TME #9: Variational Auto Encoder

In this TME, we train a Variational Auto Encoder (VAE) on the MNIST dataset. The VAE is made of 2 modules: an encoder and a decoder. The encoder learns a representation of the input images as the mean and diagonal covariance of a normal law in a latent space of dimension d. The decoder samples a normal vector z from that law, and from that vector learns to reconstruct the original images.

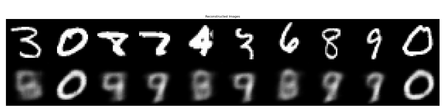
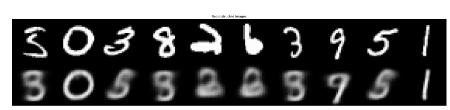
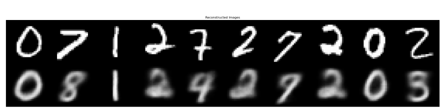
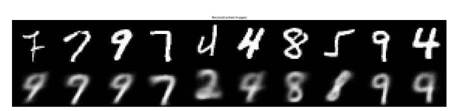
The MNIST images being grayscale, that is 2D vectors of pixels taking values between 0 and 1, they can be modeled as realizations of variables following a Bernouilli distribution. We want to maximize the expectation under the density of z of the log-likelihood of the data - which follows a Bernouilli distribution: that is equivalent to minimizing the cross-entropy between the probabilities computed by the decoder (pixel per pixel), and the real values of these probabilities in the real images. The final loss is the sum of that reconstruction loss and of a penalization keeping the output of the encoder close to a normal distribution.

We use the native torch MNIST dataset, and do not normalize the data to keep our vectors between 0 and 1.

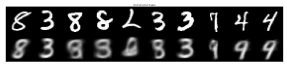
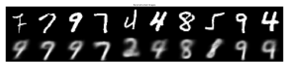
We try 2 different architecture: a linear-based architecture with fully connected layers, to which we pass a flattened view of our images, and a convolutive architecture, similar to the one used in our GAN (see TME #8). For each architecture, we build 5 different models with the latent dimensions: `[2, 3, 5, 10, 50]`. We train each model for 15 epochs, and regularly look at the image reconstructions on both the train and test set.

We can look at the quality of the reconstructed images evolve during training:

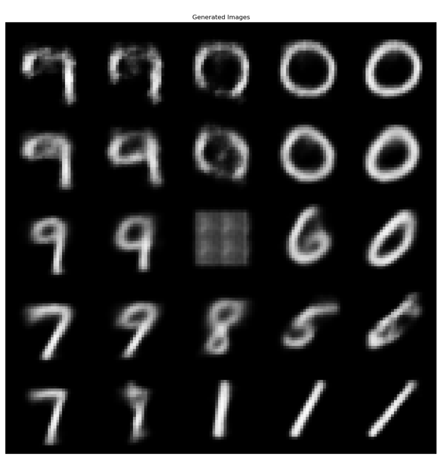
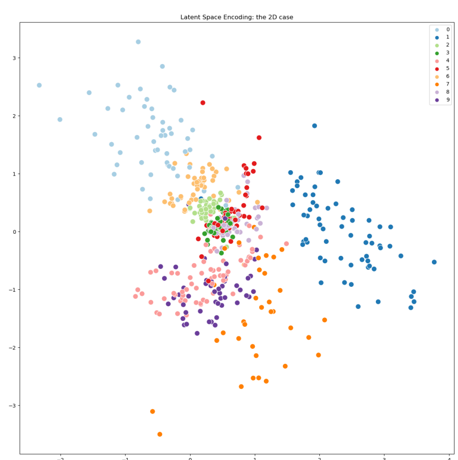
Figure Evolution of reconstructed images from the test set, for convolutive archiecture, latent space of dimension 2.

Epoch 1	Epoch 2
	
Epoch 10	Epoch 15
	

We also look at the evolution of the loss and its componenents (reconstruction loss and log-likelihood). As expected the reconstruction loss decreases as the dimension of the latent space increases, with the difference going from around $7,000$ to almost $15,000$ for the 2D latent-space model on the test set. At each dimension, the convolutive architecture performs slightly better than the linear one, with 2 times less parameters for small dimensions ($Linear2$ has 400K parameters, $Conv2$ 200K). The ranking is the opposite for the KL divergence, where linear, small dimension latent spaces models unsurprisignly have lower losses.

	Dimension 2	Dimension: 50
Linear		
Conv		

Let's explore the meaning of the latent space on our 2D model: To gain some understanding of what these dimensions mean, we reconstruct 25 images - with the same random seed - from two $z1$ and $z2$ coordinates evenly spaced across the standard normal disribution: [-2, -1, 0, 1, 2]. The result is the following grid (*left*):

Figure Exploration of the 2D-latent space	**Figure** Display of the latent space embeddings
	

We can also look at the samples's coordinates in the latent space (*right*). Numbers with the most identifiable shapes - zeros, ones - are also projected in well defined clusters at the edge of the learned representation. Numbers whose shape are more similar tend to be projected together. This explains why as shown in our prevous figure, our 2D-model still frequently reconstructs a 9 for a 7 or a 8 for a 5.