

Deep learning

## 7.2. Deep Autoencoders

François Fleuret

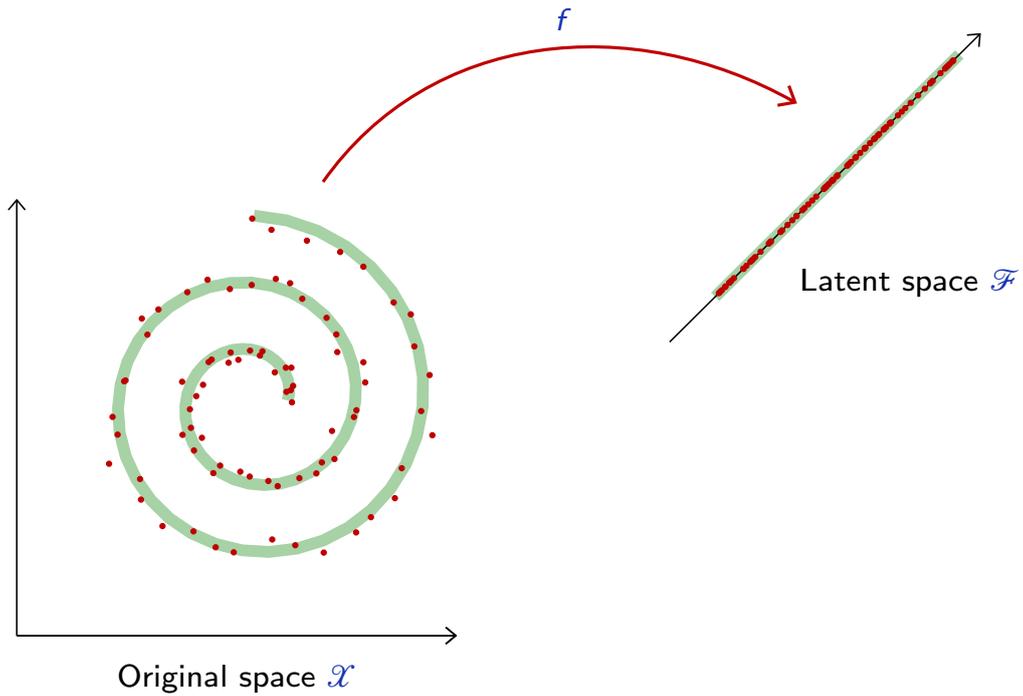
<https://fleuret.org/dlc/>



**UNIVERSITÉ  
DE GENÈVE**

Many applications such as image synthesis, denoising, super-resolution, speech synthesis, compression, etc. require to go beyond classification and regression, and model explicitly a high dimension signal.

This modeling consists of finding “meaningful degrees of freedom” that describe the signal, and are of lesser dimension.

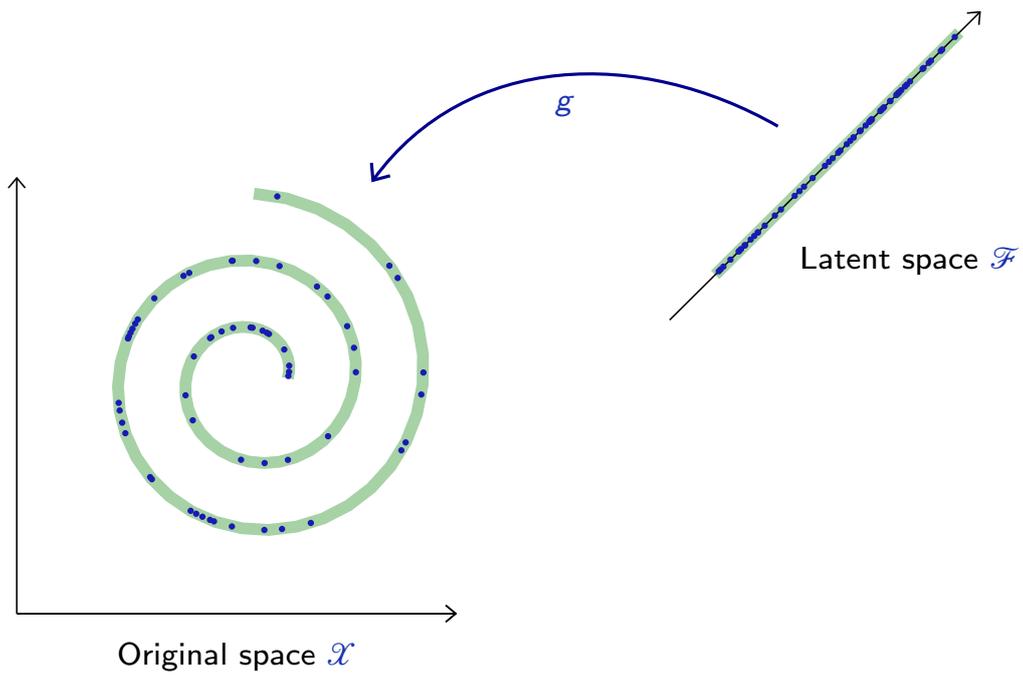


---

## Notes

We consider here a toy 2d data-set. The structure of the population is a “thick” 1d manifold, depicted as a green spiral.

Hence, there exists a mapping  $f$  which projects the points into a so-called “latent” 1d space  $\mathcal{F}$ , so that two different point from the manifold are mapped to two different points in the latent space.



---

## Notes

A way of generating new points would be to generate points in that latent space, and to then map them back to the original space with another mapping  $g$  that approximate  $f^{-1}$  on the manifold.

When dealing with real-world signals, this objective involves the same theoretical and practical issues as for classification or regression: defining the right class of high-dimension models, and optimizing them.

This motivates the use of deep architectures for signal synthesis.

---

## Notes

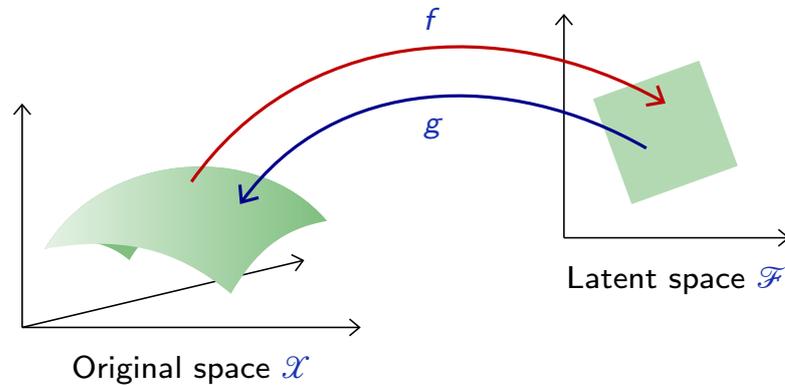
If we are given images of human faces, and we want to generate new ones, it makes sense to try to capture a small number of degrees of freedom such as morphological aspects (e.g. shape of the skull, color of the eyes, length of the nose) or physical context (e.g. orientation in the image plan, illumination).

Even though there may be many of them, there are definitely less than the resolution of the image. It is reasonable to think that a proper way of synthesizing a human face would be to model a hundred dimensions, so that given those hundred dimensions, one would be able to generate one million pixels.

# Autoencoders

An autoencoder maps a space to itself and is [close to] the identity on the data.

Dimension reduction can be achieved with an autoencoder composed of an **encoder**  $f$  from the original space  $\mathcal{X}$  to a **latent** space  $\mathcal{F}$ , and a **decoder**  $g$  to map back to  $\mathcal{X}$  (Bourlard and Kamp, 1988; Hinton and Zemel, 1994).



If the latent space is of lower dimension, the autoencoder has to capture a “good” parametrization, and in particular dependencies between components.

---

## Notes

The original space  $\mathcal{X}$  is of high dimension but the data (in green) lies on a manifold of much smaller dimension.

The encoder  $f$  maps the data to the latent space and the decoder  $g$  maps the data back to the original space.

The dimension of the latent space is a meta-parameter of the overall model and has to be chosen from prior knowledge or through trial-and-error.

Let  $q$  be the data distribution over  $\mathcal{X}$ . A good autoencoder could be characterized with the quadratic loss

$$\mathbb{E}_{X \sim q} [\|X - g \circ f(X)\|^2] \simeq 0.$$

Given two parametrized mappings  $f(\cdot; w_f)$  and  $g(\cdot; w_g)$ , training consists of minimizing an empirical estimate of that loss

$$\hat{w}_f, \hat{w}_g = \operatorname{argmin}_{w_f, w_g} \frac{1}{N} \sum_{n=1}^N \|x_n - g(f(x_n; w_f); w_g)\|^2.$$

A simple example of such an autoencoder would be with both  $f$  and  $g$  linear, in which case the optimal solution is given by PCA. Better results can be achieved with more sophisticated classes of mappings, in particular deep architectures.

---

## Notes

Given  $x \in \mathcal{X}$ ,  $f(x)$  is the projection in the latent space, and  $g(f(x))$  is its reconstruction back into  $\mathcal{X}$ .

We can say that  $(f, g)$  is a good autoencoder when the expected error (here quadratic) between a data sample and its reconstructed version is small. This means that  $g \circ f$  behaves like the identity on the original dataset.

# Deep Autoencoders

A deep autoencoder combines an encoder composed of convolutional layers, with a decoder composed of transposed convolutions or other interpolating layers. E.g. for MNIST:

```
AutoEncoder (  
  (encoder): Sequential (  
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))  
    (1): ReLU (inplace)  
    (2): Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1))  
    (3): ReLU (inplace)  
    (4): Conv2d(32, 32, kernel_size=(4, 4), stride=(2, 2))  
    (5): ReLU (inplace)  
    (6): Conv2d(32, 32, kernel_size=(3, 3), stride=(2, 2))  
    (7): ReLU (inplace)  
    (8): Conv2d(32, 8, kernel_size=(4, 4), stride=(1, 1))  
  )  
  (decoder): Sequential (  
    (0): ConvTranspose2d(8, 32, kernel_size=(4, 4), stride=(1, 1))  
    (1): ReLU (inplace)  
    (2): ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(2, 2))  
    (3): ReLU (inplace)  
    (4): ConvTranspose2d(32, 32, kernel_size=(4, 4), stride=(2, 2))  
    (5): ReLU (inplace)  
    (6): ConvTranspose2d(32, 32, kernel_size=(5, 5), stride=(1, 1))  
    (7): ReLU (inplace)  
    (8): ConvTranspose2d(32, 1, kernel_size=(5, 5), stride=(1, 1))  
  )  
)
```

---

## Notes

A deep autoencoder is an autoencoder in which both the encoder and the decoder are deep models.

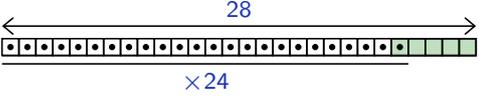
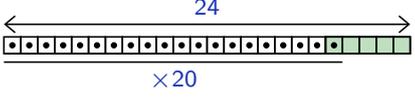
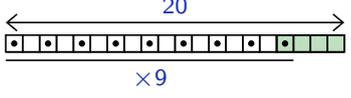
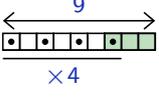
In this example as we will see in the next slide, with a  $1 \times 28 \times 28$  sample as input, the output of the encoder is  $8 \times 1 \times 1$ , hence the latent space is of dimension 8.

The dimension reduction in the encoder is achieved with a stride of 2 in the convolutional layers.

The decoder performs the computation in the reverse order, each transposed convolution layer corresponding to a convolution layer in the encoder.

## Encoder

Tensor sizes / operations

$1 \times 28 \times 28$	
<code>nn.Conv2d(1, 32, kernel_size=5, stride=1)</code>	
$32 \times 24 \times 24$	
<code>nn.Conv2d(32, 32, kernel_size=5, stride=1)</code>	
$32 \times 20 \times 20$	
<code>nn.Conv2d(32, 32, kernel_size=4, stride=2)</code>	
$32 \times 9 \times 9$	
<code>nn.Conv2d(32, 32, kernel_size=3, stride=2)</code>	
$32 \times 4 \times 4$	
<code>nn.Conv2d(32, 8, kernel_size=4, stride=1)</code>	
$8 \times 1 \times 1$	

### Notes

This table shows the size of the internal activation tensors in the encoder.

Each row represents the dimension of the **input** tensor to a layer, and accounts for both the height and the width, as images are squares.

The **•**s represent the locations where the filter is applied, and show the stride.

The highlighted locations in green depicts the last location and size of the convolutional filter, and shows why the tensor size is reduced.

The double arrows above each grid is the dimension of the input tensor in the layer (number of squares in the row), while the number below the grid (preceded by **x**) is the number of times the filter could be moved to perform the convolution, taking into account the filter size and the stride.

In the end, the encoder processes a  $1 \times 28 \times 28$  into a signal of size  $8 \times 1 \times 1$ .

## Decoder

Tensor sizes / operations

---

$8 \times 1 \times 1$ <code>nn.ConvTranspose2d(8, 32, kernel_size=4, stride=1)</code> $32 \times 4 \times 4$ <code>nn.ConvTranspose2d(32, 32, kernel_size=3, stride=2)</code> $32 \times 9 \times 9$ <code>nn.ConvTranspose2d(32, 32, kernel_size=4, stride=2)</code> $32 \times 20 \times 20$ <code>nn.ConvTranspose2d(32, 32, kernel_size=5, stride=1)</code> $32 \times 24 \times 24$ <code>nn.ConvTranspose2d(32, 1, kernel_size=5, stride=1)</code> $1 \times 28 \times 28$	
--	--

---

### Notes

This table shows the size of the internal activation tensors in the decoder.

Each row represents the dimension of the **output** tensor from a layer, and accounts for both the height and the width, as images are squares.

The ● represent the locations where the filter is applied, and show the stride.

The highlighted locations in green depicts the last location and size of the convolutional filter, and shows why the tensor size is increased.

The number above the grid (preceded by ×) is the number of times the filter could be moved to perform the transposed convolution, while the double arrow below each grid is the resulted dimension of the output tensor.

Training is achieved with quadratic loss and Adam

```
model = AutoEncoder(nb_channels, embedding_dim)

optimizer = optim.Adam(model.parameters(), lr = 1e-3)

for epoch in range(args.nb_epochs):
    for input in train_input.split(batch_size):
        z = model.encode(input)
        output = model.decode(z)
        loss = 0.5 * (output - input).pow(2).sum() / input.size(0)

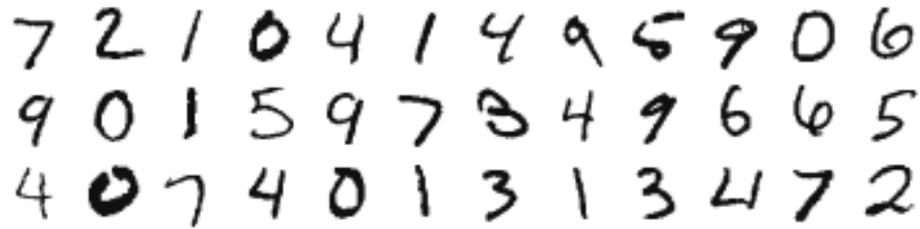
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

---

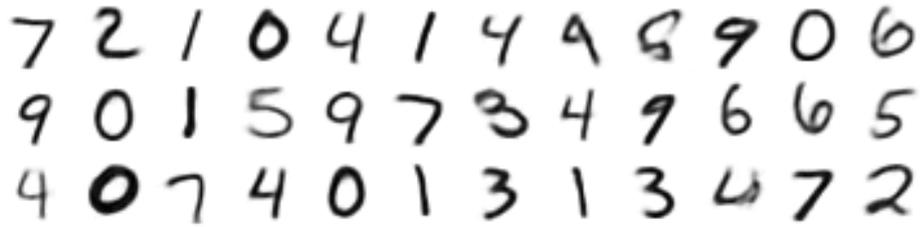
## Notes

`embedding_dim` is the number of channels at the end of the encoder (8 in the previous slides). Note that this is an **unsupervised** training: the labels of the samples are not used. The goal is to model the density of the distribution of the data.

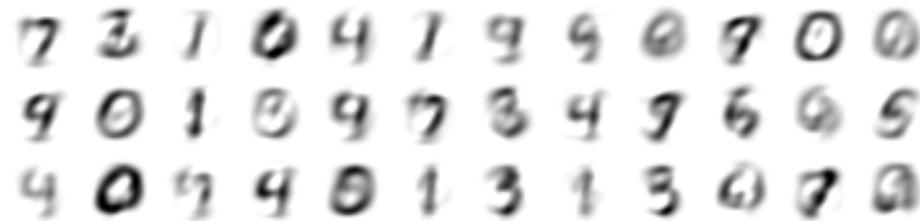
$X$  (original samples)



$g \circ f(X)$  (CNN,  $d = 8$ )



$g \circ f(X)$  (PCA,  $d = 8$ )



---

## Notes

The top images are some examples of original test MNIST images.

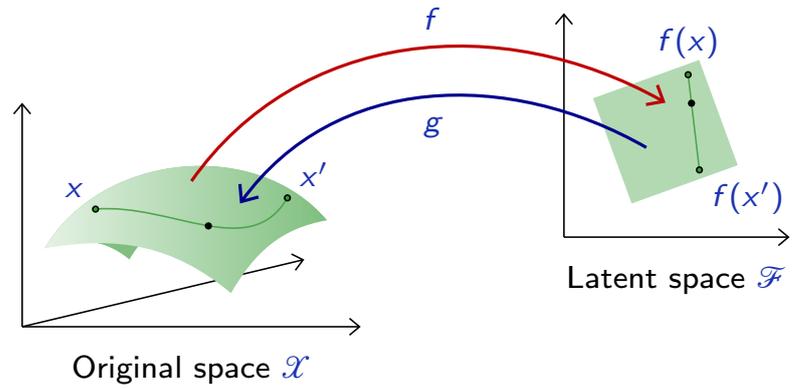
The middle images are the outputs of the corresponding original images as reconstructed when they go through the deep autoencoder described in the previous slides with a latent space of 8 dimensions

As a comparison, we show the projection with PCA, by equalizing the dimension of the latent space.

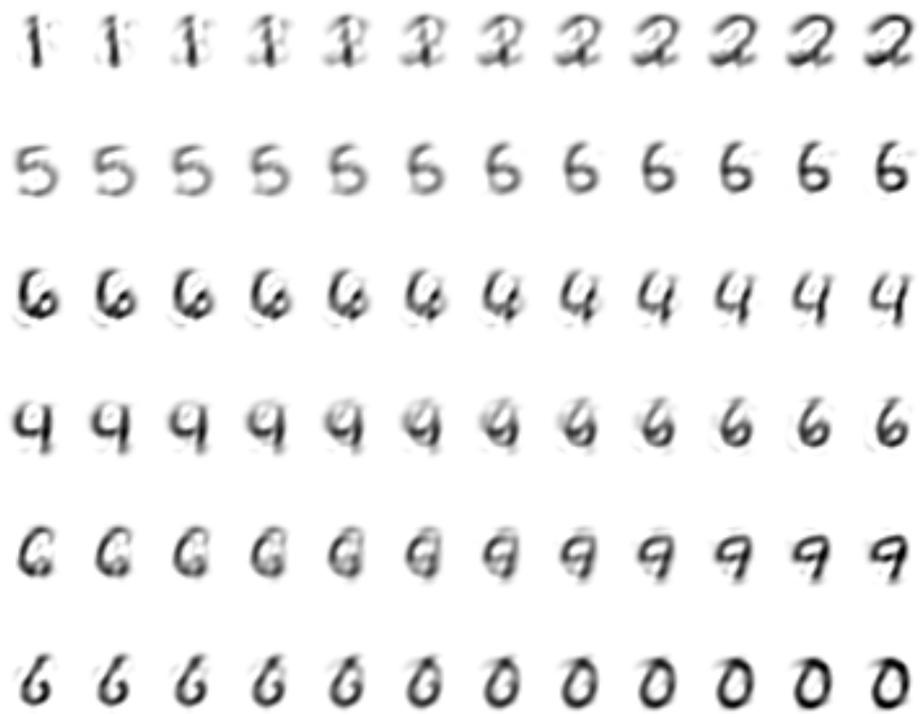
As expected, when a digit has an unusual shape, the reconstructed image is less accurate. Some statistically unusual details such as holes in the lines are not reconstructed. Such details can be reconstructed with a latent space of greater dimensions.

To get an intuition of the latent representation, we can pick two samples  $x$  and  $x'$  at random and interpolate samples along the line in the latent space

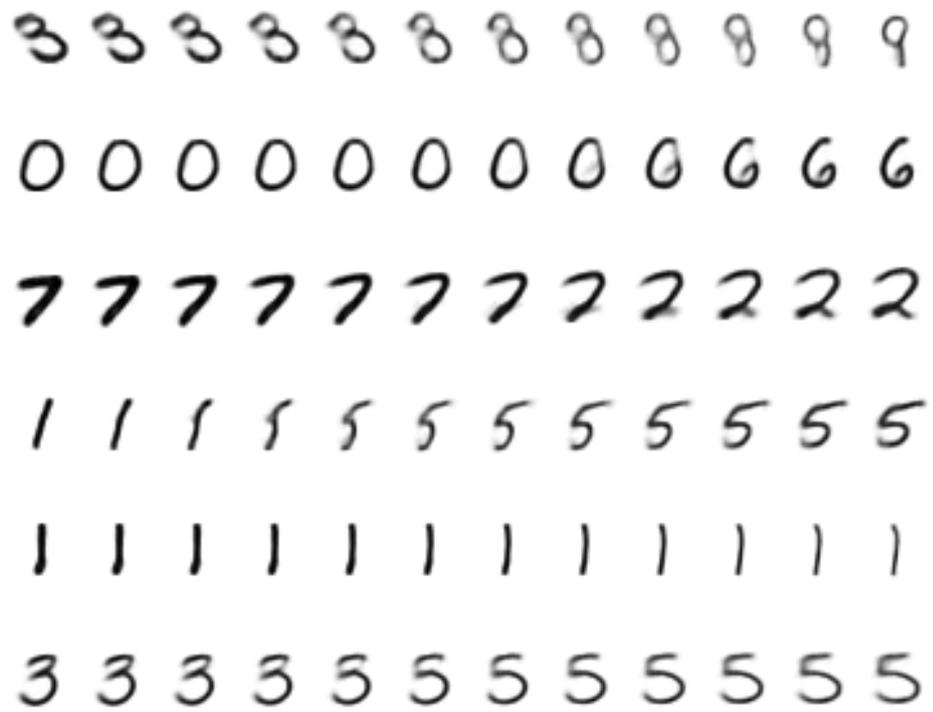
$$\forall x, x' \in \mathcal{X}^2, \alpha \in [0, 1], \xi(x, x', \alpha) = g((1 - \alpha)f(x) + \alpha f(x')).$$



PCA interpolation ( $d = 32$ )



Autoencoder interpolation ( $d = 8$ )

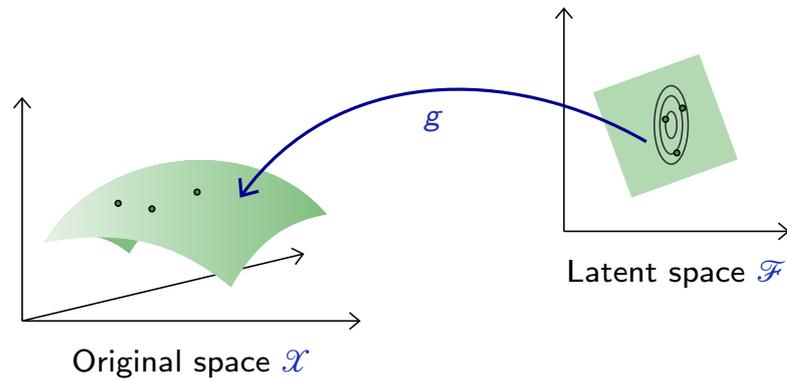


And we can assess the generative capabilities of the decoder  $g$  by introducing a [simple] density model  $q^Z$  over the latent space  $\mathcal{F}$ , sample there, and map the samples into the image space  $\mathcal{X}$  with  $g$ .

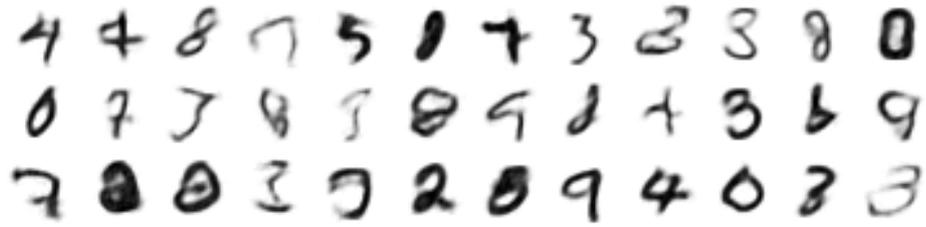
We can for instance use a Gaussian model with diagonal covariance matrix.

$$f(X) \sim \mathcal{N}(\hat{m}, \hat{\Delta})$$

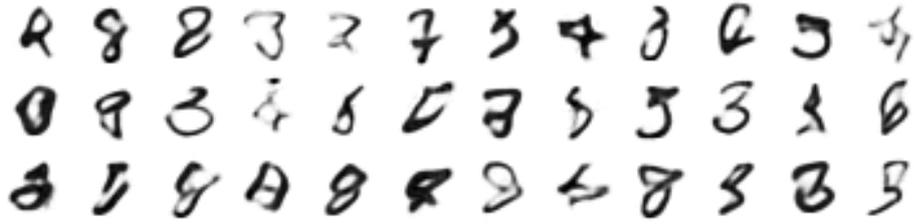
where  $\hat{m}$  is a vector and  $\hat{\Delta}$  a diagonal matrix, both estimated on training data.



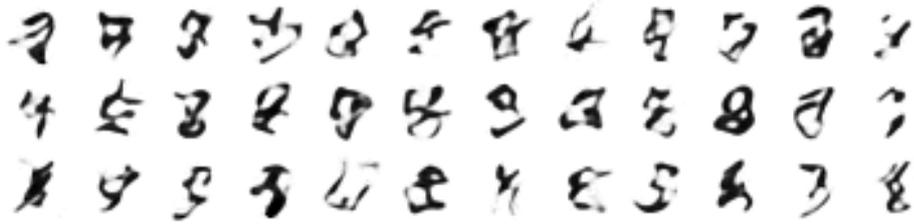
Autoencoder sampling ( $d = 8$ )



Autoencoder sampling ( $d = 16$ )



Autoencoder sampling ( $d = 32$ )



These results are unsatisfying, because the density model used on the latent space  $\mathcal{F}$  is too simple and inadequate.

Building a “good” model amounts to our original problem of modeling an empirical distribution, although it may now be in a lower dimension space.

## References

- H. Bourlard and Y. Kamp. **Auto-association by multilayer perceptrons and singular value decomposition.** Biological Cybernetics, 59(4):291–294, 1988.
- G. E. Hinton and R. S. Zemel. **Autoencoders, minimum description length and helmholtz free energy.** In Neural Information Processing Systems (NIPS), pages 3–10, 1994.