Litterature review - Autoencoders

Département de mathématiques et de statistique - Université Laval

Stéphane Caron

22 January, 2020

Contents

1 Introduction to autoencoders

An autoencoder is a neural network technique that aims to learn an input and output it back as its output (Goodfellow, Bengio, and Courville 2016). To achieve this objective, the autoencoder has 2 components: an **encoder** and a **decoder**. The encoder receives an input x and and converts it to a hidden representation z. The decoder receives a representation z and decodes it back to retrieve as much as possible the input x. Historically, autoencoders were known as a dimensionnalty reduction method, but it has now more applications by learning latent variables useful in generative modelling and other fields.

1.1 Architecture

As explained in the introduction, autoencoders are divided in two parts: the encoder and the decoder. The encoder compresses the information into a given representation and the decoder decompresses it back to its original format (see figure 1).

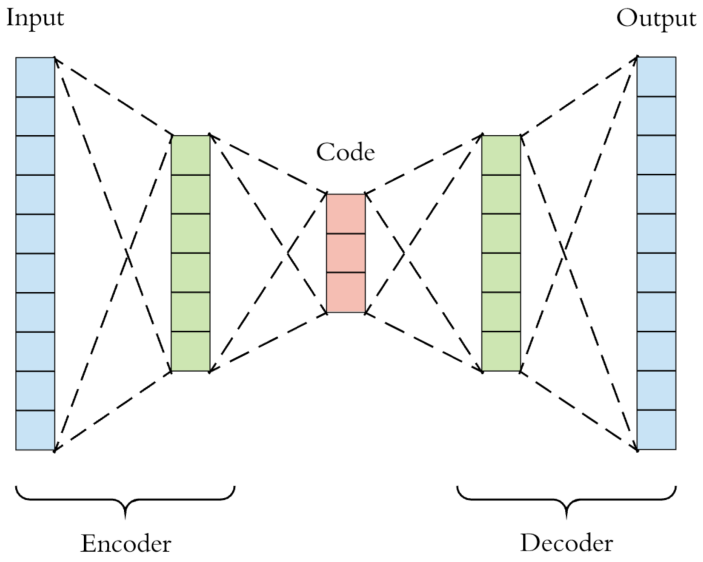

Figure 1: Autoencoders structure example.

The hidden representation (or latent representation) is often smaller because of the structure of the layers between the input and that representation. Autoencoders that have a smaller hidden representation than its inputs are called *undercomplete*. In those cases, the constraints on z force the network to learn the more important features. It could be applied to feedforward networks or to convolutional networks. We are typically interested in that hidden representation, in comparison to the output of the decoder (the reconstructured input) which may sound useless.

1.2 Optimization

Autoencoders intuition is to build back the input x by passing through the 2 components (encoder and decoder). As such, this kind of model does not need any target, so we say it is an unsupervised method. The training of the parameters is mainly done by minimizing the reconstruction error. The loss is given by:

$$L(x, p_\phi\{q_\theta(x)\})$$

where $q(x, \theta)$ is the encoder and $p(z, \phi)$ is the decoder function. The loss generally includes another component $\Omega(x)$ that acts as a regularizer (we'll see in the next sections). The regularization is often essential to prevent the network to "copy/paste" examples of the training set. Those situations can occur even more when the hidden representation is equal or greater than the inputs (*overcomplete*). The minimization of this loss function is done by gradient descent. For instance, the encoder parameters are gradually updated by:

$$\theta \leftarrow \theta - \epsilon * \frac{\partial L}{\partial \theta} \tag{1}$$

where ϵ is a learning weight that aims to control the size of the learning steps.

1.3 Example

Let's illustrate the high-level concepts presented in the last 2 sections with an overly simplistic example. We have a input X with $p = 2$ features and 10 observations. We want to encode those 2 features in one single features using an basic autoencoder with 1 hidden layer. In summary, our autoencoder has 3 layers (input

layer, hidden layer and output layer). We also choose a sigmoid activation function for our hidden layer and a linear activation function for our ouput layer (those choices are arbitrary).

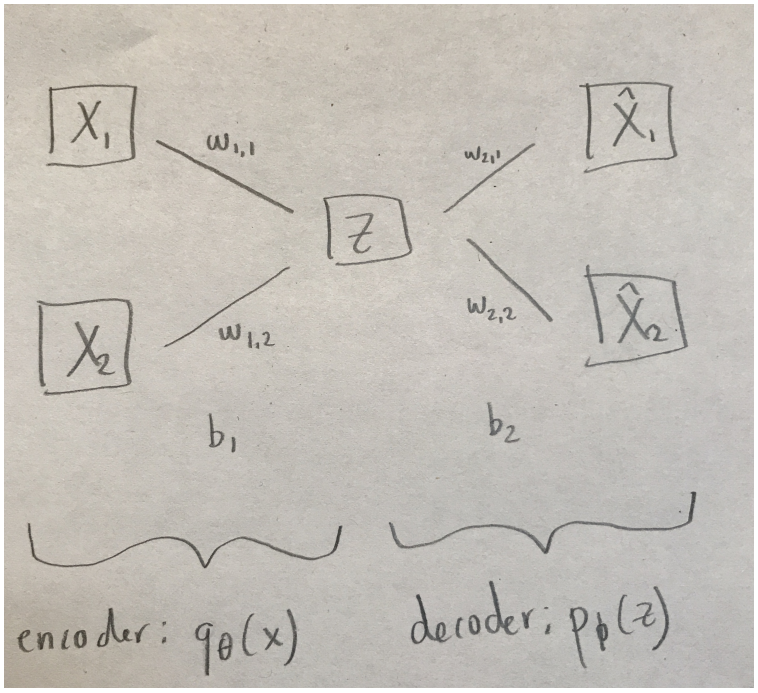

Figure 2: Autoencoders structure example.

The encoder function (that transforms x to z) could be expressed that way:

$$q_\theta(x) = h\big(x_1 * w_{1,1} + x_2 * w_{1,2} + b_1\big)$$

where $h(x)$ is a sigmoid function in the form of $h(x) = \frac{1}{1+e^{-x}}$. The parameters θ that need to be optimized are $w_{1,1}, w_{1,2}$ and b_1. We can also use the matrix notation :

$$q_\theta(X) = h\big(X * \boldsymbol{w}_1 + b_1\big)$$

In the same fashion, the decoder could be expressed as:

$$p_\phi(Z) = \begin{cases} p^1_\phi(z) = f(z * w_{2,1} + b_2) \\ p^2_\phi(z) = f(z * w_{2,2} + b_2) \end{cases}$$

where $f(x)$ is a linear function in the form of $f(x) = x$. The parameters ϕ that need to be optimized are $w_{2,1}, w_{2,2}$ and b_2. We can also use the matrix notation (which is more convinient in that case):

$$p_\phi(Z) = h\big(Z * \boldsymbol{w}_2 + b_2\big) = \hat{X}$$
$$p_\phi : \mathbb{R} \to \mathbb{R}^2$$

To optimize those parameters θ and ϕ, we need to define a loss function (see equation 2). Let's say we want to minimize the reconstruction error only, we can define our loss as:

$$
\begin{aligned}
L(x, \hat{x}) &= L\big(x, p_\phi\{q_\theta(x)\}\big) \\
&= \frac{1}{2}\sum_{i=1}^{n}[x^{(i)} - p_\phi\{q_\theta(x^{(i)}\}]^{\mathrm{T}}[x^{(i)} - p_\phi\{q_\theta(x^{(i)}\}] \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p}\big(x_j^{(i)} - p_\phi^j\{q_\theta(x_j^{(i)})\}\big)^2
\end{aligned}
\tag{2}
$$

where i stands for the i-th observation and j for the j-th element of the input X.

Once we have a loss function, we can optimize the parameters of the encoder/decoder using the *chain rule* of derivatives for each layer (see equation 1).

2 Types of autoencoders

In the previous sections, we introduced the general idea behind the autoencoders. However, there are several types of autoencoders and each has their own strengths in different areas. They differ in their architecture and their optimization process, but at the end, they all consist in unsupervised methods that learn latent structure of the data.

2.1 Origins of autoencoders

The origins of autoencoders were known in the 80s and introduced by Hinton and the PDP Lab in Stanford (Rumelhart, Hinton, and Williams 1986). In this paper, they essentially tried to learn a hidden representation using the input as the output of the model.

2.2 Deep autoencoders

Once the fundamental idea of autoencoders has been developped, a new generation of autoencoders came up more recently and where multiples hidden layers were stacked and trained bottom up to form what we call "deep autoencoders" (Hinton and Salakhutdinov 2006). In the following years, many variants have been applied to autoencoders, mainly regarding the regularization component, to be able to generalize better and better and learn useful representation and structure behind the data.

2.3 Sparse autoencoders

One way of learning useful representations using autoencoder is to impose a constraint of **sparsity** in the hidden representation (Ranzato et al. 2007). A sparse autoencoder is basically an autoencoder that has a sparsity penalty in the loss criterion:

$$
L(x, p_\phi\{q_\theta(x)\}) + \Omega(q_\theta(x))
$$

where z is the hidden representation (or the output of the encoder). That sparsity penalty ($\Omega(z)$) will pull some neurons output values towards zero in the hidden layer, making them "inactive" (Ng 2011). That regularization will prevent overfitting and prevent the network to learn by heart every input.

Sparse autoencoders are generally used to learn features for another task (e.g classification). In that sense, it can accomplish the feature engineering of a task by learning useful features.

2.4 Denoising autoencoders

Another kind of autoencoder that has been widely used are the denoising autocoders (Vincent et al. 2008). These autoencoders are really close to "vanilla deep autoencoders" except they receive a corrupted input and afterwards are trained to produce an uncorrupted output.

The denoising autoencoders will then minimize:

$$L(x, p_\phi\{q_\theta(\tilde{x})\})$$

where $\tilde{x}$ is a copy of x that has been corrupted.

There are different ways of corrupting the input. One way could be to choose a given proportion ν of inputs to randomly "destruct" by assigning a 0-value while keeping the other inputs untouched. That approach is called *masking noise*. We could also corrupt the data by adding *Gaussian noise* or setting randomly a proportion of the inputs to maximum or minimum value (*salt-and-pepper noise)*. The intuition behind this approach is again to prevent the algorithm from learning the useless identity function.

2.5 Contractive autoencoders

The contractive autoencoders (Rifai et al. 2011) are often associated with the denoising autoencoders. The loss criterion includes a regularizer penalty in the form of:

$$\Omega = \lambda \left|\left| \frac{\partial q_\theta(x)}{\partial x} \right|\right|_F^2$$

where F stands for Frobenius norm (sum of squared elements). In this case, we are adding a penalty corresponding to the Jacobian matrix of the partial derivatives of the encoder function.

Denoising autoencoders are built to make the reconstruction function resist small but finite-size perturbations of the inputs, while contractive autoencoders make the feature extraction function (the encoder) resist small changes in the inputs.

2.6 Variational autoencoders

The variational autoencoders (Kingma and Welling 2013) have a slightly different approach than other kinds of autoencoders and are particularly useful in generative modelling and reinforcement learning. Again, these autoencoders have a loss criterion in the form of:

$$L = \text{reconstruction error} + \text{regularizer}$$

However, instead of encoding a hidden representation of size n, it outputs two vectors of size n: a vector of means $\boldsymbol{\mu}$ and a vector of standard deviations $\boldsymbol{\sigma}$. Those vectors allow to sample from Gaussian distributions, which gives the variational autoencoders the unique property of having a latent space that is **continuous** (Shafkat 2018). This is the main difference with previous seen autoencoders. In other words, the basic autoencoders learn a reprensetation that "points" somewhere in the latent space, while varitional autoencoders learn a representation that points to an "area", an area that is defined by the mean $\boldsymbol{\mu}$ and the standard deviation $\boldsymbol{\sigma}$ of the latent space. That property is actually very useful in generative modelling. The figure 3 illustrates the basic structure of a variational autoencoder. In the figure, we can see that the input data first pass through dense layers (fully-connected or convolutional). At some point near the encoded representation, the layers are split into 2 components (μ and σ). The "sample 30" layer will sample a latent representation

using those μ and σ. Once we have the sampled representation, it is decoded back to the same size as the input.

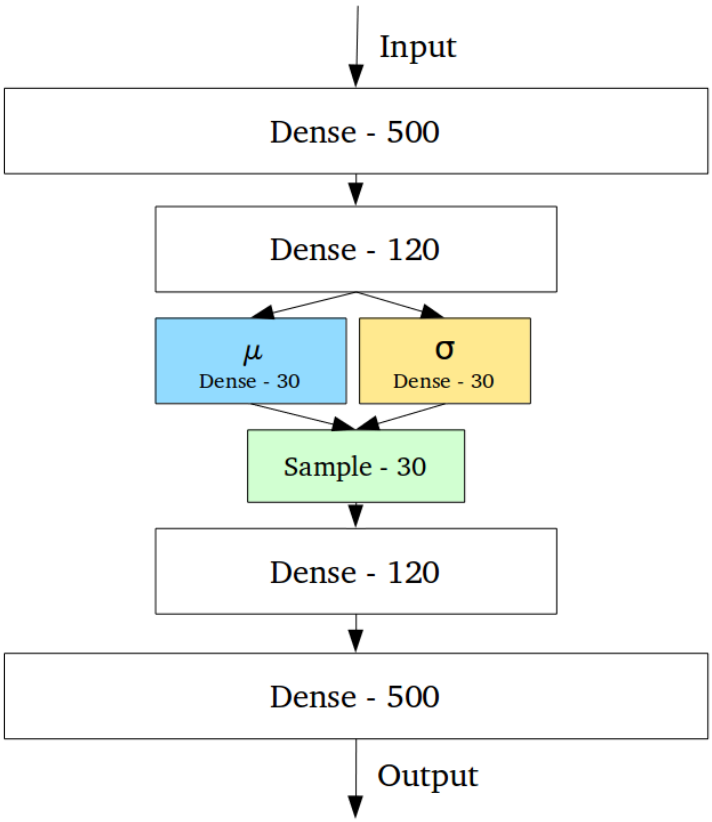

Figure 3: Variational autoencoder structure. Instead of learning a representation directly, it learns the parameters of Gaussian distributions and the hidden representation is given by sampling from those distributions.

The explicit variational autoencoder loss criterion is given by :

$$L(x, p_\phi\{q_\theta(x)\}) + D_{KL}\big[q_\theta(z|x)||p(z)\big]$$

where D_{KL} is the Kullback-Leibler divergence. Its goal is to ensure that the encoded distribution $q_\theta(x)$ and a target distribution $p(z)$ are similar. The function $p(z)$ is a Gaussian distribution $N(0, I)$. In this kind of autoencoder, the hidden representation is stochastic because it is coming from a probability distribution. In order to simplify the derivatives in the backpropagation, we do a clever trick called the "reparametrization trick". In fact, it is possible for some distribution (such as the Gaussian), to separate the parameters (μ and σ) from the stochasticity. Concretely, we can express a normally-distributed variable as:

$$z = \mu + \sigma \odot \epsilon$$

where $\epsilon \sim N(0, 1)$. In brief, that means that the "sample 30" layer in the figure above, is generated from the 2 parameters layers μ and σ and a normal sample. The backpropagation then ignores the stochastic component, and derivates the parameters layers only, which simplifies at lot the optimization process.

2.7 Adversarial autoencoders

Adversarial autoencoders (Makhzani et al. 2015) are really close to the variational autoencoders in the sense that they also produce a continuous latent space by learning a certain distribution. However, this prior distribution is set in advance and learned in an adversarial fashion instead of being learned by a divergence crtierion. In other words, we replace the KL divergence by a generative adversarial network (GAN) That's mainly why we say that adversarial autoencoders are the combination of variational entoencoders and GAN. The figure 4 shows the architecture of an adversarial autoencoder. As (Makhzani et al. 2015) described in his work, the upper portion of the architecture corresponds to the standard autoencoder that reconstruct the input x from a latent representation z. The bottom portion is a second network trained to predict whether a sample arise from the hidden representation or from a distrubution decided in advance. As such, the autoencoder will try to learn the prior distribution and fool the model at the bottom. That kind of approach is often related with generative modeling.

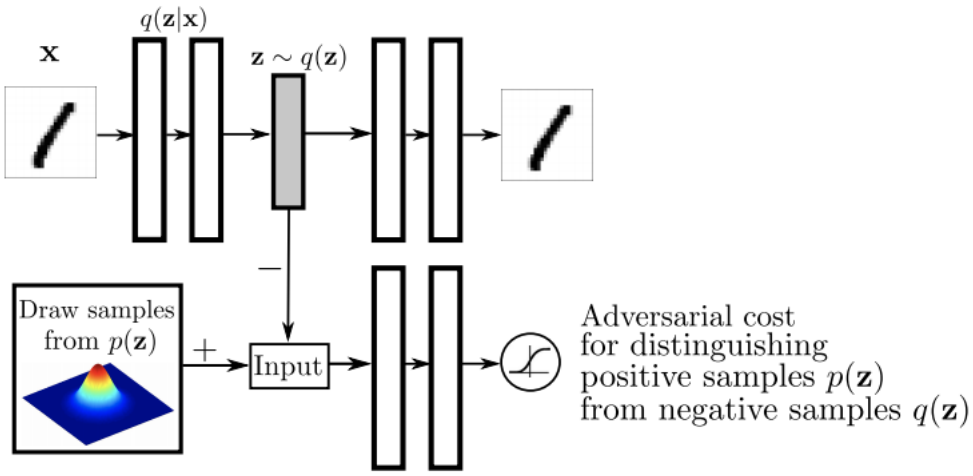

Figure 4: Basic architecture of a adversarial autoencoder taken from (Makhzani et al. 2015).

Like other autoencoders, adversarial autoencoders learn an encoding distribution $q_\theta(z|x)$. In order to do it, we impose a certain distribution on the hidden representation that we express as $p(z)$. If we express the data distribution as $p_d(x)$, we can express the posterior distribution of z as :

$$q(z) = \int_x q_\theta(z|x)p_d(x)dx$$

In the training process, the adversarial autoencoder is trying to match the $q(z)$ and the $p(z)$ distributions. There are several choices for the prior distribution from which to learn:

- Gaussian distribution
- Gamma distribution
- etc

Adversarial autoencoders are generative models that could be used to generate new data, or also to do semi-supervised learning or clustering.

3 One-class classification

In one-class classification problems, one class (which is usually refered as positive or target class in the litterature) is well characterized by instances in the training dataset while the other classes (negatives or outliers) have either no instances or very few of them (Khan and Madden 2013). In in nutshell, the objective is generally to build a subspace that contains as much positive examples as possible and excludes the negative ones, without knowing the targets. This task is often used to identify or remove small number of anomalies or outliers within a dataset. We can distinguish 3 broad categories: density estimation, boundary methods and reconstruction methods.

3.1 Density estimation

Gaussian and mixture of Gaussians are 2 well known density estimation methods. These methods assume the data is distributed according to the normal distribution or to a mixture of normals. The parameters of these normal distributions can be found iteratively with EM algorithm. As soon as the density are estimated, we can use a certain threshold to determine what should be considered as a positive or inlier observations.

Other density estimation alternatives could be the Parzen and Naive Parzen methods. Those nonparametric methods do not make any assumptions about the data distribution. In fact, those methods do not need any parameters estimations.

3.2 Boundary methods

A well known boundary method is the one-class SVM or more generally the Support Vector data description method (SVDD). In this case, we want to define a hypersphere with minimal volume covering the majority of probability mass of the data. The number of parameters to be estimated is equal to the size of the training dataset, which makes this method less effective when dealing with large dataset. One usefull feature of one-class SVM is that we can leverage kernel functions, which allow us to represent complexe subspaces.

3.3 Reconstruction methods

Reconstruction methods are generally associated with matrix factorization, which has become popular in the recent years. In brief, these methods provide a reconstruction matrix $\hat{X}$ of the input X and use a certain norm measure to determine which observations of X are anomalous.

Principal component analysis (PCA) could be considered as a reconstruction anomaly detection method. This method basically finds the directions that maximizes variances in the data. The projection of a matrix $X_{n\times p}$ into a lower dimensions matrix $T_{n\times d}$ (where $d < p$) could be expressed as:

$$T_d = XW_d$$

where W_d is the d first columns of the eigenvectors matrix of X^TX. The reconstructed matrix $\hat{X_d}$ could then be retrieved with $T_dW_d^T$. To identify anomalous data, we could look at observations of this reconstructed matrix that are far from the initial X matrix. However, PCA is highly sensitive to data perturbations, as one anomaly can increase a lot the variance within one component and completely changes the projection directions which can often hide anomalies.

To tackle the problem just mentionned, other versions of PCA were developped such as Robust PCA. This method use a more clever way of decomposing the data to isolate the noise. However, this method is still doing a linear decomposition of the data.

4 Applications of autoencoders

This section aims to survey the different known applications of autoencoders in the litterature and across different industries.

4.1 Dimensionality reduction

One of the first applications for which the autoencoders were used was associated with **dimensionality reduction**. For example, (Hinton and Salakhutdinov 2006) showed that a **deep autoencoder** could learn to represent data in a smaller representation better than the widely used PCA method (see figure 5).

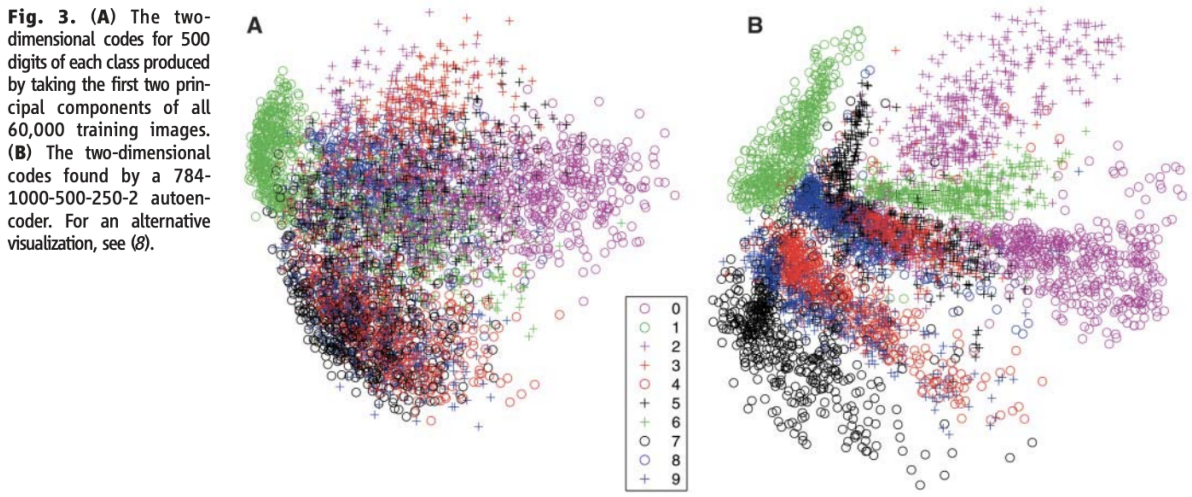

Fig. 3. (A) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. **(B)** The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).

Figure 5: Comparison between the learned representation of a PCA and a deep autoencoder.

4.2 Information retrieval

Another application of autoencoders is related with **information retrieval**. This is the task of finding an entry similar to another entry in a database. It can be done by dimensionnality reduction, and even binarization of an input, which is called semantic hashing. This has been applied to both text (Salakhutdinov and Hinton 2009) and images (Weiss, Torralba, and Fergus 2008).

4.3 Anomaly detection

A task that is often related with unsupervised methods is the idea of finding outliers or anomalies in a dataset. In fact, that's normally an information we don't have and that we can't train on it. Thus, we generally need unsupervised ways of discovering those anomalies. Anomalies can often be interesting from a business perspective when we consider they could be related with examples such as: frauds, medical problems, structural defects, malfunctioning equipment, etc.

For instance, (Zhou and Paffenroth 2017) used **deep autoencoders** to build an unsupervised anomaly detection algorithm called *robust deep autoencoders*. They applied their methodology, that includes an anomaly regularizing penalty, on MNIST dataset to find anomalies in hand written digits. Their intuition is that they can isolate the noise and outliers in the input, and the autoencoder is trained after this isolation. Their model is inspired by Robust Principal Component Analysis where the input X is decomposed in 2

parts: $L_D + S$, where L_D can be effectively reconstructed by an autoencoder and S contains the noise and the outliers in the original data.

Another interesting example of anomaly detection using autoencoders was made by (Zong et al. 2018). They used **deep autoencoders** to build a small representation and use that representation as well as the reconstruction error to train a Gaussian Mixture Model (GMM). The figure 6 shows the overview of the compression (deep autoencoder) and estimation (GMM) tasks.

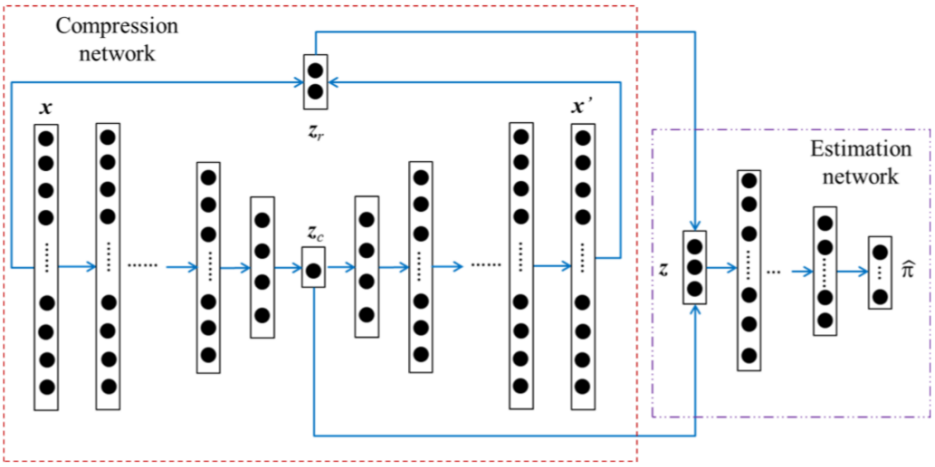

Figure 2: An overview on Deep Autoencoding Gaussian Mixture Model

Figure 6: Overview of the Deep Autoencoder and Gaussian Mixture Model.

In the figure above, the estimation network estimates the GMM parameters without using EM algorithm.

Autoencoders and density estimation are often used togheter in anomaly detection problems. It is the case in one-class learning problem such as (Cao, Nicolau, and Mcdermott 2016). Their work showed that it's possible to use the power of **deep autoencoders** to learn a small compact representation and then use density estimation techniques such as Centroid or KDD that work best in low-dimensionnal situations.

In (Chalapathy, Menon, and Chawla 2018) the authors proposed an anomaly detection framework where the representation learned by the autoencoder is oriented to the task of detecting anomaly. In fact, they first train an autoencoder and then reuse the encoder part to output a compressed representation of the input that will be used as input for a one-class neural network. The encoder parameters are then adjusted based on the error made by the one-class neural network. This one-class model is a basic feed-forward network that replicates the optimization of a one-class SVM. In a nutshell, they leverage the capacity of deep neural networks to extract rich representation of the data along with a one-class objective.

4.4 Clustering

A well known task in the unsupervised world is the task of clustering. (Dilokthanakul et al. 2016) used **varitional autoencoders** and introduced a Gaussian Mixture Model (GMM) as a prior distribution with the goal of performing clustering. They also discuss the problem of over-regularisation than varitional autoencoders can have and how to tackle this problem.

(Xie, Girshick, and Farhadi 2016) also proposed a deep embedding approach that simultaneously learns feature reprensentations and cluster assignements. Their methodology consists of first initializing a **denoising autoencoder**. Once that first model is trained, they are able to represent the data is a smaller representation

(they drop the decoder). They also initialize clusters assignements by using this representation and the k-means algorithm. Then, they simultanuasly train the k cluster centers in the feature space Z and the parameters θ that maps data points to Z. The figure 7 illustrates the network structure where we can see the autoencoder at the top (the learned encoder in gray dashed rectangle), and then the *deep embedded clustering* network at the bottom.

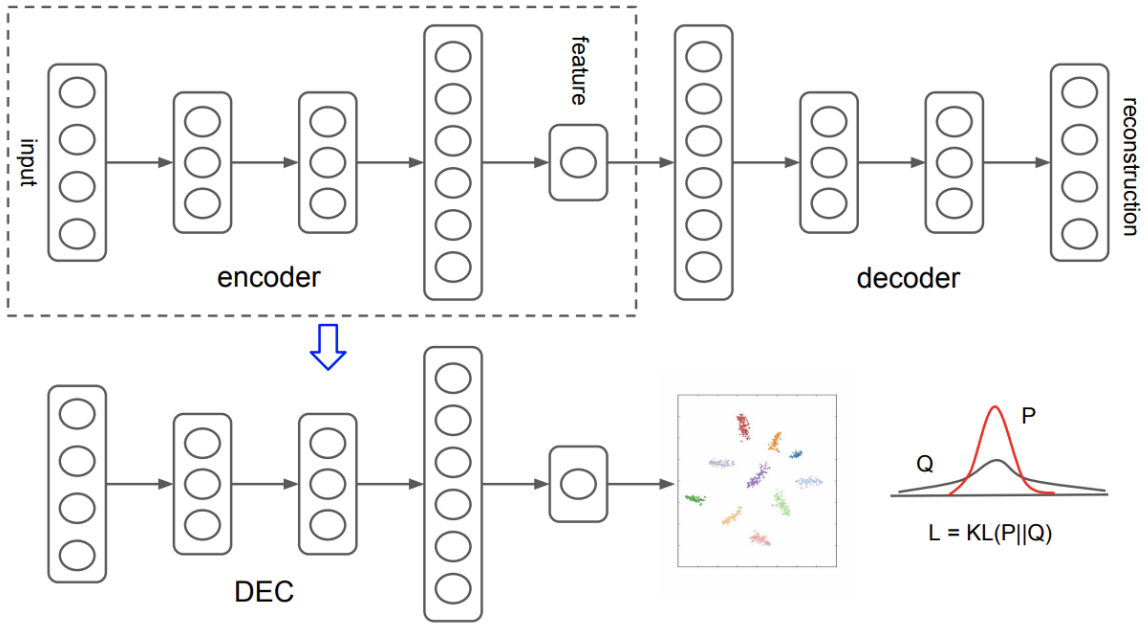

Figure 7: Deep Embedded Clustering network strcuture

(Peng et al. 2018) also used autoencoders to make clustering on a complex space where the non-linearity of neural networks allowed them to learn complex subspaces.

Multivariate density estimation is widely performed for fundamental unsupervised tasks such a clustering. (Takahashi et al. 2018) used **variational autoencoders** and applied a Student-t distribution instead of a Gaussian distribution for the decoder.

References

Cao, Van Loi, Miguel Nicolau, and James Mcdermott. 2016. "A Hybrid Autoencoder and Density Estimation Model for Anomaly Detection." In, 9921:717–26. https://doi.org/10.1007/978-3-319-45823-6_67.

Chalapathy, Raghavendra, Aditya Krishna Menon, and Sanjay Chawla. 2018. "Anomaly Detection Using One-Class Neural Networks." *CoRR* abs/1802.06360. http://arxiv.org/abs/1802.06360.

Dilokthanakul, Nat, Pedro A. M. Mediano, Marta Garnelo, Matthew C. H. Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. 2016. "Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders." *CoRR* abs/1611.02648. http://arxiv.org/abs/1611.02648.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Hinton, G E, and R R Salakhutdinov. 2006. "Reducing the Dimensionality of Data with Neural Networks." *Science* 313 (5786): 504–7. https://doi.org/10.1126/science.1127647.

Khan, Shehroz S., and Michael G. Madden. 2013. "One-Class Classification: Taxonomy of Study and Review of Techniques." *CoRR* abs/1312.0049. http://arxiv.org/abs/1312.0049.

Kingma, Diederik P, and Max Welling. 2013. "Auto-Encoding Variational Bayes." http://arxiv.org/abs/1312.6114.

Makhzani, Alireza, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. 2015. "Adversarial Autoencoders." *CoRR* abs/1511.05644. http://arxiv.org/abs/1511.05644.

Ng, Andrew. 2011. "CS294A Lecture Notes, Sparse Autoencoders." Stanford.

Peng, Xi, Jiashi Feng, Shijie Xiao, Wei-Yun Yau, Joey Zhou, and Songfan Yang. 2018. "Structured Autoencoders for Subspace Clustering." *IEEE Transactions on Image Processing* PP (June): 1–1. https://doi.org/10.1109/TIP.2018.2848470.

Ranzato, Marc Aurelio, Christopher Poultney, Sumit Chopra, and Yann LeCun. 2007. "Efficient Learning of Sparse Representations with an Energy-Based Model." In *Advances in Neural Information Processing Systems 19 - Proceedings of the 2006 Conference*, 1137–44.

Rifai, Salah, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. 2011. "Contractive Auto-Encoders: Explicit Invariance During Feature Extraction." In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 833–40. ICML'11. USA: Omnipress. http://dl.acm.org/citation.cfm?id=3104482.3104587.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1." In, edited by David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, 318–62. Cambridge, MA, USA: MIT Press. http://dl.acm.org/citation.cfm?id=104279.104293.

Salakhutdinov, Ruslan, and Geoffrey Hinton. 2009. "Semantic Hashing." *Int. J. Approx. Reasoning* 50 (7): 969–78. https://doi.org/10.1016/j.ijar.2008.11.006.

Shafkat, Irhum. 2018. "Intuitively Understanding Variational Autoencoders." 2018. https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf.

Takahashi, Hiroshi, Tomoharu Iwata, Yuki Yamanaka, Masanori Yamada, and Satoshi Yagi. 2018. "Student-T Variational Autoencoder for Robust Density Estimation." In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 2696–2702. International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2018/374.

Vincent, Pascal, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. "Extracting and Composing Robust Features with Denoising Autoencoders." In *Proceedings of the 25th International Conference on Machine Learning*, 1096–1103. ICML '08. New York, NY, USA: ACM. https://doi.org/10.1145/1390156.1390294.

Weiss, Yair, Antonio Torralba, and Rob Fergus. 2008. "Spectral Hashing." In *NIPS*.

Xie, Junyuan, Ross Girshick, and Ali Farhadi. 2016. "Unsupervised Deep Embedding for Clustering Analysis." In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, 478–87. ICML'16. New York, NY, USA: JMLR.org. http://dl.acm.org/citation.cfm?id=3045390.3045442.

Zhou, Chong, and Randy C. Paffenroth. 2017. "Anomaly Detection with Robust Deep Autoencoders." In *Proceedings of the 23rd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 665–74. KDD '17. New York, NY, USA: ACM. https://doi.org/10.1145/3097983.3098052.

Zong, Bo, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. "Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection." In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJJLHbb0-.