EE-559 – Deep learning

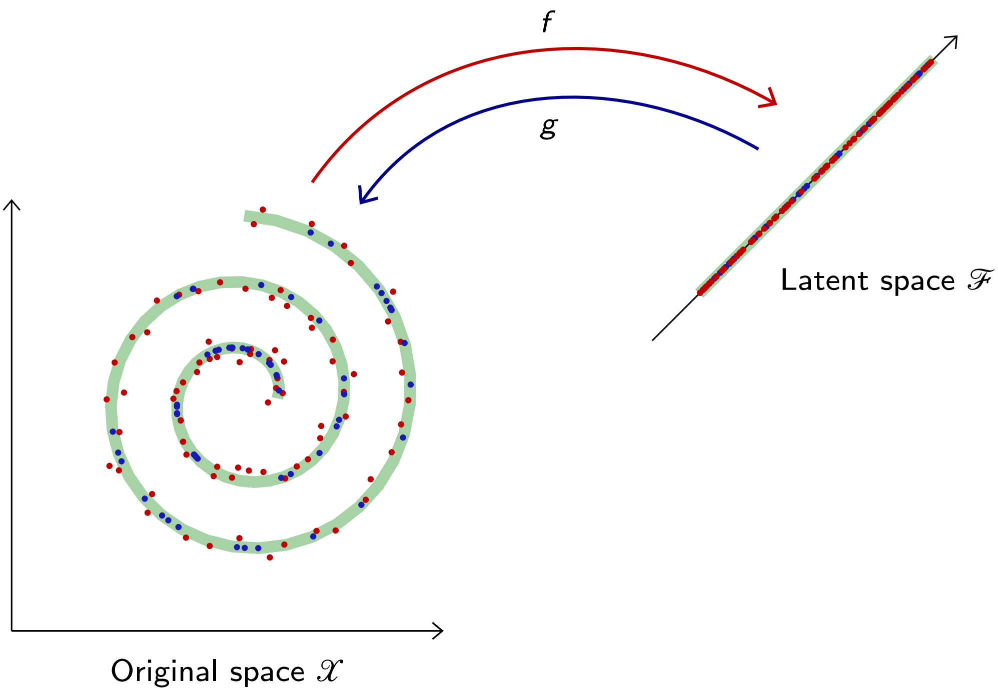
9.2. Autoencoders

François Fleuret
https://fleuret.org/ee559/
Fri Dec 7 23:41:13 UTC 2018

Embeddings and generative models

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.

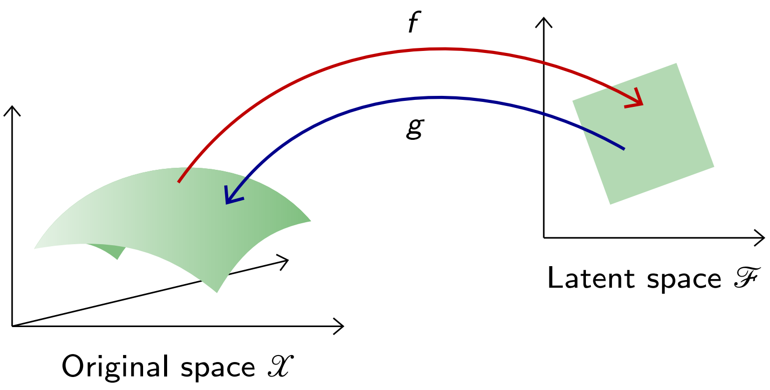


When dealing with real-world signals, this objective involves the same theoretical and practical issues as for classification or regression: defining the right class of high-dimension models, and optimizing them.

Regarding synthesis, we saw that deep feed-forward architectures exhibit good generative properties, which motivates their use explicitly for that purpose.

Autoencoders

An autoencoder (Bourlard and Kamp, 1988; Hinton and Zemel, 1994) combines an **encoder** f from the original space $\mathscr{X}$ to a **latent** space $\mathscr{F}$, and a **decoder** g to map back to $\mathscr{X}$, such that $g \circ f$ is [close to] the identity on the data.



A proper autoencoder has to capture a "good" parametrization of the signal, and in particular the statistical dependencies between the signal components.

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

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

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

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

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.

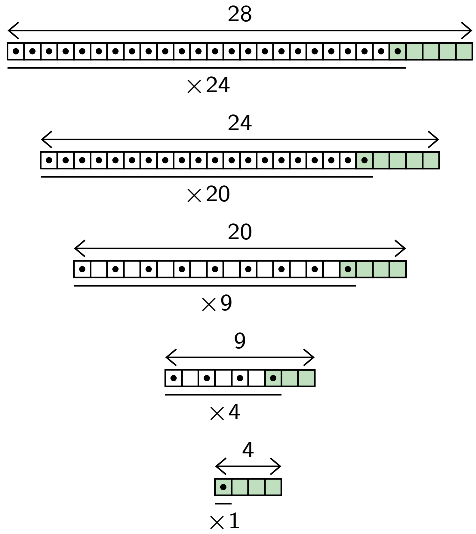
Deep Autoencoders

A deep autoencoder combines an encoder composed of convolutional layers, with a decoder composed of the reciprocal transposed convolution 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))
  )
)
```

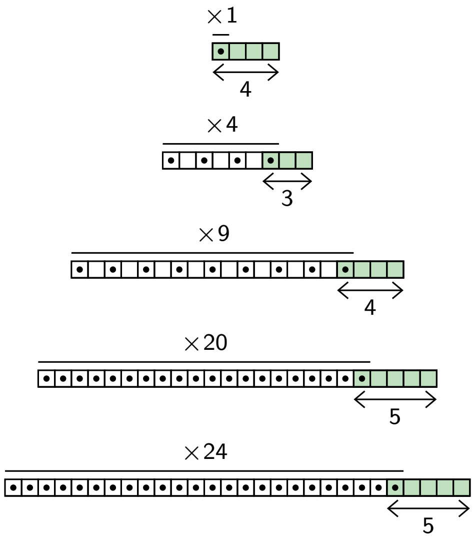
Encoder

Tensor sizes / operations
$1 \times 28 \times 28$
`nn.Conv2d(1, 32, kernel_size=5, stride=1)`
$32 \times 24 \times 24$
`nn.Conv2d(32, 32, kernel_size=5, stride=1)`
$32 \times 20 \times 20$
`nn.Conv2d(32, 32, kernel_size=4, stride=2)`
$32 \times 9 \times 9$
`nn.Conv2d(32, 32, kernel_size=3, stride=2)`
$32 \times 4 \times 4$
`nn.Conv2d(32, 8, kernel_size=4, stride=1)`
$8 \times 1 \times 1$



Decoder

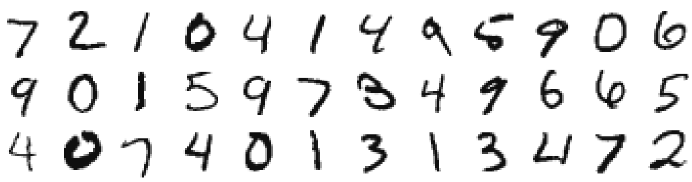
Tensor sizes / operations
$8 \times 1 \times 1$
`nn.ConvTranspose2d(8, 32, kernel_size=4, stride=1)`
$32 \times 4 \times 4$
`nn.ConvTranspose2d(32, 32, kernel_size=3, stride=2)`
$32 \times 9 \times 9$
`nn.ConvTranspose2d(32, 32, kernel_size=4, stride=2)`
$32 \times 20 \times 20$
`nn.ConvTranspose2d(32, 32, kernel_size=5, stride=1)`
$32 \times 24 \times 24$
`nn.ConvTranspose2d(32, 1, kernel_size=5, stride=1)`
$1 \times 28 \times 28$



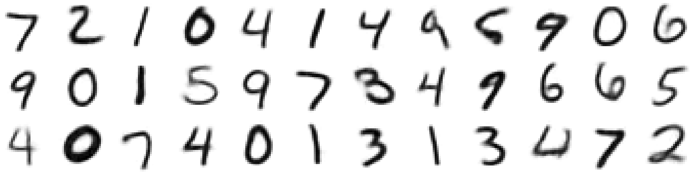
Training is achieved with quadratic loss and Adam

```
model = AutoEncoder(nb_channels, embedding_dim)

model.to(device)

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

for epoch in range(args.nb_epochs):
    for input, _ in iter(train_loader):
        input = input.to(device)

        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()
```

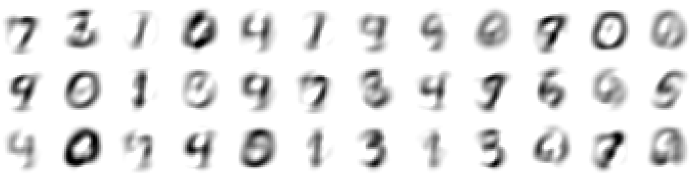

X (original samples)



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

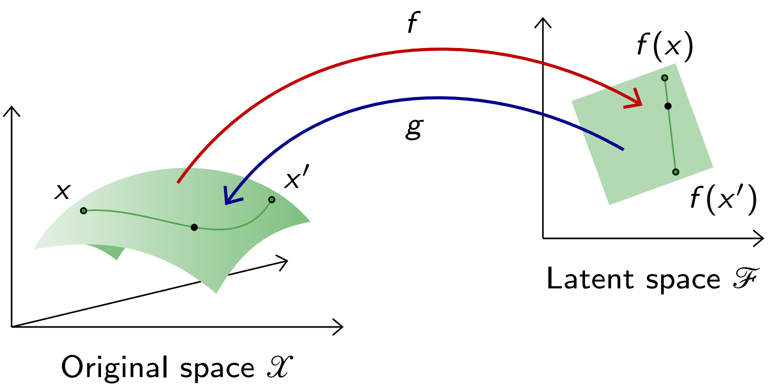


$g \circ f(X)$ (PCA, $d = 8$)



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

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



PCA interpolation ($d = 32$)

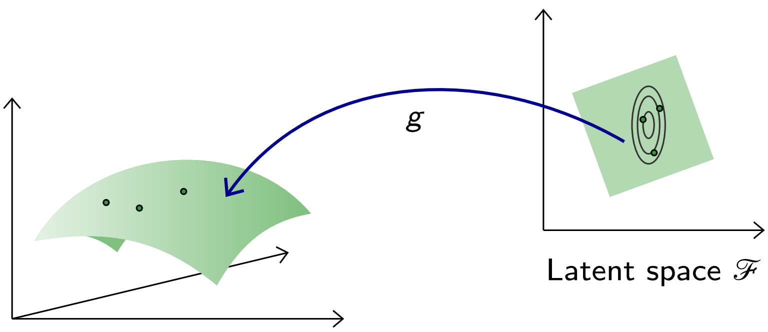
Autoencoder interpolation ($d = 8$)

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

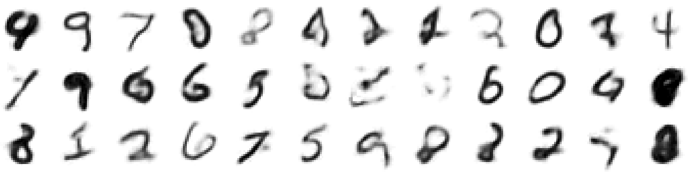
We can for instance use a Gaussian model with diagonal covariance matrix.

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

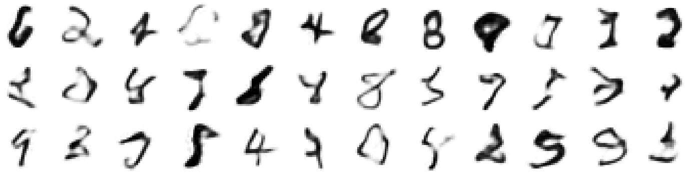
where $\hat{m}$ is a vector and $\hat{\Delta}$ a diagonal matrix, both estimated on training data.



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 $\mathscr{F}$ 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.