Final Project: Variational Autoencoders

Universitat de Barcelona

Facultat de Matemàtiques i Informàtica

Bayesian Statistics and Probabilistic Programming

2023-2024

Jaime Leonardo Sánchez Salazar

Sergio Hernández Antón

David Íñiguez Gómez

Sergi Cantón Simó

Contents

1 Introduction 2

1.1 Autoencoders . . . 2

1.2 Variational Autoencoders . . . 5

2 State of the Art 8

3 Model 9

3.1 Gaussian Encoder . . . 9

3.2 Bernoulli Decoder . . . 11

4 Experiments and Results 11

4.1 MNIST Dataset . . . 11

4.2 First Experiment . . . 12

4.3 Second Experiment . . . 13

4.4 Third Experiment . . . 15

5 Conclusions 15

References 16

Abstract

Variational autoencoder is one of the deep latent space generation models, which has become increasingly popular in image generation and anomaly detection in recent years. In this project, we first review the development and research status of traditional variational autoencoders. We then proceed to explain a particular one, which we will be using to perform a series of experiments. Our purpose is to fully understand how these models work and hold some experiments to assess their practical applications and performance.

1 Introduction

In this project, we are going to experiment with a special type of autoencoder (AE), called variational autoencoders (VAE). So, in order to understand how do VAEs work, first it is important to know the functioning of AE.

1.1 Autoencoders

As explained in [1], autoencoders are neural networks that are built so that the output of the model is the input, but trying to catch the important features of it. As we can see in Figure 1, AEs have a bottleneck structure, where the information is compressed and then decompressed, trying to capture the most relevant information of the input, to later reproduce the input considering only those important features. We could say that autoencoders are a form of compression.

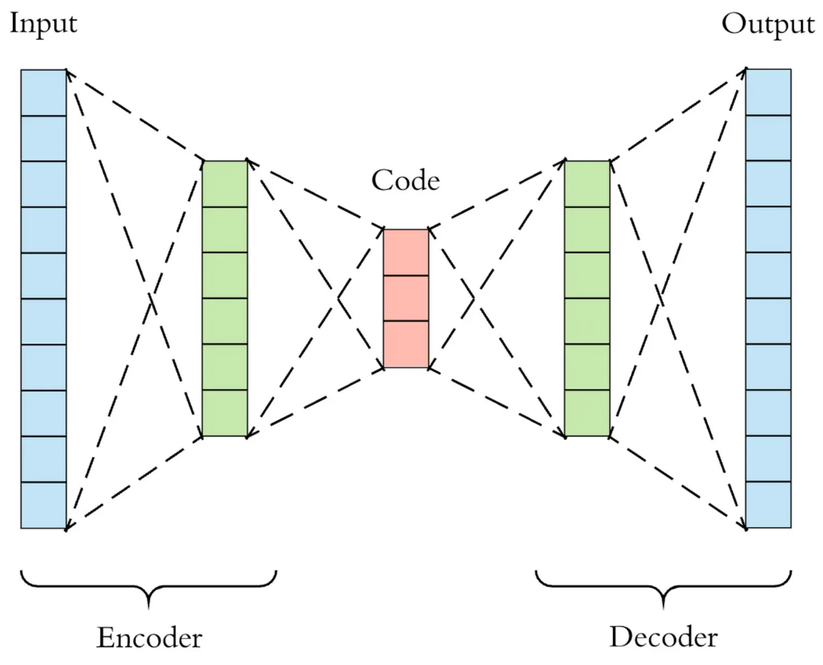

Figure 1: Autoencoder general structure.

The math behind the networks is fairly easy to understand, so we will go through it briefly. Essentially, we split the network into two segments, the encoder and the decoder.

$$\phi : \mathcal{X} \to \mathcal{F}$$

$$\psi : \mathcal{F} \to \mathcal{X}$$

$$\phi, \psi : \arg\min_{\phi,\psi} ||\mathcal{X} - (\psi \circ \phi)\mathcal{X}||^2$$

The encoder function, denoted by ϕ, maps the original data $\mathcal{X}$ to a latent space $\mathcal{F}$, which is present at the bottleneck. The decoder function, denoted by ψ, maps the latent space $\mathcal{F}$ at the bottleneck to the output. The output, in this case, is an approximated reconstruction of the input. Thus, we are basically trying to recreate the original image after some generalized non-linear compression.

In the simplest case, where we only have the input, hidden and output layers, the expression of the encoding network can be represented by the standard neural network function passed through an activation function σ, where z is the latent dimension, W and b are the weights and biases, respectively, and x the input.

$$z = \sigma(Wx + b)$$

Similarly, the decoding network can be represented in the same fashion using a different weight and bias and with the potential usage of activation functions.

$$x' = \sigma'(W'z + b')$$

We have that x' is the output of the model. With all that, we can perfectly define the loss function that will enable us to train the autoencoder through backpropagation.

$$\mathcal{L}(x, x') = ||x - x'||^2 = ||x - \sigma'(W'(\sigma(Wx + b)) + b')||^2$$

Since the input and output are the same images, this is not really supervised or unsupervised learning, so we typically call this self-supervised learning. The aim of the autoencoder is to select our encoder and decoder functions in such a way that we require the minimal information to encode the image such that it be can regenerated on the other side.

If we use too few nodes in the bottleneck layer, our capacity to recreate the image will be limited and we will regenerate images that are blurry or unrecognizable from the original. If we use too

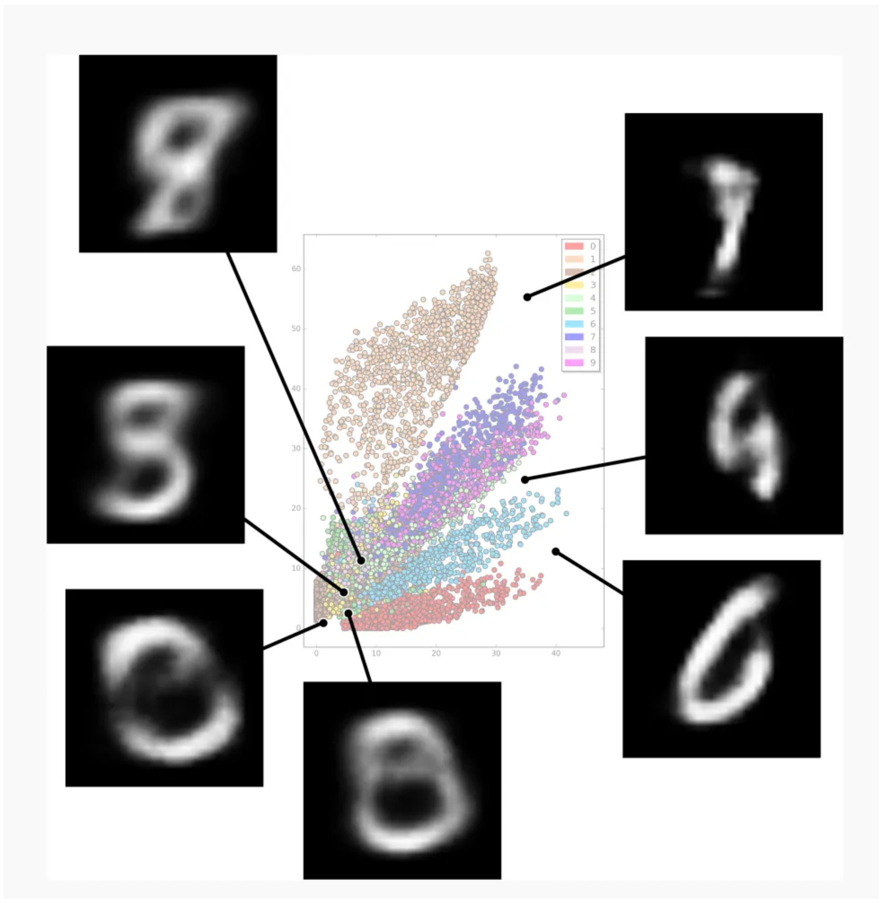

Figure 2: Latent space representation for MNIST dataset.

many nodes, then there is little point in using compression at all.

However, autoencoders have some weaknesses related to the properties of the latent space. This latent space is discrete, so we could find gaps in it and therefore its separability. This can be perfectly appreciated in Figure 2 (which shows the latent space representation for MNIST dataset), where there are gaps between the different labels within the latent space. Hence, we are not able to know which characters we are going to find in those blank spaces. This is equivalent to having a lack of data in a supervised learning problem, as our network has not been trained for these circumstances of the latent space. Another issue, as commented, is the separability of the different numbers. We can see that most numbers are correctly separated, but also we can find other sections where it is really difficult to tell the difference between labels (in this case the numbers 0 and 9).

These issues mean that if we want to improve the autoencoder performance, instead of generating the latent space points directly from the input, we have to learn a data generating distribution, which allows us to take random samples from the latent space, making it to be continuous instead of discrete. This is done by variational autoencoders (VAE), which use Bayes theorem to generate those samples.

1.2 Variational Autoencoders

As a first idea, VAEs inherit the architecture of traditional autoencoders and use this to learn a data generating distribution, which allows us to take random samples from the latent space. These random samples can then pass through our decoder network to generate unique images having similar characteristics to those with which the network was trained on.

In [1] and [2] the mathematical formulation of the VAEs is perfectly described, so we will base our explanation on those articles. Let us consider some dataset consisting of N i.i.d. samples of a random variable x (either scalar or vector). The data is assumed to be generated by some random process, involving an unobserved random variable z (i.e. the latent variable). The process we will follow is explained below:

1. We generate a value $z^{\{i\}}$, where $\{i\}$ refers to the iteration from some prior distribution $p(z;\theta)$.
2. After obtaining $z^{\{i\}}$, we generate a value $x^{\{i\}}$ from the conditional likelihood function $p(x|z = z^{\{i\}};\theta)$, which depends on $z^{\{i\}}$.

It is important to note that both prior and likelihood distributions depend on a set of parameters θ. One of the objectives of this algorithm is to estimate those parameters in order to generate artificial data from them. Another goal of this process is to determine the posterior inference of the latent variable z given an observed value x and the parameters θ.

To address this problem, a common approach consists on using the variational method (which give the name to this types of models). First, we start by determining the probability distribution $p(z|x = x^{\{i\}};\theta)$, which is unknown. We can analyze it using the Bayes theorem:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int_z p(x|z)p(z)dz}$$

Nonetheless, the denominator of the previous expression can be intractable, so instead of calculate it we will approximate it. In order to do that, let us introduce the Kullback-Leibler (KL) divergence, which measures how much a probability distribution differs from another. It quantifies the amount of information lost when using one distribution to approximate the other one. The lower the values, the more similar both distributions are (conversely, greater values indicate greater distribution differences).

In the case of continuous distributions, the KL divergence from a probability distribution P to a probability distribution Q is given by:

$$KL(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \, dx.$$

Let $\mathcal{Q}$ be a family of potential distributions representing how our data was generated. Since the real distribution is too difficult to compute, our goal is to find the one which minimizes the distance between our proposed distribution and the actual one we aim to approximate. With this, we can now convert the previous inference problem into an optimization one using following formula:

$$q^*(\mathbf{z}|\mathbf{x}) = \arg\min_{q\sim\mathcal{Q}} KL(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})) \tag{1}$$

However, we still do not know $p(z|x)$, so we cannot directly calculate the KL divergence. Let us manipulate the previous formula:

$$\begin{aligned} KL(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})) &= \int_{\mathbf{z}} q(\mathbf{z}|\mathbf{x}) \log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}\, d\mathbf{z} \\ &= \mathbb{E}_q[\log q(\mathbf{z}|\mathbf{x})] - \mathbb{E}_q[\log p(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}_q[\log q(\mathbf{z}|\mathbf{x})] - \mathbb{E}_q[\log p(\mathbf{x},\mathbf{z}) + \mathbb{E}_q[\log p(\mathbf{x})]] \\ &= \mathbb{E}_q[\log q(\mathbf{z}|\mathbf{x})] - \mathbb{E}_q[\log p(\mathbf{x},\mathbf{z}) + \log p(\mathbf{x})] \\ &= -\text{ELBO} + \log p(\mathbf{x}), \end{aligned}$$

where we applied the KL divergence definition, the quotient property of the logarithms, the definition of $p(z|x)$, the independence of $p(x)$ and q and that ELBO $= -\mathbb{E}_q[\log q(\mathbf{z}|\mathbf{x})] + \mathbb{E}_q[\log p(\mathbf{x},\mathbf{z})]$[1] .

Finally, rewriting ELBO we obtain:

$$\begin{aligned} \text{ELBO} &= -\mathbb{E}_q[\log q(\mathbf{z}|\mathbf{x})] + \mathbb{E}_q[\log p(\mathbf{x},\mathbf{z})] \\ &= -\mathbb{E}_q[\log q(\mathbf{z}|\mathbf{x})] + \mathbb{E}_q[\log p(\mathbf{z})] + \mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})] \\ &= -KL(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})) + \mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})]. \end{aligned}$$

Hence, omitting the dependence of the term $\log p(x)$ and maximizing the ELBO (since the KL divergence dependency is negative now) we can modify Expression 1 as follows:

$$q^*(\mathbf{z}|\mathbf{x}) = \arg\max_{q\sim\mathcal{Q},\theta,\phi} -KL(q(\mathbf{z}|\mathbf{x}^{\{i\}};\phi) \parallel p(\mathbf{z};\theta)) + \mathbb{E}_{q(\mathbf{z}|\mathbf{x}^{\{i\}};\phi)}[\log p(\mathbf{x}^{\{i\}}|\mathbf{z};\theta)] \tag{2}$$

After obtaining Expression 2, all that remains is to optimize it. We can obtain a suitable approach by applying stochastic gradient descent to the following loss function:

$$\mathcal{L}(\phi,\theta,\mathbf{x}^{\{i\}}) = KL\left(q(\mathbf{z}|\mathbf{x}^{\{i\}};\phi) \parallel p(\mathbf{z};\theta)\right) - \frac{1}{L}\sum_{l=1}^{L}\left[\log p\left(\mathbf{x}^{\{i\}}|\mathbf{z}^{\{i,l\}};\theta\right)\right], \tag{3}$$

[1]Note that ELBO is just an abbreviation for evidence lower bound.

where the expectation term is approximated using Monte Carlo method. Hence, we average $\log p(\mathbf{x}^{\{i\}}|\mathbf{z};\theta)$ over L samples $z^{\{i,l\}}$ drawn from $q(\mathbf{z}|\mathbf{x}^{\{i\}};\phi)$. Nevertheless, the samples $z^{\{i,l\}}$ are not drawn directly from $q(\mathbf{z}|\mathbf{x}^{\{i\}};\phi)$ in practise, due to the possibility of q being an arbitrarily complicated distribution and, hence, extremely difficult to sample. To improve sampling efficiency, we turn to the *reparameterization trick* by setting $z^{\{i,l\}} = g(\varepsilon^{\{i,l\}}, \mathbf{x}^{\{i\}};\phi)$, where $g(*,*;\phi)$ can be any neural network. The noise $\varepsilon^{\{i,l\}}$ is sampled from some simple distribution such as a gaussian distribution.

Now we can detail the full learning algorithm for VAE, which consists on repeating the following steps until convergence:

1. Get the minibatch of M datapoints.
2. Compute the minibatch loss $\sum \mathcal{L}(\phi, \theta, \mathbf{x}^{\{i\}})/M$.
3. Compute the gradients of step 2.
4. Apply the gradients to update parameters ϕ, θ.

Having explained the theoretical part of VAEs, let us have a look at its structure compared to the autoencoders usual one.

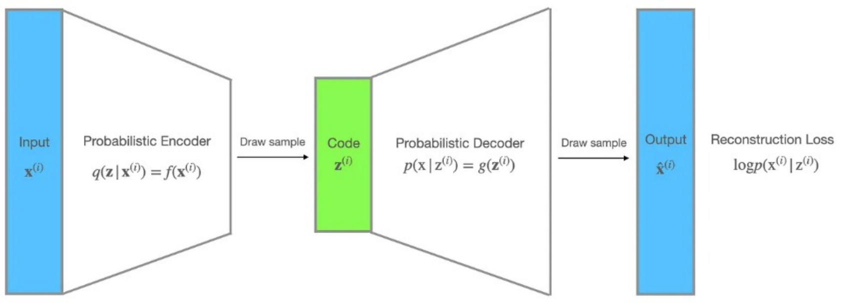

Figure 3: VAEs general structure.

The unobserved variables z can be interpreted as the code. In addition, the recognition model $q(z|x;\phi)$ can be treated as a probabilistic encoder, since given a data point x it produces a distribution over the possible values of z. On the other hand, $p(x|z;\theta)$ can be seen as a probabilistic decoder, since it produces a distribution over the possible corresponding values of x given a code z.

The Monte Carlo estimate term within the VAE loss function is exactly in the form of negative log likelihood, thus can be used as the reconstruction loss. Besides, there is a KL divergence term in the loss function (see 3), which acts as a regularizer and enforces the distribution $q(z|x;\phi)$ to stay close to the prior $p(z;\theta)$.

In essence, VAEs can be seen as a probabilistic extension of AEs. Both are powerful tools for representation learning. However, VAEs have the distinct advantage of explicitly modeling the generative process, allowing them to generate new data points that closely resemble the real data by sampling from $p(x|z;\theta)$. Additionally, the distributions learned by VAEs are highly valuable for statistical analysis.

2 State of the Art

Variational Autoencoders (VAEs), introduced in [3], have emerged as a cornerstone in generative modeling and unsupervised learning. VAEs combine neural networks with probabilistic graphical models to learn latent representations, which then can be used to generate new data samples. This dual capability of representation learning and data generation has positioned VAEs as a versatile tool in various domains. Recent advancements have focused on improving the representational power, efficiency and applicability of VAEs to a broader range of tasks.

One significant development in the VAE framework is the introduction of hierarchical VAEs (HVAEs), which incorporate multiple layers of latent variables to capture complex hierarchical structures in data in [4]. This approach enhances the model's capacity to represent intricate dependencies within the data, leading to better performance in tasks requiring high-level abstractions. Similarly, Vector Quantized VAEs (VQ-VAEs), introduced in [5], replace continuous latent variables with discrete ones, leveraging a codebook of latent embeddings. VQ-VAEs have been particularly successful in image and speech synthesis, where discrete representations can provide more precise and effective encoding of data.

The β-VAE variant, proposed in [6], introduces a hyperparameter β to balance the trade-off between reconstruction accuracy and latent space regularization. This modification allows for better disentanglement of latent factors, enhancing interpretability and controlled data generation. Furthermore, Adversarially Regularized VAEs (AR-VAEs) combine VAEs with adversarial training techniques, similar to those used in Generative Adversarial Networks (GANs), to regularize the latent space and improve sample quality (see [7]). These advancements collectively address some of the limitations of standard VAEs, such as poor sample quality and difficulty in scaling to high-dimensional data.

VAEs have found applications across a wide range of fields, demonstrating their versatility and impact. In image processing, VAEs are used for tasks like face generation, image denoising and inpainting. They are also employed in anomaly detection, with applications in industrial equipment monitoring and network security. In the realm of drug discovery and molecular design, VAEs help generate novel chemical compounds and predict protein structures, aiding in the discovery of new drugs. Additionally, VAEs are utilized in natural language processing for text generation, language modeling and dialogue systems, as well as in audio and speech processing for voice generation and music composition. Despite these successes, challenges such as scalability, sample quality and achieving robust disentanglement remain areas of active research.

Indeed, VAEs are versatile tools used across various domains due to their ability to model complex data distributions and generate high-quality synthetic data. Another important application is that VAEs are used for latent space exploration, enabling style transfer in images and data visualization, helping to uncover hidden structures in high-dimensional data. These applications highlight the broad utility of VAEs in modeling and generating data across multiple fields. In sections 4 and 5 we demonstrate some of its applications through our experiments.

Future directions for VAE research include enhancing their scalability and efficiency, integrating them with other generative models for hybrid approaches and improving their applicability to real-world problems through better disentanglement and more robust training methods. As the field progresses, VAEs are expected to continue evolving, driven by ongoing advancements and novel applications that push the boundaries of what these models can achieve

3 Model

Once we have explained the idea of an autoencoder and after seeing the fundamental principles of variational autoencoders and their state of the art, we will focus on one specific VAE. The selected autoencoder is the one presented in [3]. This model will help us to see an example of a specific VAE and, in the next section, we will see the results of some experiments that we have performed using this VAE with the MNIST dataset.

The proposed VAE (Figure 4) consists of two parts: a Gaussian autoencoder and a Bernoulli decoder. Now, let us explain each part in detail.

3.1 Gaussian Encoder

A Gaussian encoder aims to model the latent space using multidimensional Gaussian distributions. In our case, given an image, the parameters of the Gaussian distribution of one sample are obtained with a previously trained feedforward neural network.

As we well know, a Gaussian distribution has two parameters: the mean, μ, and the variance, σ^2. To estimate these two parameters of a specific black and white image, we first flatten the image in a single one-dimensional array. Then, data is passed through a linear layer, $\mathbf{h}$, with a

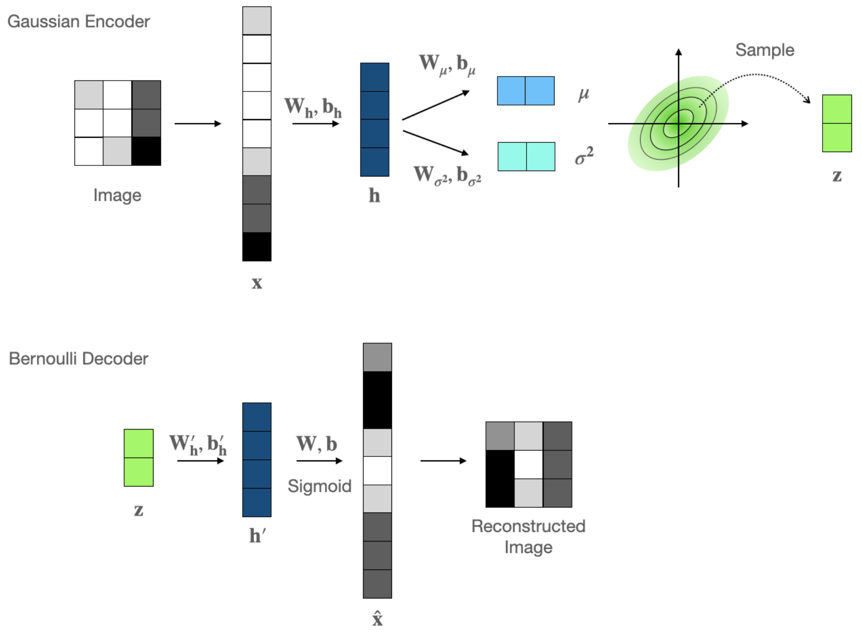

Figure 4: Structure of the VAE used.

tanh activation function. After this, the flow is divided in two: the calculation of the mean and the computation of the variance. Both mean and variance are computed by applying a linear layer to previous data. However, different parameters are used in both cases and, in the case of the variance, we exponentiate the obtained result at the end.

Mathematically, we obtain the following:

$$\mathbf{h} = \tanh(\mathbf{W_h x} + \mathbf{b_h})$$
$$\mu = \mathbf{W}_\mu h + \mathbf{b}_\mu$$
$$\sigma^2 = \exp(\mathbf{W}_{\sigma^2} h + \mathbf{b}_{\sigma^2}),$$

where $\mathbf{z}$ is the flattened image and $(\mathbf{W_h}, \mathbf{W}_\mu, \mathbf{W}_{\sigma^2})$ and $(\mathbf{b_h}, \mathbf{b}_\mu, \mathbf{b}_{\sigma^2})$ are the parameters (that is, respectively the weights and the biases) of each of the layers mentioned before.

The representation of the image as an n-dimensional vector of the latent space, $\mathbf{z}$, will be a sample of a normal distribution with the estimated parameters μ and σ^2.

Formally, we have that our q function is:

$$q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu, \sigma^2\mathbf{I}).$$

3.2 Bernoulli Decoder

The decoding part of the VAE is structured as a Bernoulli decoder. As the MNIST samples are gray-scale images, pixel values are floats between 0 and 1. Therefore, each number can be interpreted as the "probability of the pixel being 1", which is the probability of a Bernoulli distribution.

The decoder is able to reconstruct the image from the vector $\mathbf{z}$ in the latent space. First of all, the values of $\mathbf{z}$ are passed through a linear layer, $\mathbf{h}'$, with a tanh activation function. Afterwards, the flattened image is reconstructed using another linear layer, though this time we use sigmoid as our activation function. Finally, the image is obtained by stacking the flattened pixels of the image into 28 rows of 28 pixels.

Mathematically, we can express this process as:

$$\mathbf{h}' = \tanh(\mathbf{W}'_{\mathbf{h}}\mathbf{z} + \mathbf{b}'_{\mathbf{h}}),$$

where $\mathbf{W}'_{\mathbf{h}}$ and $\mathbf{b}'_{\mathbf{h}}$ are, respectively, the weights and biases of the first linear layer.

Then, the probability of the pixels of the flattened image $\mathbf{x}$ given the latent vector $\mathbf{z}$ can be expressed as:

$$p(\mathbf{x}|\mathbf{z}) = f_\sigma(\mathbf{W}\mathbf{h}' + \mathbf{b}),$$

where $\mathbf{W}$ and $\mathbf{b}$ are, respectively, the weights and biases of the second and last linear layer and f_σ is a sigmoid function.

4 Experiments and Results

With the explained specific VAE in the previous section, we held some experiments to see how a variational autoencoder works in practise and which results can be obtained from it.

First of all, we start by explaining the dataset used and, afterwards, we will detail each of our experiments and their results.[2]

4.1 MNIST Dataset

The selected VAE autoencoder is thought to be trained and tested with the specific MNIST dataset, which stands for *Modified National Institute of Standards and Technology dataset.* This is a well-known and widely used dataset, which consists of a 60,000 samples training set and 10,000 samples test set. The samples are 28×28 grey-scaled images of handwritten digits obtained from a combination of U.S. Census Bureau employees and high-school students. MNIST is optimal for learning

[2]The original code of the VAE is available here, while the modification of the code used can be found in our GitHub repository. We remark that the experiments can be reproduced using the auxiliary Python notebook.

pattern recognition methods on real-world data while expending minimal effort on preprocessing. Furthermore, it can be easily obtained from some sources such as the *PyTorch* Python library.

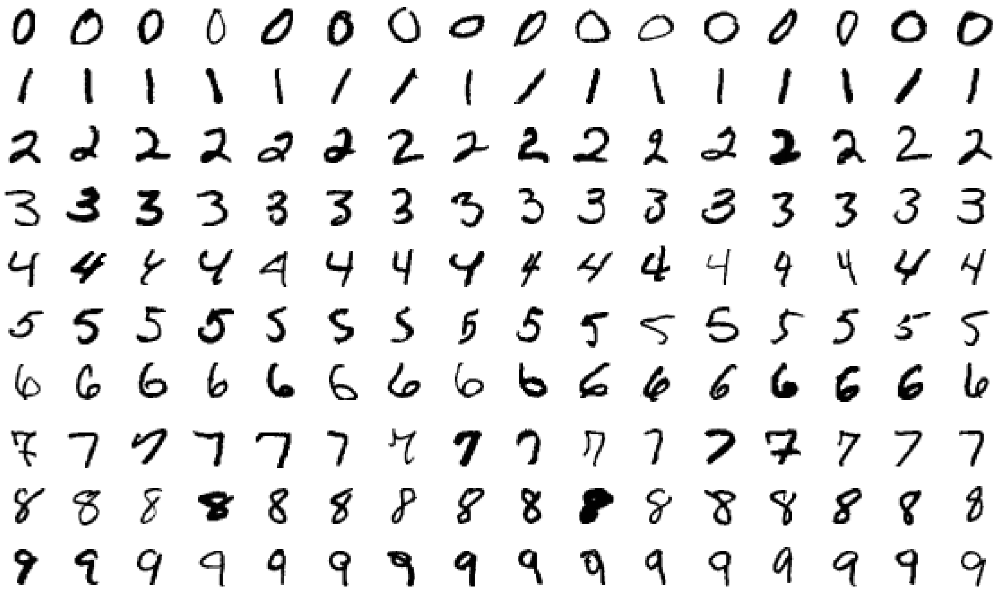
Figure 5: Subset of the MNIST database of handwritten digits.

4.2 First Experiment

As previously explained, our autoencoder represents images as vectors (or points) of the latent space. However, the dimension of the latent space can vary depending on the precision wanted because, in fact, it is a hyperparameter of the model. If we select a bottleneck of a small dimension (2 or 3, for instance), we will reconstruct the images inaccurately. That is, the recovered images will be more blurry and will differ from the original ones. This is not necessarily a bad outcome; it depends on the task we want to perform. Instead, taking a bigger dimension of the latent space, which means a bigger bottleneck, will lead to less blurry images and more similar to the original ones.

Hence, our first experiment consists on comparing original images with reconstructed ones with four different values of the latent space dimension. This experiment will help us to see how each pair of images differ varying the number of variables of the latent representation.

For fair comparison purposes, one random image from each different handwritten digit has been chosen. Then, using our trained VAE we obtained the latent vector of each image using the Gaussian encoder before reconstructing them from $\mathbf{z}$ through the Bernoulli decoder. We repeated the process for 2, 5, 10 and 50 as values of dimension. The obtained results are shown in Figure 6.

After observing the results, it seems fair to state that the greater the dimension of the latent space, the more similar the generated images are to the original ones.

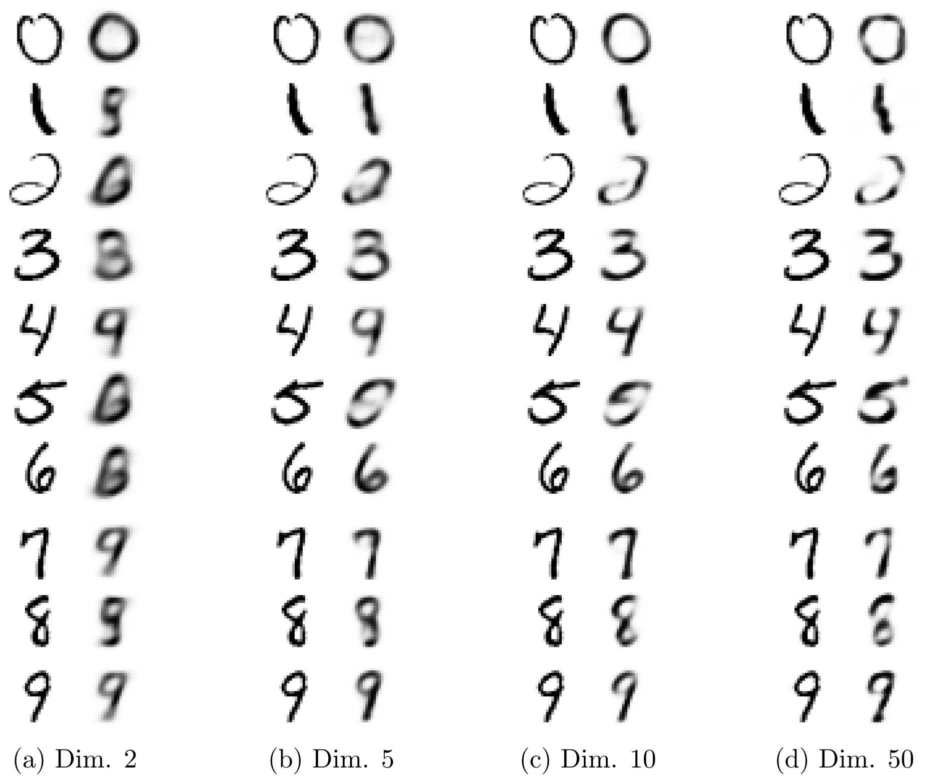
(a) Dim. 2 (b) Dim. 5 (c) Dim. 10 (d) Dim. 50

Figure 6: Comparison between the original images (left part of each sub-image) and the reconstructed ones (right part of each sub-image) for 4 different values of dimension of the latent space and for one example of each digit.

If we take a look to the reconstructed images with a latent space of dimension 2, we can observe that most of them are blurry and, in some of them, it is difficult to know what digit is even for humans. For instance, the images corresponding to the digits 1 and 6 are not clear, the 7 image could be confounded with a 9 and the 3 with an 8.

When we increase the dimension to 5, we also notice the potential in misclassification for the 4 image. Nonetheless, the images are now more similar to the original ones than in the previous case.

For dimensions 10 and 50, specially with the latter, there are no misclassifications. We also observe the increasing resemblance between both images.

4.3 Second Experiment

Once we have shown the operation of the model we proceed to our second experiment, which consists on plotting the manifold of the MNIST using a dimension of the latent space of 2. As we have seen, we need not the original images if we generate new ones by choosing a **z** vector and reconstructing them. However, it is interesting to see how the reconstructed images behave as we modify the elements of the latent space vector.

To perform this experiment, we have selected a bottleneck of 2 variables to plot a 2-dimensional grid of images, each one generated with a vector $\mathbf{z}$ with a combination of elements z_1, z_2 of a grid of 20 equispaced values from -3 to 3 for each variable. Therefore, the final plot results into a 20×20 mesh of 28×28 generated images. That is, a total of 400 different 28×28 images.

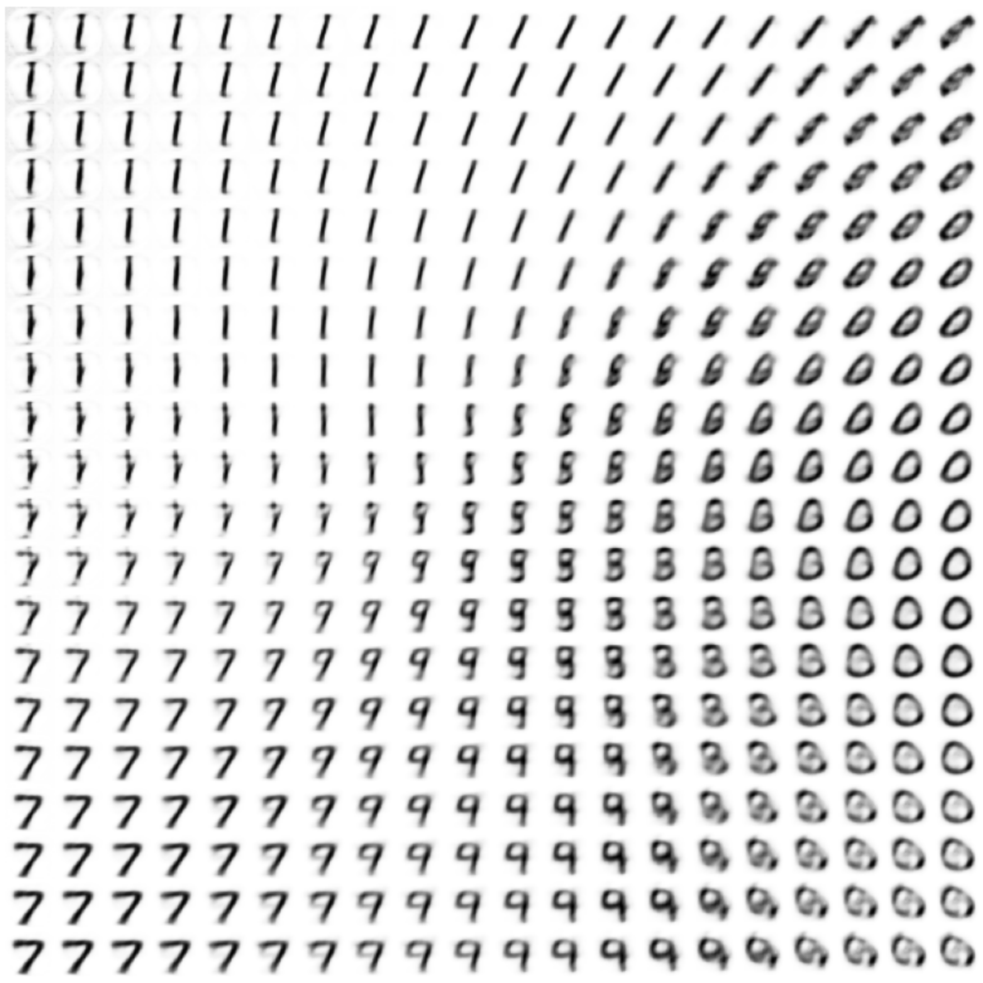

Figure 7: Meshgrid of the generated images. As the range of values of z_1, z_2 is from -3 to 3, the top left image corresponds to $\mathbf{z} = (-3, -3)$, while the bottom right one to $\mathbf{z} = (3, 3)$.

We can see from Figure 7 that there are regions where it is clear what digit we have. For instance, in the top left region we have the digit 1, the bottom left belongs to the digit 7 and the bottom right to 0. Nevertheless, there are some regions, for instance the top right corner, where it is not clear what digit we have.

As we saw before, this is due to the small dimension we are taking. If we plotted the same image for dimension 50, we would see more clear and less blurry images. Nonetheless, the obtained results are visually interesting and give a clear intuition on how it would work with higher dimensions.

4.4 Third Experiment

Our third experiment is, in fact, an extension to complement the second one. Its aim is to see how images are clustered for each label, which would lead us to observe similar patterns to the ones seen in the second experiment.

To accomplish it, we took 24 original images from each different digit and obtained its latent vector in two dimensions using our gaussian encoder. Then, we did a scatter plot of the **z** vectors (or points) obtained, using a different color for each label (representing each digit).

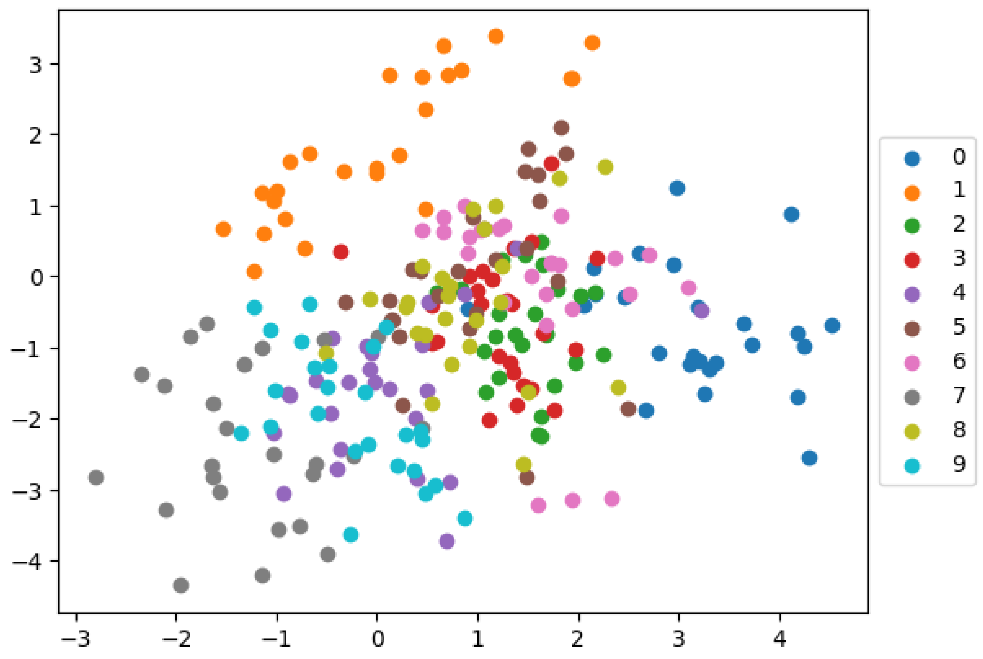

Figure 8: Scatter plot of the 2-dimensional latent vectors (or points) of 24 images for each label.

In the scatter plot shown in Figure 8, we notice that the pattern of the clusters found is similar to the results obtained in the previous experiment. As we saw before, the top left is dominated by "1" points, the bottom left by "7" points and, the bottom right, by "0" points. Similarly to what happened in the second experiment, the central region and the top right one are formed by a mix of different points from different labels, which could be solved by increasing the dimension of the latent space.

5 Conclusions

Summarizing, in this project we have explained variational autoencoders and their precursors, as well as performing three experiments to get a better grasp of their practical applications and their performance in these scenarios.

Our experiments show that the reconstructed images resemble the original ones when the dimensionality of the latent space is high enough (in our tests, 20 and 50). Moreover, we also showed how the points in the latent space are clustered depending on the digit we are representing.

Unfortunately, we could not obtain a partition of the latent space for the different digits, due to the plots being extremely difficult to interpret when considering latent spaces of dimension higher than 3. Nevertheless, we believe our results are good enough to fulfill the goals of this project due to its nature. Had we not be restricted by both time and written pages, we would have completed the proposed experiments and obtain the minimal dimension with which each digit is reconstructed accurately, though the results would only be interpretable in the second experiment.

References

[1] Matthew Stewart. *Comprehensive Introduction to Autoencoders.* `https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368`. Apr. 2019.

[2] JZ. *Mathematics Behind Variational AutoEncoders.* `https://medium.com/@j.zh/mathematics-behind-variational-autoencoders-c69297301957`. May 2022.

[3] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[4] Casper Kaae Sønderby et al. "Ladder variational autoencoders". In: *Advances in neural information processing systems* 29 (2016).

[5] Aaron Van Den Oord, Oriol Vinyals, et al. "Neural discrete representation learning". In: *Advances in neural information processing systems* 30 (2017).

[6] Irina Higgins et al. "beta-vae: Learning basic visual concepts with a constrained variational framework." In: *ICLR (Poster)* 3 (2017).

[7] Alireza Makhzani et al. "Adversarial autoencoders". In: *arXiv preprint arXiv:1511.05644* (2015).