Autoencoders

Dor Bank, Noam Koenigstein, Raja Giryes

Abstract An autoencoder is a specific type of a neural network, which is mainly designed to encode the input into a compressed and meaningful representation, and then decode it back such that the reconstructed input is similar as possible to the original one. This chapter surveys the different types of autoencoders that are mainly used today. It also describes various applications and use-cases of autoencoders.

1 Autoencoders

Autoencoders have been first introduced in [31] as a neural network that is trained to reconstruct its input. Their main purpose is learning in an unsupervised manner an "informative" representation of the data that can be used for various implications such as clustering. The problem, as formally defined in [1], is to learn the functions $A : \mathbb{R}^n \to \mathbb{R}^p$ (encoder) and $B : \mathbb{R}^p \to \mathbb{R}^n$ (decoder) that satisfy

$$\arg\min_{A,B} E[\Delta(\mathbf{x}, B \circ A(\mathbf{x})], \tag{1}$$

where E is the expectation over the distribution of x, and Δ is the reconstruction loss function, which measures the distance between the output of the decoder and the intput. The latter is usually set to be the ℓ_2-norm. Figure 1 provides an illustration of the autoencoder model.

Dor Bank
School of Electrical Engineering, Tel Aviv University, e-mail: dorbank@mail.tau.ac.il

Noam Koenigstein
Department of Industrial Engineering, Faculty of Engineering, Tel Aviv University, e-mail: noamk@tauex.tau.ac.il

Raja Giryes
School of Electrical Engineering, Tel Aviv University, e-mail: raja@tauex.tau.ac.il

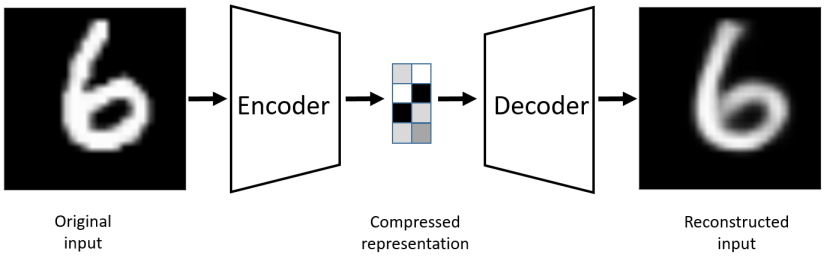


Fig. 1: An autoencoder example. The input image is encoded to a compressed representation and then decoded.

In the most popular form of autoencoders, A and B are neural networks [29]. In the special case that A and B are linear operations, we get a linear autoencoder [2]. In the case of linear autoencoder where we also drop the non-linear operations, the autoencoder would achieve the same latent representation as Principal Component Analysis (PCA) [27]. Therefore, an autoencoder is in fact a generalization of PCA, where instead of finding a low dimensional hyperplane in which the data lies, it is able to learn a non-linear manifold.

Autoencoders may be trained end-to-end or gradually layer by layer. In the latter case, they are "stacked" together, which leads to a deeper encoder. In [24], this is done with convolutional autoencoders, and in [40] with denoising autoencoder (described below).

This chapter is organized as follows. In Section 2, different regularization techniques for autoencoders are considered, whose goal is to ensure that the learned compressed representation is meaningful. In Section 3, the variational autoencoders are presented, which are considered to be the most popular form of autoencoders. Section 4 covers very common applications for autoencoders, and Section 5 describes some recent advanced techniques in this field. Section 6 concludes this chapter.

2 Regularized autoencoders

Since in training, one may just get the identity operator for A and B, which keeps the achieved representation the same as the input, some additional regularization is required. The most common option is to make the dimension of the representation smaller than the input. This way, a $bottleneck$ is imposed. This option also directly serves the goal of getting a low dimensional representation of the data. This representation can be used for purposes such as data compression, feature extraction, etc. Its important to note that even if the $bottleneck$ is comprised of only one node, then overfitting is still possible if the capacity of the encoder and the decoder is large enough to encode each sample to an index.

In cases where the size of the hidden layer is equal or greater than the size of the input, there is a risk that the encoder will simply learn the identity function. To prevent it without creating a bottleneck (i.e. smaller hidden layer) several options exists for regularization, which we describe hereafter, that would enforce the autoencoder to learn a different representation of the input.

An important tradeoff in autoencoders is the bias-variance tradeoff. On the one hand, we want the architecure of the autoencoder to be able to reconstruct the input well (i.e. reduce the reconstruction error). On the other hand, we want the low representation to generalize to a meaningful one. We now turn to describe different methods to tackle such tradeoffs.

2.1 Sparse Autoencoders

One way to deal with this tradeoff is to enforce sparsity on the hidden activations. This can be added on top of the bottleneck enforcement, or instead of it. There are two strategies to enforce the sparsity regularization. They are similar to ordinary regularization, where they are applied on the activations instead of the weights. The first way to do so, is to apply L_1 regularization, which is known to induce sparsity. Thus, the autoencoder optimization objective becomes

$$\arg\min_{A,B} E[\Delta(\mathbf{x}, B \circ A(\mathbf{x})] + \lambda \sum_i |a_i|, \tag{2}$$

where a_i is the activation at the ith hidden layer and i iterates over all the hiddens activations. Another way to do so, is to use the KL-divergence, which is a measure of the distance between two probability distributions. Instead of tweaking the $lambda$ parameter as in the L_1 regularization, we can assume the activation of each neuron act as a Bernouli variable with probability p and tweak that probability. At each batch, the actual probability is then measured, and the difference is calculated and applied as a regularization factor. For each neuron j, the calculated empirical probability is $\hat{p}_j = \frac{1}{m} \sum_i a_i(x)$, where i iterates over the samples in the batch. Thus the overall loss function would be

$$\arg\min_{A,B} E[\Delta(\mathbf{x}, B \circ A(\mathbf{x})] + \sum_j KL(p||\hat{p}_j), \tag{3}$$

where the regularization term in it aims at matching p to $\hat{p}$.

2.2 Denoising Autoencoders

Denoising autoencoders [39] can be viewed either as a regularization option, or as robust autoencoders which can be used for error correction. In these architectures,

the input is disrupted by some noise (e.g., additive white Gaussian noise or erasures using Dropout) and the autoencoder is expected to reconstruct the clean version of the input, as illustrated in Figure 2.

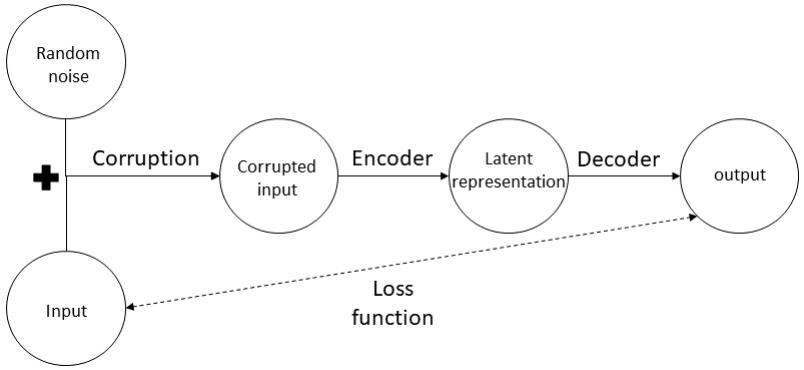


Fig. 2: A denoising autoencoder example. The disrupted input image is encoded to a representation and then decoded.

Note that $\tilde{\mathbf{x}}$ is a random variable, whose distribution is given by $C(\tilde{\mathbf{x}}|\mathbf{x})$. Two common options for C are:

$$C_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\mathbf{x}, \sigma^2 \mathcal{I}), \tag{4}$$

and

$$C_p(\tilde{\mathbf{x}}|\mathbf{x}) = \beta \odot \mathbf{x}, \quad \beta \sim Ber(p), \tag{5}$$

where $\odot$ detnotes an element-wise (Hadamard) product. In the first option, the variance parameter σ sets the impact of the noise. In the second, the parameter p sets the probability of a value in $\mathbf{x}$ not being nullified. A relationship between denoising autoencoders with dropout to analog coding with erasures has been shown in [3].

2.3 Contractive Autoencoders

In denoising autoencoders, the emphasis is on letting the encoder be resistant to some perturbations of the input. In contractive autoencoders, the emphasis is on making the feature extraction less sensitive to small perturbations, by forcing the encoder to disregard changes in the input that are not important for the reconstruction by the decoder. Thus, a penatly is imposed on the Jacobian of the network. The Jacobian matrix of the hidden layer h consists of the derivative of each node h_j with respect to each value x_i in the input x. Formally: $J_{ji} = \nabla_{x_i} h_j(x_i)$. In contractive autoencoders

we try to minimize its L2 norm, such that the overall optimization loss would be:

$$\arg\min_{A,B} E[\Delta(x, B \circ A(x)] + \lambda ||J_A(x)||_2^2. \tag{6}$$

The reconstruction loss function and the regularization loss actually pull the result towards opposite directions. By minimizing the squared Jacobian norm, all the latent representations of the input tend to be more similar to each other, and by thus make the reconstruction more difficult, since the differences between the representations are smaller. The main idea is that variations in the latent representation that are not important for the reconstructions would be diminished by the regularization factor, while important variations would remain because of their impact on the reconstruction error.

3 Variational Autoencoders

A major improvement in the representation capabilities of autoencoders has been achieved by the Variational Autoencoders (VAE) model [20]. Following Variational Bayes (VB) Inference [4], VAE are generative models that attempt to describe data generation through a probabilistic distribution. Specifically, given an observed dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ of V i.i.d samples, we assume a generative model for each datum $\mathbf{x}_i$ conditioned on an unobserved random latent variable $\mathbf{z}_i$, where θ are the parameters governing the generative distribution. This generative model is also equivalent to a *probabilistic decoder*. Symmetrically, we assume an approximate posterior distribution over the latent variable $\mathbf{z}_i$ given a datum $\mathbf{x}_i$ denoted by recognition, which is equivalent a *probabilistic encoder* and governed by the parameters ϕ. Finally, we assume a prior distribution for the latent variables $\mathbf{z}_i$ denoted by $p_\theta(\mathbf{z}_i)$. Figure 3 depicts the relationship described above. The parameters θ and ϕ are unknown and needs to learned from the data. The the observed latent variables $\mathbf{z}_i$ can be interpreted as a *code* given by the *recognition model* $q_\phi(\mathbf{z}|\mathbf{x})$.

The marginal log-likelihood is expressed as a sum over the individual data points $\log p_\theta(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N) = \sum_{i=1}^N \log p_\theta(\mathbf{x}_i)$, and each point can be rewritten as:

$$\log p_\theta(\mathbf{x}_i) = D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}_i) || p_\theta(\mathbf{z}|\mathbf{x}_i)\right) + \mathcal{L}(\theta, \phi; \mathbf{x}_i), \tag{7}$$

where the first term is the Kullback-Leibler divergence of the approximate recognition model from the true posterior and the second term is called the *variational lower bound* on the marginal likelihood defined as:

$$\mathcal{L}(\theta, \phi; \mathbf{x}_i) \triangleq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)}\Big[-\log q_\phi(\mathbf{z}|\mathbf{x}) + \log p_\theta(\mathbf{x}, \mathbf{z})\Big]. \tag{8}$$

Since the Kullback-Leibler divergence is non-negative, $\mathcal{L}(\theta, \phi; \mathbf{x}_i)$ is a lower bound on the marginal log-likelihood and since the the marginal log-likelihood is indepen-

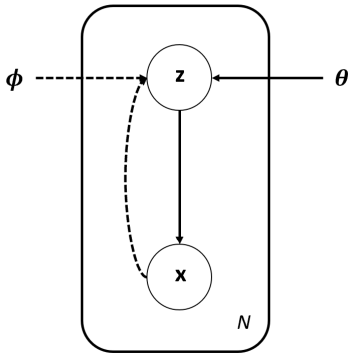


Fig. 3: A Graphical Representatin of VAE

dent of the parameters θ and ϕ, by maximizing it improve our approximation of the posterior with respect to the Kullback-Leibler divergence.

The variational lower bound can be further expanded as follows:

$$\mathcal{L}(\theta, \phi; \mathbf{x}_i) = -D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}_i)||p_\theta(\mathbf{z})\right) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_i)}\Big[\log p_\theta(\mathbf{x}_i|\mathbf{z})\Big] \tag{9}$$

Variational inference follows by maximizing $\mathcal{L}(\theta, \phi; \mathbf{x}_i)$ for all data points with respect to θ and ϕ.

Given a dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ with N data points, we can estimate the marginal likelihood lower-bound of the full dataset $\mathcal{L}(\theta, \phi; \mathbf{X})$ using a mini-batch $\mathbf{X}^M = \{\mathbf{x}_i\}_{i=1}^M$ of size M as follows:

$$\mathcal{L}(\theta, \phi; \mathbf{X}) \approx \tilde{\mathcal{L}}^M(\theta, \phi; \mathbf{X}^M) = \frac{N}{M}\sum_{i=1}^{M}\mathcal{L}(\theta, \phi; \mathbf{x}_i) \tag{10}$$

Classical mean-field VB assumes a factorized approximate posterior followed by a closed form optimization updates (which usually required conjugate priors). However, VAE follows a different path in which the gradients of $\tilde{\mathcal{L}}^M(\theta, \phi; \mathbf{X}^M)$ are approximated using a the reparameterization trick and stochastic gradient optimization.

3.1 The Reparameterization Trick

The reparameterization trick is a simple approach to estimate $\mathcal{L}(\theta, \phi; \mathbf{x}_i)$ based on a small sample of size L. Consider Equation 8, we can reparameterize the random variable $\tilde{\mathbf{z}} \sim q_\phi(\mathbf{z}|\mathbf{x})$ using a differentiable transformation $g_\phi(\epsilon, \mathbf{x})$ using an auxiliary

noise vrabile ϵ drawn from some distribution $\epsilon \sim p(\epsilon)$ [20]. Using this tecnique, $\mathcal{L}(\theta, \phi; \mathbf{x}_i)$ is approximated as follows:

$$\mathcal{L}(\theta, \phi; \mathbf{x}_i) \approx \tilde{\mathcal{L}}(\theta, \phi; \mathbf{x}_i) = \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(\mathbf{x}_i, \mathbf{z}_{(i,l)}) - \log q_\phi(\mathbf{z}_{(i,l)|\mathbf{x}_i}), \tag{11}$$

where $\mathbf{z}_{(i,l)} = g_\phi(\epsilon_{(i,l)}, \mathbf{x}_i)$ and $\epsilon_{(i,l)}$ is a random noise drawn from $\epsilon_l \sim p(\epsilon)$.

Remember we wish to optimize the mini-batch estimates from Equation 10. By plugging Equation 11 we get the following differentiable expression:

$$\hat{\mathcal{L}}^M(\theta, \phi; \mathbf{X}) = \frac{N}{M} \sum_{i=1}^{M} \tilde{\mathcal{L}}(\theta, \phi; \mathbf{x}_i), \tag{12}$$

which can be derived according to θ and ϕ and plugged into an optimizer framework.

Algorithm 1 Pseudo-code for VAE

$(\theta, \phi) \leftarrow$ Initialize Parameter
repeat
  $\mathbf{X}^M \leftarrow$ Random minibatch of M datapoints
  $\epsilon \leftarrow L$ random samples of $p(\epsilon)$
  $\mathbf{g} \leftarrow \nabla_{(\theta,\phi)} \hat{\mathcal{L}}^{\mathbf{M}}(\theta, \phi; \mathbf{X})$ {Gradients of Equation 12}
  $(\theta, \phi) \leftarrow$ Update parameters based on $\mathbf{g}$ {e.g., update with SGD or Adagrad}
until Convergenge of (θ, ϕ)
return (θ, ϕ)

Algorithm 1 summarizes the full optimization procedure for VAE. Often L can be set to 1 so long as M is large enough. Typical numbers are $M = 100$ and $L = 1$.

Equation11 presents a lower bound on the log-likelihood $\log p_\theta(\mathbf{x}_i)$. In [5], the equation is changed to

$$\mathcal{L}(\theta, \phi; \mathbf{x}_i) = \frac{1}{L} \sum_{l=1}^{L} \log \frac{1}{k} \sum_{j=1}^{k} \frac{p_\theta(\mathbf{x}_i, \mathbf{z}_{(j,l)})}{q_\phi(\mathbf{z}_{(j,l)|\mathbf{x}_i})}. \tag{13}$$

Intuitively, instead of taking the gradient of a single randomized latent representation, the gradients of the generative network are learned by a weighted average of the sample over different samples from its (approximated) posterior distribution. The weights simply the likelihood functions $q_\phi(\mathbf{z}_{(j,l)|\mathbf{x}_i})$.

3.2 Example: The Case of Normal Distribution

Usually, we approximate $p(\mathbf{z}|\mathbf{x})$ with a Gaussian distribution $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(g(\mathbf{x}), h(\mathbf{x}))$, where $g(\mathbf{x})$ and $h(\mathbf{x})$ are the mean and the covariance of the distribution defined by

the encoder network. Namely, the encoder takes an input $\mathbf{x}_i$ and maps it into a mean and covariance that determine the approximate posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$.

To enable backpropagation through the network, sampling from $q_\phi(\mathbf{z}|\mathbf{x})$ can simplified using the reparametrisation trick as follows:

$$\mathbf{z} = h(\mathbf{x})\xi + g(\mathbf{x}), \tag{14}$$

where $\xi \sim \mathcal{N}(0, \mathbf{I})$ is a normal distribution.

Finally, we denote the decoder with an additional function f, and require that $x \approx f(\mathbf{z})$. The loss function of the entire network then becomes:

$$loss = c \left\| x - f(\mathbf{z}) \right\|^2 + D_{KL} \left(\mathcal{N}(g(\mathbf{x}), h(\mathbf{x})), \mathcal{N}(0, \mathbf{I}) \right), \tag{15}$$

which can be automatically derived with respect to the network parameters in g, h and f and optimized with backpropogation.

3.3 Disentangled Autoencoders

The variational lower bound as presented at Eq. 9, can be viewed as the summation of two terms: The right term that includes the reconstruction capability of samples, and the left term that acts as a regularization that biases $q_\phi(z|\mathbf{x}^{(i)}$ towards the assumed prior $p_\theta(z)$. Disentangled autoencoders include variational autoencoders with a small addition. They add a parameter β is as a multiplicative factor for the KL divergence [17] at Eq. 9. Its maximization factor is thus:

$$\mathcal{L}(\theta, \phi, \mathbf{x}^{(i)}) = -\beta D_{KL}(q_\phi(z|\mathbf{x}^{(i)})||p_\theta(z)) + E_{q_\phi(z|\mathbf{x}^{(i)})}[\log p_\theta(\mathbf{x}^{(i)}|z)]. \tag{16}$$

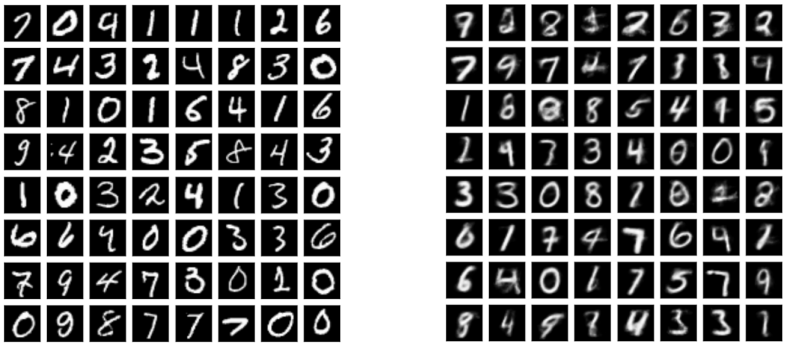
In practice, the prior $p_\theta(z)$ is commonly set as the standard multivariate normal distribution $\mathcal{N}(0, \mathcal{I})$. In those cases, all the features are uncorrelated, and the KL divergence regularizes the latent features distribution $q_\phi(z|\mathbf{x}^{(i)}$ to a less correlated one. Note that the larger the β, the less correlated (more disentangled) the features will be.

4 Applications of autoencoders

Learning a representation via the autoencoder can be used for various applications. The different types of autoencoders may be modified or combined to form new models for various applications. For example, in [28], they are used for classification, captioning, and unsupervised learning. We describe below some of the applications of autoencoders.

4.1 Autoencoders as a generative model

As explained in Section 3, variational autoencoders are generative models that attempt to describe data generation through a probabilistic distribution. Furthermore, as can be seen in Equation 9, the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x}^{(i)}$ which is derived by the encoder, is regularized towards a continuous and complete distribution in the shape of the predefined prior of the latent variables $p_\theta(\mathbf{z})$. Once trained, one can simply samples random variables from the the same prior, and feed it to the decoder. Since the decoder was trained generate $\mathbf{x}$ from $p_\theta(\mathbf{x}_i|\mathbf{z})$, it would generate a meaningful newly-generated sample. In figure 4, original and generated images are displayed over the MNIST dataset. When discussing the generation of new samples, the immediate debate involves the comparison between VAE and GANs. The first is trained (usually) on MSE which yields slightly blurred images, but allows inference over the latent variables in order to control the output. The latter is the latter is trained over the realism of the generated images which gives remarkable results, but in the cost of the control on the resulting images. More on that subject, and a method combine both models, can be found at Section 5.1.



(a) Sample from the original MNIST dataset. (b) VAE generated MNIST images.

Fig. 4: Generated images of from a variational autoencoder, trained on the MNIST dataset with a prior $p_\theta(\mathbf{z}) = \mathcal{N}(0, \mathcal{I})$. Left: original images from the dataset. Right: generated images.

4.2 Use of autoencoders for classification

While autoencoders are being trained in an unsupervised manner (i.e., in the absence of labels), they can be used also in the semi-supervised setting (where part of the data do have labels) for improving classification results. In this case, the encoder is used as a feature extractor and is "plugged" into a classification network. This is

mainly done in the semi-supervised learning setup, where a large dataset is given for a supervised learning task, but only a small portion of it is labeled.

The key assumption is that samples with the same label should correspond to some latent presentation, which can be approximated by the latent layer of autoencoders. First, the autoencoders are trained in an unsupervised way, as described in previous sections. Then (or in parallel), the decoder is put aside, and the encoder is used as the first part of a classification model. Its weights may be fine tuned [9] or stay fixed during training. A simpler strategy can be found in [12], where a support vector machine (SVM) is trained on the output features of the encoder.

Another approach use autoencoders as a regularization technique for a classification network. For example, in [21, 46], two networks are connected to the encoder, a classification network (trained with the labelled data) and the decoder network (trained to reconstruct the data, whether labeled or unlabeled). Having the reconstruction head in addition to the classification head serves a regularizer for the latter. An illustration is given in figure 5.

4.3 Use of autoencoders for clustering

Clustering is an usupervised problem, where the target is to split the data to groups such that sampless in each group are similar to one another, and different from the samples in the other groups. Most of the clustering algorithms are sensitive to the dimensions of the data, and suffer from the curse of dimensionality.

Assuming that the data have some low-dimensional latent representation, one may use autoencoders to calculate such representations for the data, which are composed of much less features. First, the autoencoder is trained as described in the sections before. Then, the decoder is put aside, similarly to the usage in classification. The latent representation (the encoders output) of each data point is then kept, and serves as the input for any given clustering algorithm (e.g., K-means).

The main disadvantage of using vanilla autoencoders for clustering is that the embeddings are trained solely for reconstruction and not for the clustering application. To overcome this, several modifications can be made. In [33], the clustering is done similarly to the K-means algorithm [41], but the embeddings are also retrained at each iteration. In this training an argument is added to the autoencoder loss function, which penalizes the distance between the embedding and the cluster center.

In [14], A prior distribution is made on the embeddings. Then, the optimization is done both by the reconstruction error and by the KL-Divergence between the resulting embeddings distribution and the assumed prior. This can be done implicitly, by training a VAE with the assumed prior. At [6], this is done while assuming a multivariate Gaussian mixture.

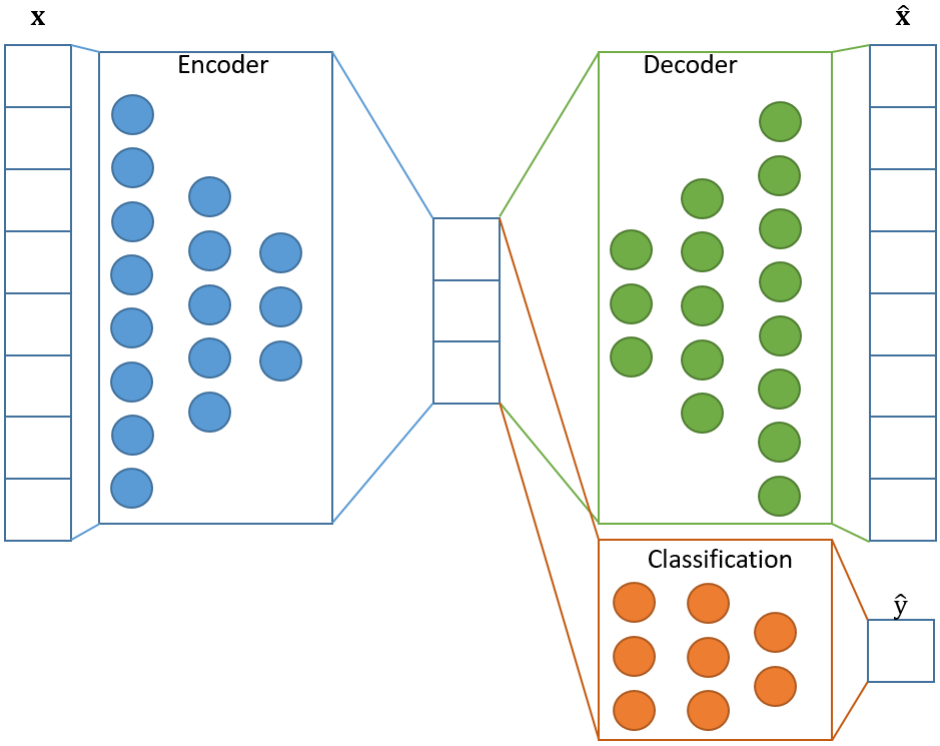


Fig. 5: An illustration for using autoencoders as regularization for supervised models. Given the reconstruction loss $R(x, \hat{x})$, and the classification lost function $\mathcal{L}(y, \hat{y})$, the new loss function would be $\tilde{\mathcal{L}} = \mathcal{L}(y, \hat{y}) + \lambda R(x, \hat{x})$, where λ is the regularization parameter

.

4.4 Use of autoencoders for anomaly detection

Anomaly detection is another unsupervised task, where the objective is to learn a normal profile given only the normal data examples and then identify the samples not conforming to the normal profile as anomalies. This can be applied in different applications such as fraud detection, system monitoring, etc. The use of autoencoders for this tasks, follows the assumption that a trained autoencoder would learn the latent subspace of normal samples. Once trained, it would result with a low reconstruction error for normal samples, and high reconstruction error for anomalies [15, 13, 48, 47].

4.5 Use of autoencoders for recommendation systems

A recommender system, is a model or system that seek to predict users preferences or affinities to items [30]. Recommender systems are prominent in e-commerce websites, application stores, online content providers and have many other commercial applications. A classical approach in recommender system models is Collaborative Filtering (CF) [16]. In CF, user preferences are inferred based on information from other user preferences. The hidden assumption is that the human preferences are highly correlated i.e., people that exhibit similar preferences in the past will exhibit similar preferences in the future.

An basic example of the use of autoencoders for recommender systems is the AutoRec model [32]. The AutoRec model has two variants: user-based AutoRec (U-AutoRec) and item-based AutoRec (I-AutoRec). In U-AutoRec the autoencoder learns a lower dimensional representation of item preferences for specific users while in I-AutoRec, the autoencoder learns a lower dimensional representation of user preferences for specific items.

For example, assume a dataset consisting of M user and N items. Let $\mathbf{r_m} \in \mathcal{R}^{\mathbf{N}}$ be a preference vector for the user m consisting of its preference score to each of the N items. U-AutoReco's decoder is $z = g(\mathbf{r}_m)$ mapping $\mathbf{r}_m$ into representation the representation vector $z \in \mathcal{R}^d$, where $d \ll N$. The reconstruction given the encoder $f(z)$ is $h(\mathbf{r}_m; \theta) = f(g(\mathbf{r}_m))$, where θ are the model's parameters. The U-AutoRec objective is defined as

$$\arg\min_\theta \sum_{m=1}^{M} \|\mathbf{r}_m - h(\mathbf{r}_m; \theta)\|_O^2 + \lambda \cdot reg. \tag{17}$$

Here, $\|\cdot\|_O^2$ means that the loss is defined only on the observed preferences of the user. At prediction time, we can investigate the reconstruction vector and find items that the user is likely to prefer.

The I-AutoRec is defined symetrically as follows: Let $\mathbf{r}_n$ be item n's preference vector for each user. The I-AutoRec objective is defined as

$$\arg\min_\theta \sum_{n=1}^{N} \|\mathbf{r}_n - h(\mathbf{r}_n; \theta)\|_O^2 + \lambda \cdot reg. \tag{18}$$

At prediction time, we reconstruct the preference vector for each item, and look for potential users with high predicted preference.

In [35, 34], the basic AutoRec model was extended by including de-noising techniques and incorporating users and items side information such as user demographics or item descriptoin. The de-noising serve as another type of regularization that prevent the auto-encoder overfitting rare patterns that do not concur with general user preferences. The side information whas shown to improve accuracy and speed-up the training process.

Similar to the original AutoRec, two symetrical models have been proposed, one that works with user preference $\mathbf{r}_m$ vectors and the other with item preference vectors $\mathbf{r}_n$. In the general case, these vectors may consist of explicit ratings. The Collaborative Denoising Auto-Encoder (CDAE) model [45] is essentially applying the same approach on vectors of implicit ratings rather than explicit ratings. Finally, a variational approach have been attempted by applaying VAE in a similar fashion [22].

4.6 Use of autoencoders for dimensionality reduction

Real world data such as text or images is often represented using a sparse high-dimensional representation. While many models and applications work directly in the high dimensional space, this often leads to the *curse of dimensioanlity* [10]. The goal of dimensionality reduction is to learn a a lower dimensional manifold, so-called "intrinsic dimensionality" space.

A classical approach for dimensionality reduction is Principal Component Analysis (PCA) [44]. PCA is a linear projection of data points into a lower dimensional space such that the squared reconstruction loss is minimized. As a linear projection, PCA is optmial. However, non-linear methods such as autoencoders, may and often do achieve superior results.

Other methods for dimensionalty reduction employ different objectives. For example, Linear Discriminant Analysis (LDA) is a supervised method to find a linear subspace, which is optimal for discriminating data from different classes [7]. ISOMAP [36] learns a low dimensional manifold by retaining the geodesic distance between pairwise data in the original space. For a survey of different dimensioanlity methods see [38].

The use of autoencoders for dimensionality reduction is stright forward. In fact, the dimensionality reduction is performed by every autoencoder in the bottleneck layer. The projection of the original input into the lower-dimensional bottleneck representation is a dimension reduction operation through the encoder and under the objective given to the decoder. For example, an autoencoder comrised of a simple fully connected encoder and decoder with a squared loss objective performs dimension reduction with a similar objective to PCA. However, the non-linearity activation functions often allows for a superior reconstruction when compared to simple PCA. More complex architectures and different objectives allow different complex dimension reduction models. To review the different applications of autoencoders for dimension reduction, we erefer the interested reader to [18, 42, 43].

5 Advanced autoencoder techniques

Autoencoders are usually trained by a loss function corresponding to the difference between the input and the output. As shown above, one of the strengths of autoencoders is the ability to use their latent representation for different usages. On the other hand, by looking at the reconstruction quality of autoencoders for images, one of its major weaknesses becomes clear, as the resulting images are usually blurry. The reason for that is the used loss function, which does not take into account how realistic its results are and does not use the prior knowledge that the input images are not blurred. In recent years, there were some developments related to autoencoders, which deal with this weakness.

5.1 Adversarially learned inference

An alternative generative model to autoencoders that synthesize data (such as images) is the Generative Adversarial Networks (GANs). In a nutshell, a GAN architecture consists of two parts: The generator which generates new samples, and a discriminator which is trained to distinguish between real samples, and generated ones. The generator and the discriminator are trained together using a loss function that enforces them to compete with each other, and by thus improves the quality of the generated data. This leads to generated results that are quite compelling visually. Yet, one of the disadvantages of GANs is mode collapse, which unlike autoencoders, may cause them to represent via the latent space just part of the data (miss some modes in its distribution) and not all of it.

In Adversarially Learned Inference (ALI) there is an attempt to merge both ideas and get a compromise of their strengths and weaknesses [8]. Instead of training a VAE with some loss function between the input and the output, a discriminator is used to distinguish between $(\mathbf{x}, \hat{\mathbf{z}})$ pairs, where $\mathbf{x}$ is an input sample and $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$ is sampled from the encoders output, and $(\tilde{\mathbf{x}}, \mathbf{z})$ pairs, where $\mathbf{z} \sim p(\mathbf{z})$ is sampled from the used prior in the VAE, and $\tilde{\mathbf{x}} \sim p(\mathbf{x}|\mathbf{z})$ is the decoders output. This way the decoder is enforced to output realistic results in order to "fool" the discriminator. Yet, the autoencoder structure is maintained. An example of how ALI enables altering specific features in order to get meaningful alterations in images is presented in Figure 6.

5.2 Deep feature consistent variational autoencoder

In this section, a different loss function is presented to optimize the autoencoder. Given an original image and a reconstructed one, instead of measuring some norm on the pixel difference (such as the ℓ_2), a different measure is used that takes into account the correlation between the pixels.

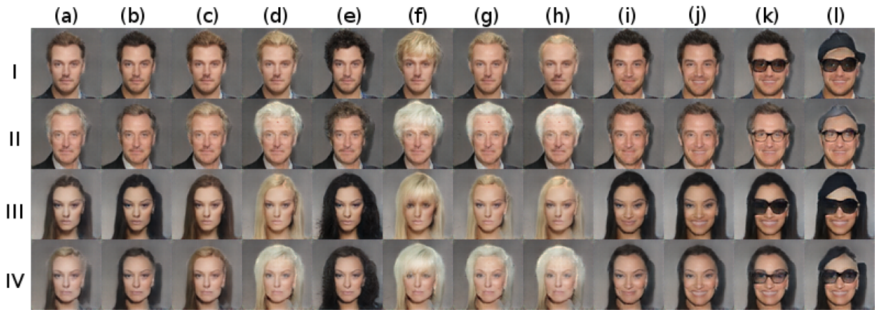


Fig. 6: An Image drawn from [8]. A model is first trained on the CelebA dataset [23]. It includes 40 different attributes on each image, which in ALI are linearly embedded in the encoder, decoder, and discriminator. Following the training phase, a single fixed latent code z is sampled. Each row has a subset of attributes that are held constant across columns. The attributes are male, attractive, young for row I; male attractive, older for row II; female, attractive, young for row III; female, attractive, older for Row IV. Attributes are then varied uniformly over rows across all columns in the following sequence: (b) black hair; (c) brown hair; (d) blond hair; (e) black hair, wavy hair; (f) blond hair, bangs; (g) blond hair, receding hairline; (h) blond hair, balding; (i) black hair, smiling; (j) black hair, smiling, mouth slightly open; (k) black hair, smiling, mouth slightly open, eyeglasses; (l) black hair, smiling, mouth slightly open, eyeglasses, wearing hat.

Pretrained classification networks are commonly used for transfer learning. They allow transcending between different input domains, where the weights of the model, which have been trained for one domain, are fine tuned for the new domain in order to adapt to the changes between the domains. This can be done by training all the models' (pretrained) weights for several epochs, or just the final layers. Another use of pretrained networks is style transfer, where a style of one image is transfered to another image [11], .e.g., causing a regular photo looks like a painting of a given painter (e.g., Van Gogh) while maintaining its content (e.g., keeping the trees, cars, houses, etc. at the same place). In this case, the pretrained networks serve as a loss function.

The same can be done for autencoders. A pretrained network can be used for creating a loss function for autoencoders [19]. After encoding and decoding an image, both the original and reconstructed image are inserted as input to a pretrained network. Assuming the pretrained network results with high accuracy, and the domain which it was trained on is not too different than the one of the autoencoder, then each layer can be seen as a successful feature extractor of the input image. Therefore, instead of measuring the difference between the two images directly, it can be measured between their representation in the network layers. By measuring the difference between the images at different layers in the network imposes a more realistic difference measure for the autoencoder.

5.3 Conditional image generation with PixelCNN decoders

Another alternative proposes a composition between autoencoders and PixelCNN [26]. In PixelCNN [25], the pixels in the image are ordered by some arbitrary order (e.g., top to bottom, left to right, or RGB values). Then the output is formed sequentially where each pixel is a result of both the output of previous pixels, and the input. This strategy takes into account the local spatial statistics of the image, as illustrated in Figure 7. For example, below a background pixel, there is a higher chance to have another background pixel, than the chance of having a foreground pixel. With the use of the spatial ordering (in addition to the input pixel information), the probability of getting a blurred pixel diminishes. In a later development [37], the local statistics was replaced by the usage of an RNN, but the same concept of pixel generation was remained. This concept can be combined with autoencoders by setting the decoder to be structured as a pixelCNN network generating the output image in a sequential order.

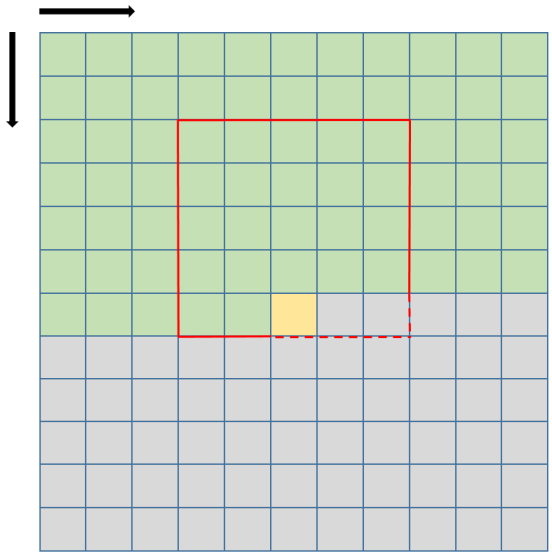


Fig. 7: The pixelCNN generation framework. The pixels are generated sequentially. In this case they are generated from top to bottom and from laft to right. The next pixel to be generated is the yellow one. The green pixels are the already generated ones. For generating the yellow pixel, the pixelRNN takes into account the hidden state, and the information of the green pixels in the red square.

6 Conclusion

This chapter presented autoencoders showing how the naive architectures that were first defined for them evolved to powerful models with the core abilities to learn a meaningful representation of the input and to model generative processes. These two abilities can be easily transformed to various use-cases, where part of them were covered. As explained in Section 5.1, one of the autoencoders fall-backs, is that its reconstruction errors do not include how realistic the outputs are. As for modeling generative processes, despite the success of variational and disentangled autoencoders, the way to choose the size and distribution of the hidden state is still based on experimentation, by considering the reconstruction error, and by varying the hidden state at post training. A future research that better sets these parameters is required.

To conclude, the goal of autoencoders is to get a compressed and meaningful representation. We would like to have a representation that is meaningful to us, and at the same time good for reconstruction. In that trade off, it is important to find the architectures which serves all needs.

References

1. Baldi, P.: Autoencoders, unsupervised learning, and deep architectures. In: I. Guyon, G. Dror, V. Lemaire, G. Taylor, D. Silver (eds.) Proceedings of ICML Workshop on Unsupervised and Transfer Learning, *Proceedings of Machine Learning Research*, vol. 27, pp. 37–49. PMLR, Bellevue, Washington, USA (2012)
2. Baldi, P., Hornik, K.: Neural networks and principal component analysis: Learning from examples without local minima. Neural Netw. **2**(1), 53–58 (1989). DOI 10.1016/0893-6080(89)90014-2. URL `http://dx.doi.org/10.1016/0893-6080(89)90014-2`
3. Bank, D., Giryes, R.: On the relationship between dropout and equiangular tight frames. CoRR **abs/1810.06049** (2018). URL `http://arxiv.org/abs/1810.06049`
4. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg (2006)
5. Burda, Y., Grosse, R.B., Salakhutdinov, R.: Importance weighted autoencoders. CoRR **abs/1509.00519** (2015)
6. Dilokthanakul, N., Mediano, P.A.M., Garnelo, M., Lee, M.C.H., Salimbeni, H., Arulkumaran, K., Shanahan, M.: Deep unsupervised clustering with gaussian mixture variational autoencoders. ArXiv **abs/1611.02648** (2017)
7. Duda, R.O., Hart, P.E., Stork, D.G., et al.: Pattern classification. International Journal of Computational Intelligence and Applications **1**, 335–339 (2001)
8. Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O., Courville, A.C.: Adversarially learned inference. ArXiv **abs/1606.00704** (2016)
9. Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S.: Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. **11**, 625âĂŞ660 (2010)
10. Friedman, J.H.: On bias, variance, 0/1âĂŤloss, and the curse-of-dimensionality. Data mining and knowledge discovery **1**(1), 55–77 (1997)
11. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 2414–2423 (2016)

12. Gogoi, M., Begum, S.A.: Image classification using deep autoencoders. In: 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 1–5 (2017). DOI 10.1109/ICCIC.2017.8524276
13. Gong, D., Liu, L., Le, V., Saha, B., Mansour, M.R., Venkatesh, S., van den Hengel, A.: Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection (2019)
14. Guo, X., Liu, X., Zhu, E., Yin, J.: Deep clustering with convolutional autoencoders. pp. 373–382 (2017)
15. Hasan, M., Choi, J., Neumann, J., Roy-Chowdhury, A.K., Davis, L.S.: Learning temporal regularity in video sequences. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 733–742 (2016)
16. Herlocker, J.L., Konstan, J.A., Riedl, J.: Explaining Collaborative Filtering Recommendations. In: Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, CSCW '00, pp. 241–250. ACM (2000)
17. Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M.M., Mohamed, S., Lerchner, A.: beta-vae: Learning basic visual concepts with a constrained variational framework. In: ICLR (2017)
18. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. science **313**(5786), 504–507 (2006)
19. Hou, X., Shen, L., Sun, K., Qiu, G.: Deep feature consistent variational autoencoder. CoRR **abs/1610.00291** (2016). URL http://arxiv.org/abs/1610.00291
20. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. CoRR **abs/1312.6114** (2013)
21. Le, L., Patterson, A., White, M.: Supervised autoencoders: Improving generalization performance with unsupervised regularizers. In: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (eds.) Advances in Neural Information Processing Systems 31, pp. 107–117. Curran Associates, Inc. (2018). URL http://papers.nips.cc/paper/7296-supervised-autoencoders-improving-generalization-performance-with-unsupervised-regularizers.pdf
22. Liang, D., Krishnana, R.G., Hoffman, M.D., Jebara, T.: Variational autoencoders for collaborative filtering. CoRR **abs/1802.05814** (2018). URL https://arxiv.org/abs/1802.05814
23. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ICCV âĂŹ15, p. 3730âĂŞ3738. IEEE Computer Society, USA (2015). DOI 10.1109/ICCV.2015.425. URL https://doi.org/10.1109/ICCV.2015.425
24. Masci, J., Meier, U., Cireşan, D., Schmidhuber, J.: 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, pp. 52–59. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
25. van den Oord, A., Kalchbrenner, N., Kavukcuoglu, K.: Pixel recurrent neural networks (2016)
26. Oord, A.v.d., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., Kavukcuoglu, K.: Conditional image generation with pixelcnn decoders. In: Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, pp. 4797–4805. Curran Associates Inc., USA (2016). URL http://dl.acm.org/citation.cfm?id=3157382.3157633
27. Plaut, E.: From principal subspaces to principal components with linear autoencoders (2018)
28. Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., Carin, L.: Variational autoencoder for deep learning of images, labels and captions. In: Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, pp. 2352–2360 (2016)
29. Ranzato, M., Huang, F.J., Boureau, Y., LeCun, Y.: Unsupervised learning of invariant feature hierarchies with applications to object recognition. In: 2007 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2007). DOI 10.1109/CVPR.2007.383157
30. Ricci, F., Rokach, L., Shapira, B.: Introduction to recommender systems handbook. In: Recommender systems handbook, pp. 1–35. Springer (2011)

31. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chap. Learning Internal Representations by Error Propagation, pp. 318–362. MIT Press, Cambridge, MA, USA (1986). URL http://dl.acm.org/citation.cfm?id=104279.104293
32. Sedhain, S., Menon, A.K., Sanner, S., Xie, L.: Autorec: Autoencoders meet collaborative filtering. In: Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015, Florence, Italy, May 18-22, 2015 - Companion Volume, pp. 111–112 (2015)
33. Song, C., Liu, F., Huang, Y., Wang, L., Tan, T.: Auto-encoder based data clustering. In: J. Ruiz-Shulcloper, G. Sanniti di Baja (eds.) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, pp. 117–124. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
34. Strub, F., Mary, J.: Collaborative Filtering with Stacked Denoising AutoEncoders and Sparse Inputs. In: NIPS Workshop on Machine Learning for eCommerce. Montreal, Canada (2015)
35. Strub, F., Mary, J., Gaudel, R.: Hybrid recommender system based on autoencoders. CoRR **abs/1606.07659** (2016). URL http://arxiv.org/abs/1606.07659
36. Tenenbaum, J.B., De Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. science **290**(5500), 2319–2323 (2000)
37. Van Den Oord, A., Kalchbrenner, N., Kavukcuoglu, K.: Pixel recurrent neural networks. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICMLâĂŹ16, p. 1747âĂŞ1756. JMLR.org (2016)
38. Van Der Maaten, L., Postma, E., Van den Herik, J.: Dimensionality reduction: a comparative review. J Mach Learn Res **10**, 66–71 (2009)
39. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th International Conference on Machine Learning, ICML '08, pp. 1096–1103. ACM, New York, NY, USA (2008). DOI 10.1145/1390156.1390294. URL http://doi.acm.org/10.1145/1390156.1390294
40. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. **11**, 3371–3408 (2010). URL http://dl.acm.org/citation.cfm?id=1756006.1953039
41. Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S.: Constrained k-means clustering with background knowledge. In: Proceedings of the Eighteenth International Conference on Machine Learning, ICML âĂŹ01, p. 577âĂŞ584. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
42. Wang, W., Huang, Y., Wang, Y., Wang, L.: Generalized autoencoder: A neural network framework for dimensionality reduction. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp. 490–497 (2014)
43. Wang, Y., Yao, H., Zhao, S.: Auto-encoder based dimensionality reduction. Neurocomputing **184**, 232–242 (2016)
44. Wold, S., Esbensen, K., Geladi, P.: Principal component analysis. Chemometrics and intelligent laboratory systems **2**(1-3), 37–52 (1987)
45. Wu, Y., DuBois, C., Zheng, A.X., Ester, M.: Collaborative denoising auto-encoders for top-n recommender systems. In: Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, San Francisco, CA, USA, February 22-25, 2016, pp. 153–162 (2018)
46. Zhang, Y., Lee, K., Lee, H.: Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICMLâĂŹ16, p. 612âĂŞ621. JMLR.org (2016)
47. Zhao, Y., Deng, B., Shen, C., Liu, Y., Lu, H., Hua, X.S.: Spatio-temporal autoencoder for video anomaly detection. In: Proceedings of the 25th ACM International Conference on Multimedia, MM âĂŹ17, p. 1933âĂŞ1941. Association for Computing Machinery, New York, NY, USA (2017). DOI 10.1145/3123266.3123451. URL https://doi.org/10.1145/3123266.3123451
48. Zong, B., Song, Q., Min, M.R., Cheng, W., Lumezanu, C., ki Cho, D., Chen, H.: Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In: ICLR (2018)