mind. Models are defined in a descriptive language instead of a common programming language, and trained in a C++ program.

Torch [154]. It is a Lua library which promises speed and flexibility, but the most notorious feature is its large ecosystem of community-contributed tutorials and packages.

MXNet [155]. This project is currently held at the Apache Incubator for incoming projects into the Apache Foundation. It is written in C++ and Python, and offers APIs in several additional languages, such as R, Scala, Perl and Julia. MXNet provides flexibility in the definition of models, which can be programmed symbolically as well as imperatively.

Keras [156]. Keras is a higher-level library for deep learning in Python, and can rely on Tensorflow, Theano, MXNet or Cognitive Toolkit for the underlying operations. It simplifies the creation of deep learning architectures by providing several shortcuts and predefined utilities, as well as a common interface for several deep learning toolkits.

In addition to the previous ones, other well known deep learning frameworks are Theano [157], Microsoft Cognitive Toolkit (CNTK[3]) and Chainer [4].

Setting various differences apart, all of these frameworks present some common traits when building AEs. Essentially, the user has to define the model layer by layer, placing activations where desired. When establishing the objective function, they will surely include the most usual ones, but uncommon loss functions such as correntropy or some regularizations such as contraction may need to be implemented additionally.

Very few pieces of software have specialized in the construction of AEs. Among them, there is an implementation of the sparse AE available in packages Autoencoder [158] and SAENET [159] of the CRAN repository for R, as well as an option for easily building basic AEs in H2O[5]. The yadlt[6] library for Python implements denoising AEs and several ways of stacking AEs.

6.2.2. Example of use

For the purposes of the case study in Section 6.3, some simple implementations of different shallow AEs have been developed and published on a public code repository under a free software license[7]. In order to use these scripts, the machine will need to have Keras and Tensorflow installed. This can be achieved from a Python package manager, such as `pip` or `pipenv`, or even general package managers from some Linux distributions.

In the provided repository, the reader can find four scripts dedicated to AEs and one to PCA. Among the first ones, `autoencoder.py` defines the Keras model for a given AE type with the specified activation for the encoding layer. For its part, `utils.py` implements regularizations and modifications in order to be able to define basic, sparse, contractive, denoising and robust AEs.

Executable scripts are `mnist.py` and `cancer.py`. The first trains any AE with the MNIST dataset and outputs a graphical representation of the encoding and reconstruction of some test instances, whereas the latter needs the Wisconsin Breast Cancer Diagnosis (WDBC) dataset in order to train an AE for it. To use them, just call the Python interpreter with the script as an argument, e.g. `python mnist.py`.

[3] `https://docs.microsoft.com/cognitive-toolkit/`
[4] `https://chainer.org/`
[5] `http://docs.h2o.ai`
[6] `https://deep-learning-tensorflow.readthedocs.io/`
[7] `https://github.com/fdavidcl/ae-review-resources`

In order to modify the learned model in one of these scripts, the user will need to adjust parameters in the construction of an `Autoencoder` object. The following is an example which will define a sparse denoising AE:

```
dsae = Autoencoder(
  input_dim    = 784,   encoding_dim = 36,
  weight_decay = False, sparse       = True,
  contractive  = False, denoising    = True,
  robust       = False, activation   = "tanh"
)
```

Other numerical parameters for each AE type can be further customized inside the `build` method. The training process of this AE can be launched via a `MNISTTrainer` object:

```
MNISTTrainer(dsae).train(
  optimizer = "adam", epochs = 50,
  loss = losses.binary_crossentropy
).predict_test()
```

Finally, running the modified script will train the AE and output some graphical representations.

The `Autoencoder` class can be reused to train AEs with other datasets. For this, one would need to implement funtionality analogous to the `MNISTTrainer` class, which loads and prepares data, which is provided to the AE model to be trained. A different example can be found in the `CancerTrainer` class for the WDBC dataset.

6.3. Case study: handwritten digits

In order to offer some insight into the behavior of the main kinds of AE that can be applied to the same problem, as well as some of the key points in their configuration, we can study the resulting codifications and reconstructions when training them with the well known dataset of handwritten digits MNIST [37]. To do so, we have trained several AEs with the 60 000 training instances, and have obtained reconstructions for the first test instance of each class. Input values, originally ranging from 0 to 255, have been scaled to the $[0, 1]$ interval.

By default, the architecture of every AE has been as follows: a 784-unit input layer, a 36-unit encoding layer with tanh activation and a 784-unit output layer with sigmoid activation. They have been trained with the RMSProp algorithm for a total of 60 epochs and use binary cross-entropy as their reconstruction error, except for the robust AE which uses its own loss function, correntropy. They are all provided identical weight initializations and hyperparameters.

Firstly, the performance impact of the encoding length and the optimizer is studied. Next, changes in the behavior of a standard AE due to different activation functions are analyzed. Lastly, the main AE models for feature fusion are compared sharing a common configuration. Scripts that were used to generate these results were implemented in Python, with the Keras library over the Tensorflow backend.

6.3.1. Settings of encoding length

As discussed previously, the number of units in the encoding layer can determine whether the AE is able to learn a useful representation. This fact is captured in Fig. 13, where an encoding of 16 variables is too small for the shallow AE to be successfully trained with the default configuration, but a 36-variable codification achieves reasonably good reconstructions. The accuracy of these can be improved at the cost of enlarging the encodings, as can be seen with the 81 and 144-variable encodings. Square numbers were chosen for the encoding lengths for easier graphical representation, as will be seen in Section 6.3.4, but any other length would have been as valid.

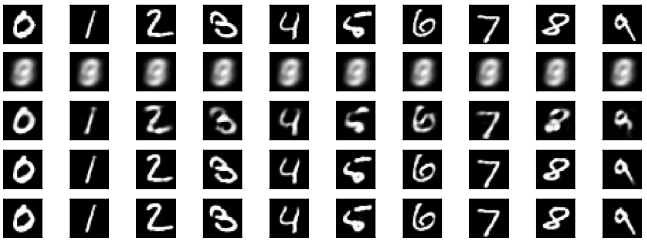


Figure 13: First row: test samples; Remaining rows: reconstructions obtained with 16, 36, 81 and 144 units in the encoding layer, respectively.

6.3.2. Comparison of optimizers

As introduced in Section 3.1.3, AEs can use several optimization methods, usually based on SGD. Each variant attempts to improve SGD in a different way, habitually by accumulating previous gradients in some way or dynamically adapting parameters such as the learning rate. Therefore, they will mainly differ in their ability to converge and their speed in doing so.

The optimizers used in these examples were baseline SGD, AdaGrad, Adam and RMSProp. Their progressive improvement of the objective function through the training phase is compared in Fig. 14. It is easily observed that SGD variants vastly improve the basic method, and Adam obtains the best results among them, being closely followed by AdaGrad. The speed of convergence seems slightly higher in Adam as well.

In addition, Fig. 15 provides the reconstructions generated for some test instances for a basic AE trained with each of those optimizers. As could be intuitively deduced by the convergence, or lack thereof, of the methods, SGD was not capable of finding weights which would recover any digit. AdaGrad, for its part, did improve on SGD but its reconstructions are relatively poor, whereas Adam and RMSProp display superior performance, with little difference between them.

6.3.3. Comparison of activation functions

Activation functions play an important role in the way gradients are propagated through the network. In this

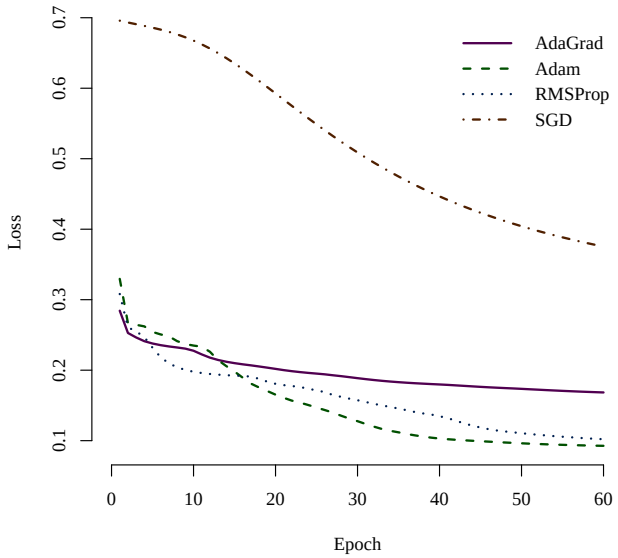


Figure 14: Evolution of the loss function when using several optimizers.

case, we apply four widely used activations in the encoding layer of a basic AE and compare how they affect its reconstruction ability. Some example results can be seen in Fig. 16.

Sigmoid and hyperbolic tangent are functions with similar properties, but in spite of this they produce remarkably dissimilar results. Reconstructions are poor when using sigmoidal activation, while tanh achieves representations much closer to the original inputs.

With respect to ReLU and SELU, the observed results are surprisingly solid and almost indistinguishable. They perform slightly better than tanh in the sense that reconstructions are noticeably sharper. Their good performance in this case may be due to the nature of the data, which is restricted to the $[0, 1]$ interval and does not necessarily show the behavior of these activations in general.

6.3.4. Comparison of the main AE models

It can be interesting to study the different traits the codifications may acquire when variations on the basic AE are introduced. The reconstructions produced by six different AE models are shown in Fig. 17.

The basic AE (Fig. 17a) and the one with weight decay (Fig. 17b) both generate recognizable reconstructions, although slighly blurry. They however do not produce much variability among different digits in the encoding layer, which means they are not making full use of its 36 dimensions. The weight decay corresponds to Eq. 12 with λ set to 0.01.

The sparse AE has been trained according to Eq. 15 with an expected activation value of -0.7. Its reconstructions are not much different from those of the previous ones, but in this case the encoding layer has much lower activations in average, as can be appreciated by the darker representations in Fig. 17c. Most of the information is therefore tightly condensed in a few latent variables.

The contractive AE achieves other interesting properties in its encoding: it has attempted to model the data as a lower dimensional manifold, where digits that seem more similar will be placed closer than those which are very unalike. As a consequence, the 0 and the 1 shown in Fig. 17d have very differing codifications, whereas the 3 and the 8 have relatively similar ones. Intuitively, one would need to travel larger distances along the learned manifold to go from a 0 to a 1, than from a 3 to an 8.

The denoising AE is able to eliminate noise from test instances, at the expense of losing some sharpness in the reconstruction, as can be seen in Fig. 17e. Finally, the robust AE (Fig. 17f) achieves noticeably higher clarity in the reconstruction and more variance in the encoding than the standard AEs.

7. Conclusions

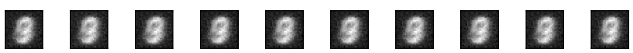
As Pedro Domingos states in his famous tutorial [19], and as can be seen from the large number of publications on the subject, feature engineering is the key to obtain good machine learning models, able to generalize and provide decent performance. This process consists in choosing the most relevant subset of features or combining some of them to create new ones. Automated fusion of features, specially when performed by nonlinear techniques, has demonstrated to be very effective. Neural network-based autoencoders are among the approaches to conduct this kind of task.

This paper started offering the reader with a general view of which an AE is, as well as its essential foundations. After introducing the usual AE network structures, a new AE taxonomy, based on the properties of the inferred model, has been proposed. Those AE models mainly used in feature fusion have been explained in detail, highlighting their most salient characteristics and comparing them with more classical feature fusion techniques. The use of disparate activation functions and training methods for AEs has been also thoroughly illustrated.

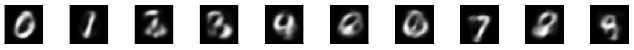
In addition to AEs for feature fusion, many other AE models and applications have been listed. The number of new proposals in this field is always growing, so it is easy to find dozens of AE variants, most of them based on the fundamental models described above.

This review is complemented by a final section proposing guidelines for selecting the most appropriate AE model based on different criteria, such as the type of units, loss function, activation function, etc., as well as mentioning available software to put this knowledge into practice. Empirical results on the well known MNIST dataset obtained from several AE configurations, combining disparate activation functions, optimizers and models, have been compared. The aim is to offer the reader help when facing this type of decision.

(a) Test inputs



(b) SGD



(c) AdaGrad

(d) Adam

(e) RMSProp

Figure 15: Test samples and reconstructions obtained with different optimizers.

(a) Test inputs

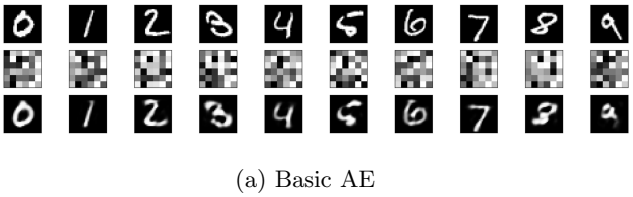
(b) Sigmoid

(c) Hyperbolic tangent

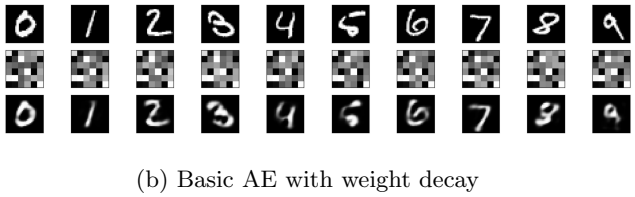
(d) ReLU

(e) SELU

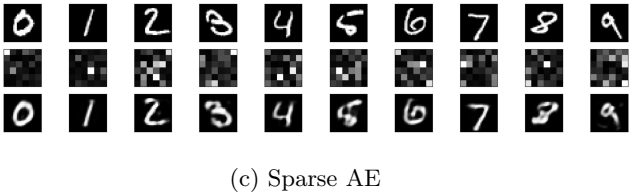
Figure 16: Test samples and reconstructions obtained with different activation functions.



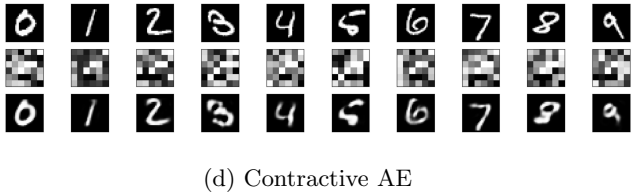
(a) Basic AE



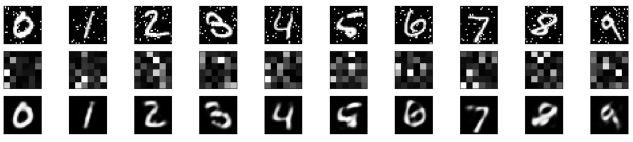
(b) Basic AE with weight decay



(c) Sparse AE



(d) Contractive AE



(e) Denoising AE with sparsity regularization

(f) Robust AE with weight decay

Figure 17: Reconstructing test samples with different AE models. First row of each figure shows test samples, second row shows activations of the encoding layer and third row displays reconstructions. Encoded values range from -1 (black) to 1 (white).

Acknowledgments: This work is supported by the Spanish National Research Projects TIN2015-68454-R and TIN2014-57251-P, and Project BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

References

[1] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, The bulletin of mathematical biophysics 5 (4) (1943) 115–133. doi:10.1007/BF02478259.

[2] D. O. Hebb, The organization of behavior: A neuropsychological theory, John Wiley And Sons, 1949. doi:10.1002/sce.37303405110.

[3] F. Rosenblatt, The perceptron, a perceiving and recognizing automaton (Project PARA), Cornell Aeronautical Laboratory, 1957.

[4] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–538. doi:10.1038/323533a0.

[5] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5) (1989) 359–366. doi:10.1016/0893-6080(89)90020-8.

[6] S. Hochreiter, The vanishing gradient problem during learning recurrent neural nets and problem solutions, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6 (02) (1998) 107–116. doi:10.1142/S0218488598000094.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural computation 1 (4) (1989) 541–551. doi:10.1162/neco.1989.1.4.541.

[8] G. E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural computation 18 (7) (2006) 1527–1554. doi:10.1162/neco.2006.18.7.1527.

[9] Jürgen Schmidhuber, Deep learning in neural networks: An overview, Neural networks : the official journal of the International Neural Network Society 61 (2015) 85–117. doi:10.1016/j.neunet.2014.09.003.

[10] G. E. Hinton, Deep belief networks, Scholarpedia 4 (5) (2009) 5947. doi:10.4249/scholarpedia.5947.

[11] Y. LeCun, Y. Bengio, The handbook of brain theory and neural networks, MIT Press, Cambridge, MA, USA, 1998, Ch. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258.

[12] R. J. Williams, D. Zipser, A learning algorithm for continually running fully recurrent neural networks, Neural computation 1 (2) (1989) 270–280. doi:10.1162/neco.1989.1.2.270.

[13] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[14] M. Ranzato, Y.-L. Boureau, S. Chopra, Y. LeCun, A unified energy-based framework for unsupervised learning, in: M. Meila, X. Shen (Eds.), Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, Vol. 2 of Proceedings of Machine Learning Research, PMLR, San Juan, Puerto Rico, 2007, pp. 371–379.

[15] D. H. Ballard, Modular learning in neural networks, in: Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1, AAAI'87, AAAI Press, 1987, pp. 279–284.

[16] R. Bellman, Dynamic Programming, Princeton University Press, 1957.

[17] M. Dash, H. Liu, Feature selection for classification, Intelligent Data Analysis 1 (1997) 131–156. doi:10.3233/IDA-1997-1302.

[18] H. Liu, H. Motoda, Feature extraction, construction and selection: A data mining perspective, Vol. 453, Springer Science & Business Media, 1998. doi:10.1007/978-1-4615-5725-8.

[19] P. Domingos, A few useful things to know about machine learning, Communications of the ACM 55 (10) (2012) 78–87. doi:10.1145/2347736.2347755.

[20] Y. Bengio, A. Courville, P. Vincent, Representation learning: A review and new perspectives, IEEE transactions on pattern analysis and machine intelligence 35 (8) (2013) 1798–1828. doi:10.1109/TPAMI.2013.50.

[21] G. E. Hinton, Learning distributed representations of concepts, in: Proceedings of the eighth annual conference of the cognitive science society, Vol. 1, Amherst, MA, 1986, p. 12. doi:10.1109/69.917563.

[22] S. García, J. Luengo, F. Herrera, Data preprocessing in data mining, Springer, 2015. doi:10.1007/978-3-319-10247-4.

[23] D. Wettschereck, D. W. Aha, T. Mohri, A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms, in: Lazy learning, Springer, 1997, pp. 273–314. doi:10.1023/A:1006593614256.

[24] M. A. Hall, Correlation-based feature selection for machine learning, Ph.D. thesis, University of Waikato Hamilton (1999).

[25] H. Peng, F. Long, C. H. Q. Ding, Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (2005) 1226–1238. doi:10.1109/TPAMI.2005.159.

[26] P. Mitra, C. A. Murthy, S. K. Pal, Unsupervised feature selection using feature similarity, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (3) (2002) 301–312. doi:10.1109/34.990133.

[27] S. García, J. Luengo, F. Herrera, Tutorial on practical tips of the most influential data preprocessing algorithms in data mining, Knowledge-Based Systems 98 (2016) 1–29. doi:10.1016/j.knosys.2015.12.006.

[28] I. Guyon, A. Elisseeff, An Introduction to Feature Extraction, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 1–25. doi:10.1007/978-3-540-35488-8_1.

[29] K. Pearson, LIII. On lines and planes of closest fit to systems of points in space, Philosophical Magazine Series 6 2 (11) (1901) 559–572. doi:10.1080/14786440109462720.

[30] H. Hotelling, Analysis of a complex of statistical variables into principal components, Journal of educational psychology 24 (6) (1933) 417. doi:10.1037/h0071325.

[31] R. A. Fisher, The statistical utilization of multiple measurements, Annals of Human Genetics 8 (4) (1938) 376–386. doi:10.1111/j.1469-1809.1938.tb02189.x.

[32] B. Schölkopf, A. Smola, K.-R. Müller, Nonlinear component analysis as a kernel eigenvalue problem, Neural computation 10 (5) (1998) 1299–1319. doi:10.1162/089976698300017467.

[33] J. B. Tenenbaum, V. De Silva, J. C. Langford, A global geometric framework for nonlinear dimensionality reduction, science 290 (5500) (2000) 2319–2323. doi:10.1126/science.290.5500.2319.

[34] L. Cayton, Algorithms for manifold learning, Tech. rep., University of California at San Diego (2005).

[35] J. A. Lee, M. Verleysen, Nonlinear dimensionality reduction, Springer Science & Business Media, 2007. doi:10.1007/978-0-387-39351-3.

[36] U. G. Mangai, S. Samanta, S. Das, P. R. Chowdhury, A survey of decision fusion and feature fusion strategies for pattern classification, IETE Technical review 27 (4) (2010) 293–307. doi:10.4103/0256-4602.64604.

[37] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324. doi:10.1109/5.726791.

[38] M. A. Kramer, Nonlinear principal component analysis using autoassociative neural networks, AIChE journal 37 (2) (1991) 233–243. doi:10.1002/aic.690370209.

[39] H. Schwenk, Y. Bengio, Training methods for adaptive boosting of neural networks, in: Advances in neural information processing systems, 1998, pp. 647–653. doi:10.1162/089976600300015178.

[40] R. Hecht-Nielsen, Replicator neural networks for universal optimal source coding, Science (1995) 1860–1863doi:10.1126/science.269.5232.1860.

[41] D. Chicco, P. Sadowski, P. Baldi, Deep autoencoder neural

networks for gene ontology annotation predictions, in: Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, ACM, 2014, pp. 533–540. doi:10.1145/2649387.2649442.

[42] P. Baldi, K. Hornik, Neural networks and principal component analysis: Learning from examples without local minima, Neural networks 2 (1) (1989) 53–58. doi:10.1016/0893-6080(89)90014-2.

[43] H. Kamyshanska, R. Memisevic, On autoencoder scoring, in: S. Dasgupta, D. McAllester (Eds.), Proceedings of the 30th International Conference on Machine Learning, Vol. 28 of Proceedings of Machine Learning Research, PMLR, Atlanta, Georgia, USA, 2013, pp. 720–728.

[44] L. Deng, M. L. Seltzer, D. Yu, A. Acero, A.-r. Mohamed, G. Hinton, Binary coding of speech spectrograms using a deep auto-encoder, in: Eleventh Annual Conference of the International Speech Communication Association, 2010.

[45] P. Baldi, Autoencoders, unsupervised learning, and deep architectures, in: Proceedings of ICML Workshop on Unsupervised and Transfer Learning, 2012, pp. 37–49.

[46] D. E. Knuth, Two notes on notation, The American Mathematical Monthly 99 (5) (1992) 403–422. doi:10.2307/2325085.

[47] X. Glorot, A. Bordes, Y. Bengio, Domain adaptation for large-scale sentiment classification: A deep learning approach, in: Proceedings of the 28th international conference on machine learning (ICML-11), 2011, pp. 513–520.

[48] Ç. Gülçehre, Y. Bengio, Knowledge matters: Importance of prior information for optimization, Journal of Machine Learning Research 17 (8) (2016) 1–32.

[49] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, Self-normalizing neural networks, in: Advances in Neural Information Processing Systems (NIPS), 2017.

[50] O. Kuchaiev, B. Ginsburg, Training deep autoencoders for recommender systems, in: International Conference on Learning Representations, 2018.

[51] Y. LeCun, L. Bottou, G. B. Orr, K. R. Müller, Efficient BackProp, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 9–50. doi:10.1007/3-540-49430-8_2.

[52] H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks, Journal of Machine Learning Research 10 (Jan) (2009) 1–40.

[53] H. Bourlard, Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition, Biological cybernetics 59 (4) (1988) 291–294. doi:10.1007/BF00332918.

[54] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks, in: Advances in neural information processing systems, 2007, pp. 153–160.

[55] H. Robbins, S. Monro, A stochastic approximation method, The annals of mathematical statistics (1951) 400–407doi:10.1007/978-1-4612-5110-1_9.

[56] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (Jul) (2011) 2121–2159.

[57] T. Tieleman, G. Hinton, Lecture 6.5-rmsprop, COURSERA: Neural networks for machine learning 4 (2) (2012) 26–31.

[58] D. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representations, 2015.

[59] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, A. Y. Ng, On optimization methods for deep learning, in: Proceedings of the 28th international conference on machine learning (ICML-11), 2011, pp. 265–272.

[60] A. Cauchy, Méthode générale pour la résolution des systemes déquations simultanées, Comptes Rendus des Séances de l'Académie des Sciences A (25) (1847) 536–538.

[61] A. Krogh, J. A. Hertz, A simple weight decay can improve generalization, in: Advances in neural information processing systems, 1992, pp. 950–957.

[62] B. A. Olshausen, D. J. Field, Sparse coding with an overcomplete basis set: A strategy employed by v1?, Vision research 37 (23) (1997) 3311–3325. doi:10.1016/S0042-6989(97)00169-7.

[63] B. A. Olshausen, D. J. Field, Emergence of simple-cell receptive field properties by learning a sparse code for natural images, Nature 381 (6583) (1996) 607–609. doi:10.1038/381607a0.

[64] H. Lee, C. Ekanadham, A. Y. Ng, Sparse deep belief net model for visual area v2, in: Advances in neural information processing systems, 2008, pp. 873–880.

[65] S. Kullback, R. A. Leibler, On information and sufficiency, The annals of mathematical statistics 22 (1) (1951) 79–86. doi:10.1214/aoms/1177729694.

[66] A. Ng, Sparse autoencoder, CS294A Lecture notes 72 (2011) (2011) 1–19.

[67] C. Poultney, S. Chopra, Y. L. Cun, et al., Efficient learning of sparse representations with an energy-based model, in: Advances in neural information processing systems, 2007, pp. 1137–1144.

[68] Y.-l. Boureau, Y. L. Cun, et al., Sparse feature learning for deep belief networks, in: Advances in neural information processing systems, 2008, pp. 1185–1192.

[69] S. Rifai, Y. Bengio, P. Vincent, Y. N. Dauphin, A generative process for sampling contractive auto-encoders, in: Proceedings of the 29th International Coference on International Conference on Machine Learning, ICML'12, Omnipress, 2012, pp. 1811–1818.

[70] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th International Conference on Machine Learning, ICML '08, ACM, 2008, pp. 1096–1103.

[71] Y. LeCun, Modèles connexionnistes de l'apprentissage, Ph.D. thesis, These de Doctorat, Université Paris 6 (1987).

[72] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, Journal of Machine Learning Research 11 (Dec) (2010) 3371–3408.

[73] Y. Qi, Y. Wang, X. Zheng, Z. Wu, 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.

[74] W. Liu, P. P. Pokharel, J. 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.

[75] J. Masci, U. Meier, D. Cireşan, J. Schmidhuber, Stacked convolutional auto-encoders for hierarchical feature extraction, in: T. Honkela, W. Duch, M. Girolami, S. Kaski (Eds.), Artificial Neural Networks and Machine Learning – ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I, Springer Berlin Heidelberg, 2011, pp. 52–59. doi:10.1007/978-3-642-21735-7_7.

[76] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, L. D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Computation 1 (1989) 541–551. doi:10.1162/neco.1989.1.4.541.

[77] N. Srivastava, E. Mansimov, R. Salakhudinov, Unsupervised learning of video representations using LSTMs, in: International Conference on Machine Learning, 2015, pp. 843–852.

[78] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[79] D. P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114.

[80] C. W. Fox, S. J. Roberts, A tutorial on variational bayesian inference, Artificial intelligence review (2012) 1–11doi:10.1007/s10462-011-9236-8.

[81] A. Dosovitskiy, T. Brox, Generating images with perceptual similarity metrics based on deep networks, in: Advances in Neural Information Processing Systems, 2016, pp. 658–666.

[82] D. J. Rezende, S. Mohamed, D. Wierstra, Stochastic backpropagation and approximate inference in deep generative models,

in: Proceedings of the 31st International Conference on Machine Learning (ICML-14), 2014, pp. 1278–1286.
[83] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, B. Frey, Adversarial autoencoders, arXiv preprint arXiv:1511.05644.
[84] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in neural information processing systems, 2014, pp. 2672–2680.
[85] O. Fabius, J. R. van Amersfoort, Variational recurrent auto-encoders, in: International Conference on Learning Representations, 2015.
[86] A. Makhzani, B. J. Frey, PixelGAN autoencoders, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 1972–1982.
[87] Y. Pu, W. Wang, R. Henao, L. Chen, Z. Gan, C. Li, L. Carin, Adversarial symmetric variational autoencoder, in: Advances in Neural Information Processing Systems, 2017, pp. 4331–4340.
[88] Q. Meng, D. Catchpoole, D. Skillicom, P. J. Kennedy, Relational autoencoder for feature extraction, in: 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, pp. 364–371. doi:10.1109/IJCNN.2017.7965877.
[89] S. Razakarivony, F. Jurie, Discriminative autoencoders for small targets detection, in: 22nd IEEE International Conference on Pattern Recognition (ICPR), IEEE, 2014, pp. 3528–3533. doi:10.1109/ICPR.2014.607.
[90] O. rsoy, E. Alpaydn, Unsupervised feature extraction with autoencoder trees, Neurocomputing 258 (2017) 63–73. doi:10.1016/j.neucom.2017.02.075.
[91] F. Z. He, Z. Zhang, M. Qian, C. Shi, X. Xie, Qing, Representation learning via Dual-Autoencoder for recommendation, Neural Networks 83–89doi:10.1016/J.NEUNET.2017.03.009.
[92] R. Memisevic, Gradient-based learning of higher-order image features, in: IEEE International Conference on Computer Vision (ICCV), IEEE, 2011, pp. 1591–1598. doi:10.1109/ICCV.2011.6126419.
[93] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, C. D. Manning, Semi-supervised recursive autoencoders for predicting sentiment distributions, in: Proceedings of the conference on empirical methods in natural language processing, Association for Computational Linguistics, 2011, pp. 151–161.
[94] R. Socher, E. H. Huang, J. Pennin, C. D. Manning, A. Y. Ng, Dynamic pooling and unfolding recursive autoencoders for paraphrase detection, in: Advances in Neural Information Processing Systems, 2011, pp. 801–809.
[95] F. Wang, J. Sun, Survey on distance metric learning and dimensionality reduction in data miningdoi:10.1007/s10618-014-0356-z.
[96] R. Salakhutdinov, G. E. Hinton, Learning a nonlinear embedding by preserving class neighbourhood structure, in: International Conference on Artificial Intelligence and Statistics, 2007, pp. 412–419.
[97] L. Van Der Maaten, E. Postma, J. Van den Herik, Dimensionality reduction: a comparative review, Tech. rep.
[98] I. T. Jolliffe, Introduction, in: Principal component analysis, Springer, 1986, pp. 1–7. doi:10.1007/978-1-4757-1904-8.
[99] I. 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.
[100] R. A. Fisher, The use of multiple measurements in taxonomic problems, Annals of Eugenics 7 (2) (1936) 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x.
[101] W. S. Torgerson, Multidimensional scaling: I. theory and method, Psychometrika 17 (4) (1952) 401–419. doi:10.1007/BF02288916.
[102] J. W. Sammon, A nonlinear mapping for data structure analysis, IEEE Transactions on computers 100 (5) (1969) 401–409. doi:10.1109/T-C.1969.222678.
[103] W. Yu, G. Zeng, P. Luo, F. Zhuang, Q. He, Z. Shi, Embedding with autoencoder regularization, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 208–223. doi:10.1007/978-3-642-40994-3_14.
[104] S. T. Roweis, L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, science 290 (5500) (2000) 2323–2326. doi:10.1126/science.290.5500.2323.
[105] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Computation 15 (2003) 1373–1396. doi:10.1162/089976603321780317.
[106] K. Jia, L. Sun, S. Gao, Z. Song, B. E. Shi, Laplacian auto-encoders: An explicit learning of nonlinear data manifold, Neurocomputing 160 (2015) 250–260. doi:10.1016/j.neucom.2015.02.023.
[107] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, Ch. Deep generative models, pp. 651–716, `http://www.deeplearningbook.org`.
[108] P. Smolensky, Information processing in dynamical systems: Foundations of harmony theory, Tech. rep., Colorado University at Boulder, Department of Computer Science (1986).
[109] P. M. Long, R. Servedio, Restricted boltzmann machines are hard to approximately evaluate or simulate, in: Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 703–710.
[110] G. E. Hinton, Training products of experts by minimizing contrastive divergence, Training 14 (8). doi:10.1162/089976602760128018.
[111] J. Xu, L. Xiang, Q. Liu, H. Gilmore, J. Wu, J. Tang, A. Madabhushi, Stacked Sparse Autoencoder (SSAE) for Nuclei Detection on Breast Cancer Histopathology Images, IEEE Transactions on Medical Imaging (1) 119–130. doi:10.1109/TMI.2015.2458702.
[112] C. Xu, D. Tao, C. Xu, A survey on multi-view learning, arXiv preprint arXiv:1304.5634.
[113] T. Ye, T. Wang, K. McGuinness, Y. Guo, C. Gurrin, Learning multiple views with orthogonal denoising autoencoders, in: Q. Tian, N. Sebe, G.-J. Qi, B. Huet, R. Hong, X. Liu (Eds.), MultiMedia Modeling, Vol. 9516, Springer International Publishing, 2016, pp. 313–324. doi:10.1007/978-3-319-27671-7_26.
[114] Y. Liu, X. Feng, Z. Zhou, Multimodal video classification with stacked contractive autoencoders, Signal Processing 120 (2016) 761–766. doi:10.1016/j.sigpro.2015.01.001.
[115] F. Herrera, F. Charte, A. J. Rivera, M. J. del Jesus, Multilabel Classification. Problem analysis, metrics and techniques, Springer, 2016. doi:10.1007/978-3-319-41111-8.
[116] C.-K. Yeh, W.-C. Wu, W.-J. Ko, Y.-C. F. Wang, Learning deep latent space for multi-label classification, in: AAAI Conference on Artificial Intelligence, 2017.
[117] W. Xu, H. Sun, C. Deng, Y. Tan, Variational autoencoder for semi-supervised text classification, in: AAAI Conference on Artificial Intelligence, 2017.
[118] N. Kalchbrenner, P. Blunsom, Recurrent continuous translation models., in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Vol. 3, 2013, p. 413.
[119] R. F. Alvear-Sandoval, A. R. Figueiras-Vidal, On building ensembles of stacked denoising auto-encoding classifiers and their further improvement, Information Fusion 39 (2018) 4152. doi:10.1016/j.inffus.2017.03.008.
[120] G. K. Wallace, The JPEG still picture compression standard, IEEE transactions on consumer electronics 38 (1) (1992) xviii–xxxiv. doi:10.1145/103085.103089.
[121] C. C. Tan, C. Eswaran, Using autoencoders for mammogram compression, Journal of Medical Systems 35 (1) (2011) 49–58. doi:10.1007/s10916-009-9340-3.
[122] T. Dumas, A. Roumy, C. Guillemot, Image compression with stochastic winner-take-all auto-encoder, in: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017), 2017. doi:10.1109/ICASSP.2017.7952409.
[123] D. Del Testa, M. Rossi, Lightweight lossy compression of biometric patterns via denoising autoencoders, IEEE Signal Processing Letters 22 (12) (2015) 2304–2308.

doi:10.1109/LSP.2015.2476667.
[124] Y. Miao, P. Blunsom, Language as a latent variable: Discrete generative models for sentence compression, in: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 2016, pp. 319–328.
[125] D. Hsu, Time series compression based on adaptive piecewise recurrent autoencoder, arXiv preprint arXiv:1707.07961.
[126] C. C. Aggarwal, An introduction to outlier analysis, in: Outlier analysis, Springer, 2013, pp. 1–40. doi:978-3-319-47578-3_1.
[127] M. Sakurada, T. Yairi, Anomaly detection using autoencoders with nonlinear dimensionality reduction, ACM Press, 2014, pp. 4–11. doi:10.1145/2689746.2689747.
[128] J. Chen, S. Sathe, C. Aggarwal, D. Turaga, Outlier detection with autoencoder ensembles, in: Proceedings of the 2017 SIAM International Conference on Data Mining, SIAM, 2017, pp. 90–98. doi:10.1137/1.9781611974973.11.
[129] J. Castellini, V. Poggioni, G. Sorbi, Fake Twitter followers detection by denoising autoencoder, in: Proceedings of the International Conference on Web Intelligence - WI '17, ACM Press, New York, New York, USA, pp. 195–202. doi:10.1145/3106426.3106489.
[130] L. Chi, X. Zhu, Hashing techniques: A survey and taxonomy, ACM Computing Surveys (CSUR) 50 (1) (2017) 11. doi:10.1145/3047307.
[131] A. Gionis, P. Indyk, R. Motwani, Similarity search in high dimensions via hashing, in: Proceedings of the 25th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 1999, pp. 518–529.
[132] R. Salakhutdinov, G. Hinton, Semantic hashing, International Journal of Approximate Reasoning 50 (7) (2009) 969–978. doi:10.1016/j.ijar.2008.11.006.
[133] G. Salton, E. A. Fox, H. Wu, Extended Boolean information retrieval, Communications of the ACM 26 (11) (1983) 1022–1036. doi:10.1145/182.358466.
[134] M. Carreira-Perpinán, W. Wang, Distributed optimization of deeply nested systems, in: Artificial Intelligence and Statistics, 2014, pp. 10–19.
[135] M. A. Carreira-Perpinán, R. 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.
[136] U. M. Fayyad, A. Wierse, G. G. Grinstein, Information visualization in data mining and knowledge discovery, Morgan Kaufmann, 2002.
[137] G. E. Hinton, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507. doi:10.1126/science.1127647.
URL http://www.sciencemag.org/cgi/doi/10.1126/science.1127647
[138] O. L. Mangasarian, W. N. Street, W. H. Wolberg, Breast cancer diagnosis and prognosis via linear programming, Operations Research 43 (4) (1995) 570–577. doi:10.1287/opre.43.4.570.
[139] T.-C. Poon, Digital holography and three-dimensional display: Principles and Applications, Springer Science & Business Media, 2006. doi:10.1007/0-387-31397-4.
[140] T. Shimobaba, Y. Endo, R. Hirayama, Y. Nagahama, T. Takahashi, T. Nishitsuji, T. Kakue, A. Shiraki, N. Takada, N. Masuda, T. Ito, Autoencoder-based holographic image restoration, Applied Optics (13) F27. doi:10.1364/AO.56.000F27.
[141] L. Gondara, Medical image denoising using convolutional denoising autoencoders, 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW) (2016) 241–246doi:10.1109/ICDMW.2016.0041.
[142] X. Lu, Y. Tsao, S. Matsuda, C. Hori, Speech enhancement based on deep denoising autoencoder, in: 14th Annual Conference of the International Speech Communication Association (INTERSPEECH), 2013.
[143] T. Ishii, H. Komiyama, T. Shinozaki, Y. Horiuchi, S. Kuroiwa, Reverberant speech recognition based on denoising autoencoder, in: 14th Annual Conference of the International Speech Communication Association (INTERSPEECH), 2013.
[144] D. T. Grozdić, S. T. Jovičić, Whispered speech recognition using deep denoising autoencoder and inverse filtering, IEEE/ACM Transactions on Audio, Speech, and Language Processing 25 (12) (2017) 2313–2322. doi:10.1109/TASLP.2017.2738559.
[145] G. H. Golub, C. Reinsch, Singular value decomposition and least squares solutions, Numerische mathematik 14 (5) (1970) 403–420. doi:10.1007/BF02163027.
[146] C. Hong, J. Yu, J. Wan, D. Tao, M. Wang, Multimodal Deep Autoencoder for Human Pose Recovery, IEEE Transactions on Image Processing (12) 5659–5670. doi:10.1109/TIP.2015.2487860.
[147] H. Zhou, T. Zhang, W. Lu, Vision-based pose estimation from points with unknown correspondences, IEEE Transactions on Image Processing 23 (8) (2014) 3468–3477. doi:10.1109/TIP.2014.2329765.
[148] J. Vig, S. Sen, J. Riedl, Tagsplanations: explaining recommendations using tags, in: Proceedings of the 14th international conference on Intelligent user interfaces, ACM, 2009, pp. 47–56. doi:10.1145/1502650.1502661.
[149] F. Charte, A. J. Rivera, M. J. del Jesus, F. Herrera, Quinta: a question tagging assistant to improve the answering ratio in electronic forums, in: IEEE EUROCON 2015-International Conference on Computer as a Tool, IEEE, 2015, pp. 1–6. doi:10.1109/EUROCON.2015.7313677.
[150] H. Wang, X. Shi, D.-Y. Yeung, Relational stacked denoising autoencoder for tag recommendation, in: Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
[151] Q. Zhang, L. T. Yang, Z. Chen, P. Li, A survey on deep learning for big data, Information Fusion 42 (2018) 146–157. doi:10.1016/j.inffus.2017.10.006.
[152] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, et al., Tensorflow: A system for large-scale machine learning, in: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16, 2016, pp. 265–283.
[153] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding (2014) 675–678.
[154] R. Collobert, K. Kavukcuoglu, C. Farabet, Torch7: A matlab-like environment for machine learning, in: BigLearn, NIPS Workshop, 2011.
[155] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems, arXiv preprint arXiv:1512.01274.
[156] F. Chollet, et al., Keras, https://github.com/fchollet/keras (2015).
[157] Theano Development Team, Theano: A Python framework for fast computation of mathematical expressions, arXiv e-prints abs/1605.02688.
[158] E. Dubossarsky, Y. Tyshetskiy, autoencoder: Sparse Autoencoder for Automatic Learning of Representative Features from Unlabeled Data, r package version 1.1 (2015).
URL https://CRAN.R-project.org/package=autoencoder
[159] S. Hogg, E. Dubossarsky, SAENET: A Stacked Autoencoder Implementation with Interface to 'neuralnet', r package version 1.1 (2015).
URL https://CRAN.R-project.org/package=SAENET
[160] D. Ciregan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2012, pp. 3642–3649.

Appendix A. Description of used datasets

Appendix A.1. Breast Cancer Diagnosis (Wisconsin)

The well known dataset of diagnosis of breast cancer in Wisconsin (WDBC) [138] is briefly used in Section 5.5 to provide a 2-dimensional visualization example.

This dataset consists of 569 instances corresponding to patients, each of which present 30 numeric input features and one of two classes that identify the type of tumor: benign or malignant. The dataset is slightly imbalanced, exhibiting a 37.3% of instances associated to the malignant class, while the remaining 62.7% correspond to benign tumors. The data have been normalized for the training process of the basic AE that generated the example.

Originally, features were extracted from a digitized image of a fine-needle aspiration sample of a breast mass, and described ten different traits of each cell nucleus. The mean, standard error and largest value of these features are computed, resulting in the 30 input attributes for each patient, gathered in the published dataset.

WDBC is usually relied on as an example dataset and most classifiers generally obtain high accuracy: the authors of the original proposal already achieved 97% of classification accuracy in cross-validation. However, it presents some issues when applying AEs: its small imbalance may cause instances classified as benign to contribute more to the loss function, inducing some bias in the resulting network, which may reconstruct these more accurately than the rest. Furthermore, it is composed of relatively few instances, which may not be sufficient for some deep learning techniques to be able to generalize.

Appendix A.2. MNIST

MNIST [37] is a widely used dataset within deep learning research. It is regularly chosen as a benchmark for new techniques and neural architectures. It has been the base of our case study in Section 6.3.

The dataset consists of 60 000 instances, divided into a 50 000-instance set for training and the remaining 10 000 for test. each corresponding to a 28x28-sized image of a handwritten digit, from 0 to 9. The values of this 28x28 matrix or 784-variable input represent the gray level of each pixel, and therefore range from 0 to 255, but they have been rescaled to the $[0, 1]$ interval in our examples.

This dataset is actually a modified subset of a previous work from NIST[8] for character recognition. The original images used only black or white pixels, whereas in MNIST they have been anti-aliased.

MNIST has been used as benchmark for a large variety of deep learning proposals, since it is reasonably easy to extract higher-level features out of simple grayscale images, and it provides a high enough amount of training data. State-of-the-art work[9] achieves an error rate of around 0.2% [119, 160].

[8] Available at `http://doi.org/10.18434/T4H01C`.

[9] A collection of methods applied to MNIST and their results is available at `http://yann.lecun.com/exdb/mnist/`.