HELMHOLTZ AI | ARTIFICIAL INTELLIGENCE COOPERATION UNIT

CERN School of Computing

Transposed Convolutions and Auto-Encoders

Peter Steinbach

Helmholtz-Zentrum Dresden-Rossendorf / 2024-09-19

Today's Agenda

1. More Convolutions
2. autoencoders

More Convolutions

Recap a convolution in 1D

1D Convolutions

Convolution layer

Input

1	4	-1	0	2	-2	1	3	3	1

W

from [Fle22]

1D Convolutions

Convolution layer

Input

1	4	-1	0	2	-2	1	3	3	1

W

Kernel

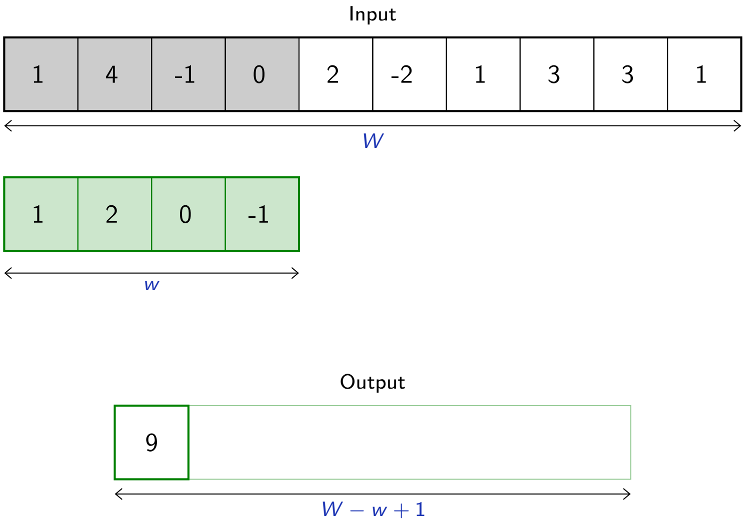
1	2	0	-1

w

from [Fle22]

1D Convolutions

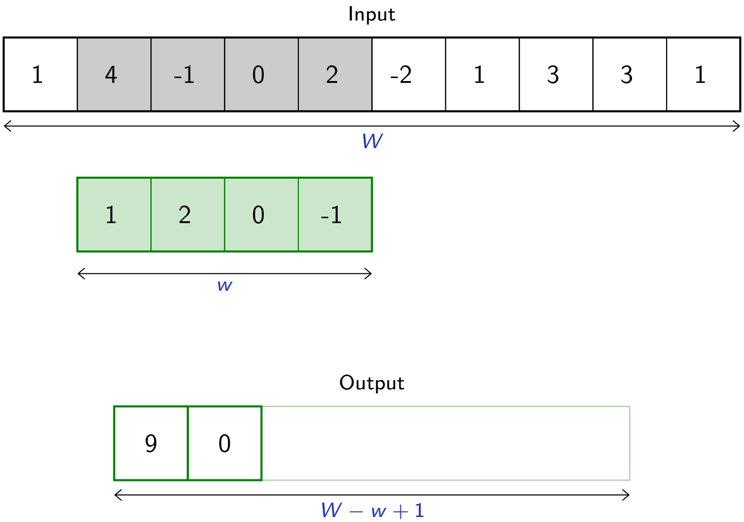
Convolution layer



from [Fle22]

1D Convolutions

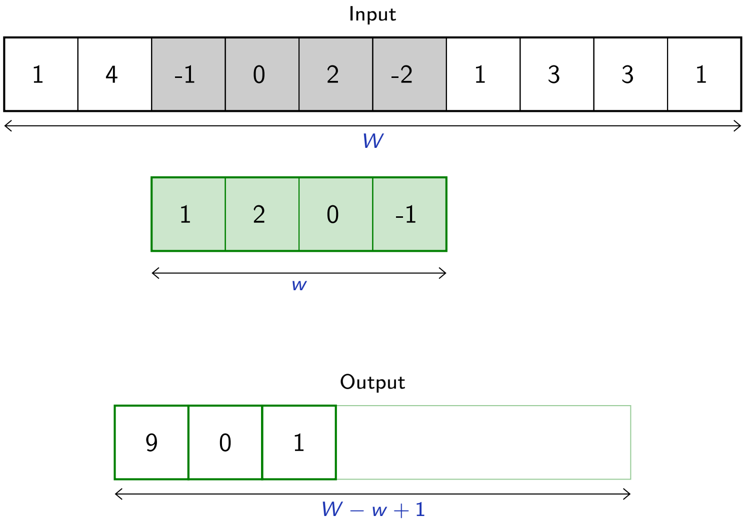
Convolution layer



from [Fle22]

1D Convolutions

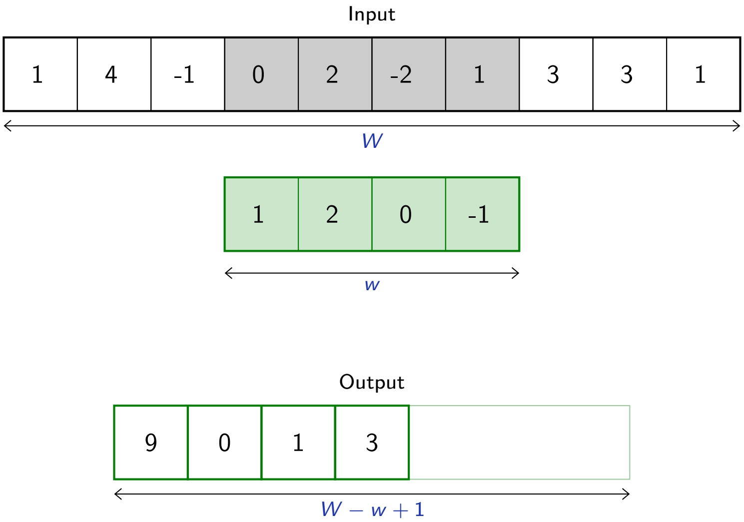
Convolution layer



from [Fle22]

1D Convolutions

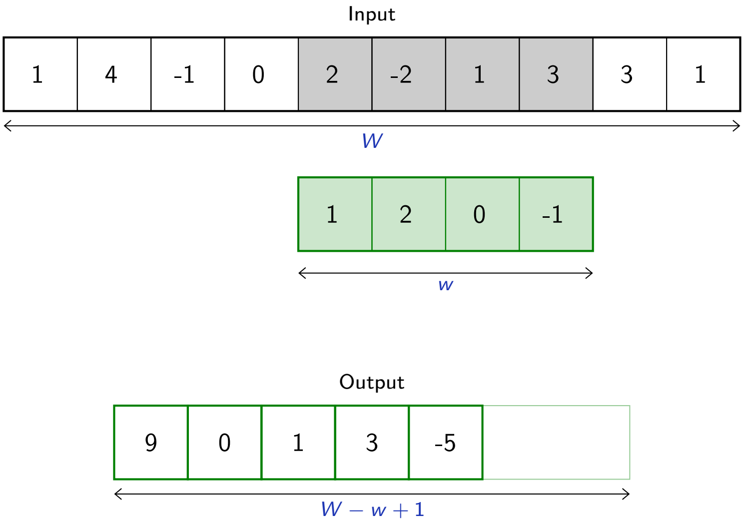
Convolution layer



from [Fle22]

1D Convolutions

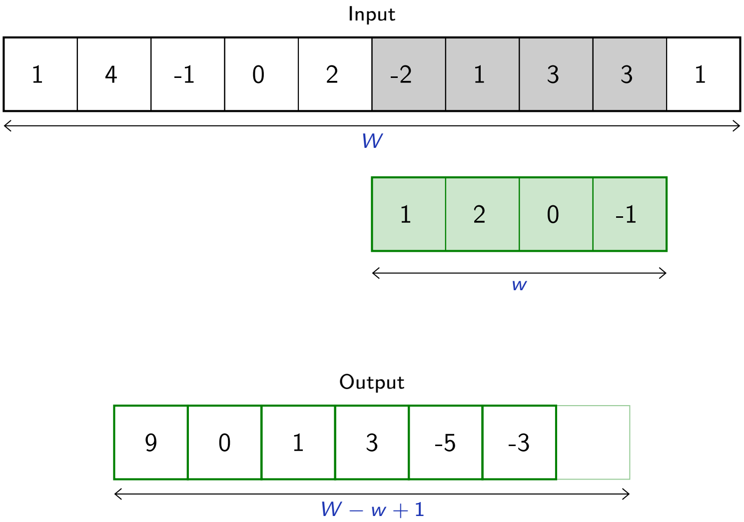
Convolution layer



from [Fle22]

1D Convolutions

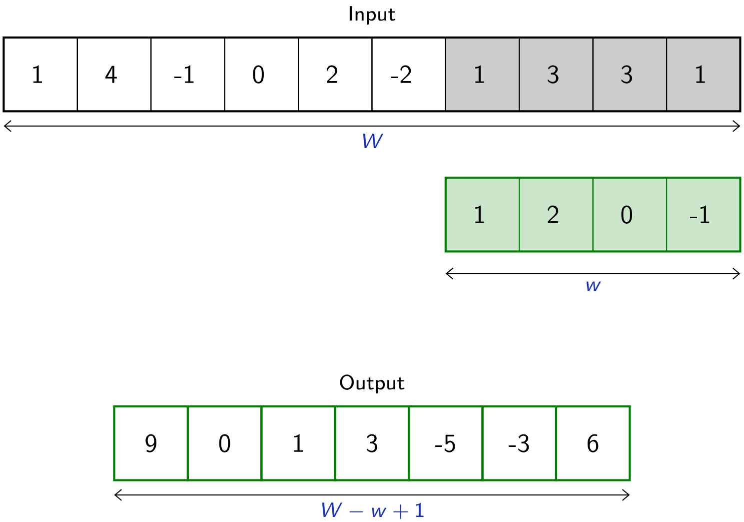
Convolution layer



from [Fle22]

1D Convolutions

Convolution layer



from [Fle22]

1D Convolutions

Convolution layer

Input

1	4	-1	0	2	-2	1	3	3	1

W

Kernel

1	2	0	-1

w

Output

9	0	1	3	-5	-3	6

$W - w + 1$

from [Fle22]

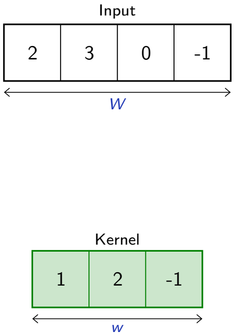
Transposed Convolutions

In the deep-learning field, since it corresponds to transposing the weight matrix of the equivalent fully-connected layer, it is called a **transposed convolution**.

$$\begin{pmatrix} \kappa_1 & \kappa_2 & \kappa_3 & 0 & 0 & 0 & 0 \\ 0 & \kappa_1 & \kappa_2 & \kappa_3 & 0 & 0 & 0 \\ 0 & 0 & \kappa_1 & \kappa_2 & \kappa_3 & 0 & 0 \\ 0 & 0 & 0 & \kappa_1 & \kappa_2 & \kappa_3 & 0 \\ 0 & 0 & 0 & 0 & \kappa_1 & \kappa_2 & \kappa_3 \end{pmatrix}^\top = \begin{pmatrix} \kappa_1 & 0 & 0 & 0 & 0 \\ \kappa_2 & \kappa_1 & 0 & 0 & 0 \\ \kappa_3 & \kappa_2 & \kappa_1 & 0 & 0 \\ 0 & \kappa_3 & \kappa_2 & \kappa_1 & 0 \\ 0 & 0 & \kappa_3 & \kappa_2 & \kappa_1 \\ 0 & 0 & 0 & \kappa_3 & \kappa_2 \\ 0 & 0 & 0 & 0 & \kappa_3 \end{pmatrix}$$

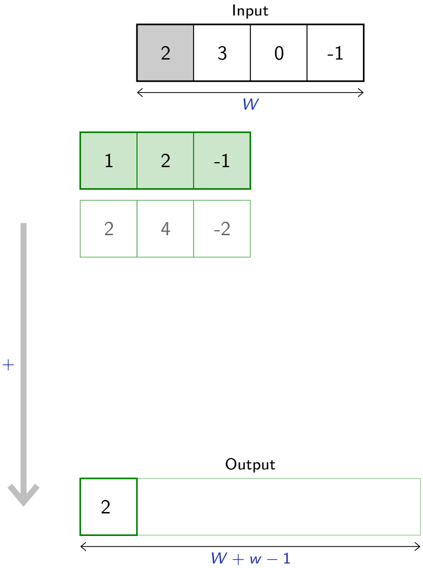
A convolution can be seen as a series of inner products, a transposed convolution can be seen as a weighted sum of translated kernels.

from [Fle22]

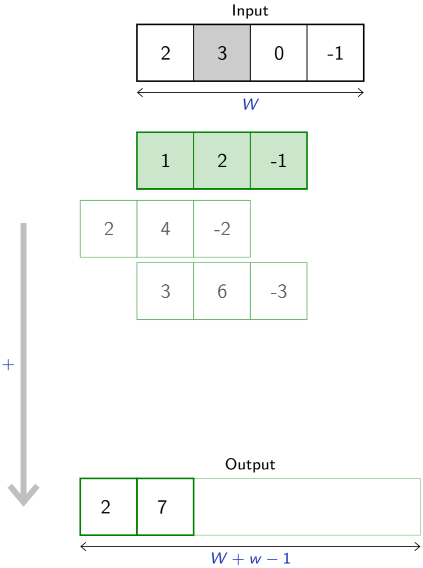


from [Fle22]

Transposed convolution layer

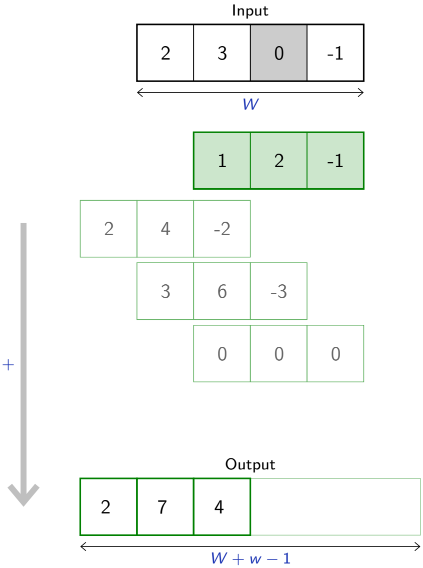


from [Fle22]

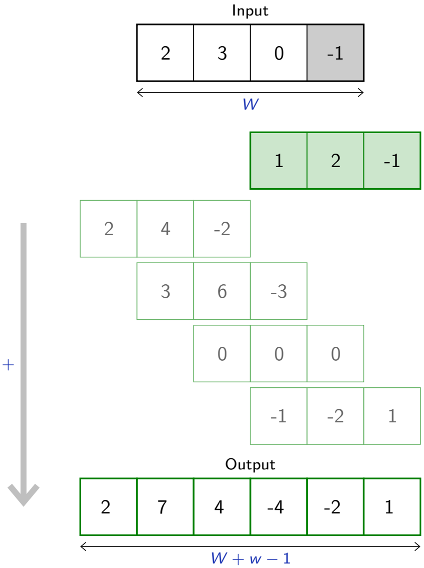


from [Fle22]

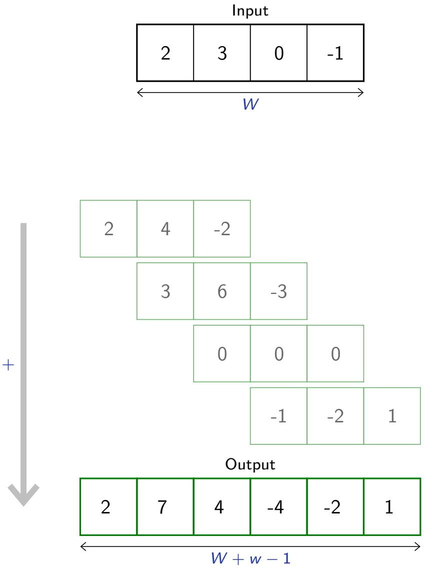
Transposed convolution layer



from [Fle22]

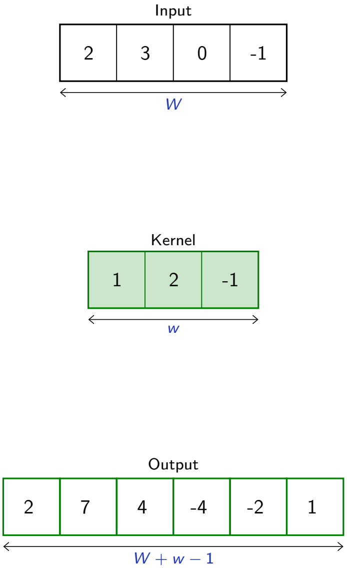


from [Fle22]



from [Fle22]

Transposed convolution layer



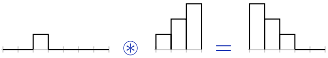
from [Fle22]

Pytorch

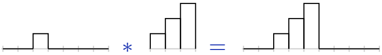
`F.conv_transpose1d` implements the operation we just described. It takes as input a batch of multi-channel samples, and produces a batch of multi-channel samples.

We can compare on a simple 1d example the results of a standard and a transposed convolution:

```
>>> x = torch.tensor([[[0., 0., 1., 0., 0., 0., 0.]]])
>>> k = torch.tensor([[[1., 2., 3.]]])
>>> F.conv1d(x, k)
tensor([[[ 3.,  2.,  1.,  0.,  0.]]])
```



```
>>> F.conv_transpose1d(x, k)
tensor([[[ 0.,  0.,  1.,  2.,  3.,  0.,  0.,  0.,  0.]]])
```



from [Fle22]

stride, padding, dilation in transposed convolition

Transposed convolutions also have a `dilation` parameter that behaves as for convolution and expends the kernel size without increasing the number of parameters by making it sparse.

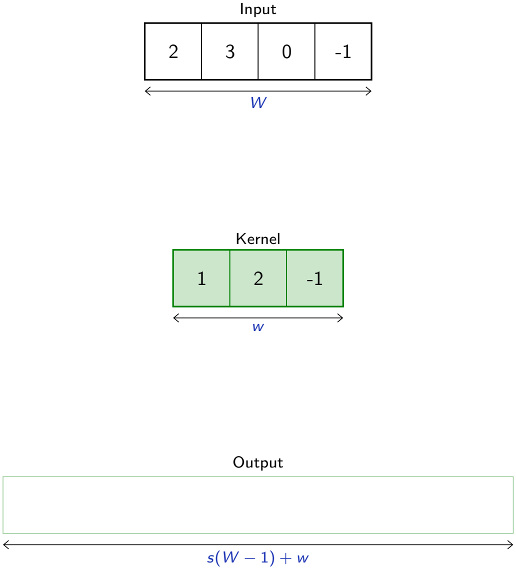
They also have a `stride` and `padding` parameters, however, due to the relation between convolutions and transposed convolutions:

While for convolutions `stride` and `padding` are defined in the input map, for transposed convolutions these parameters are defined in the output map, and the latter modulates a cropping operation.

from [Fle22]

1D: with strides

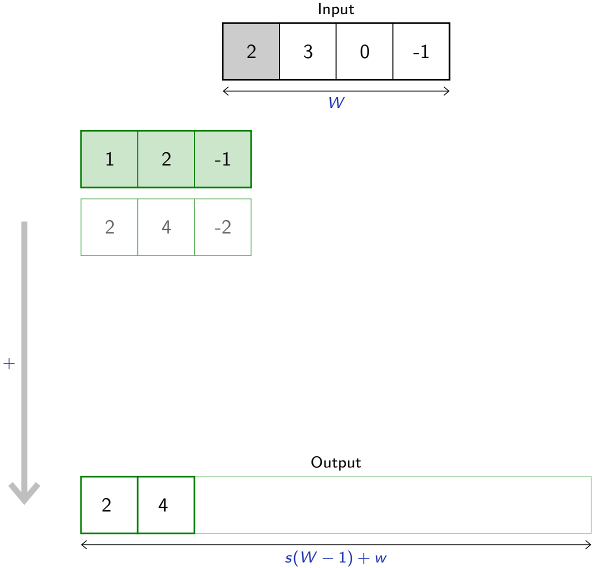
Transposed convolution layer (stride $= 2$)



from [Fle22]

1D: with strides

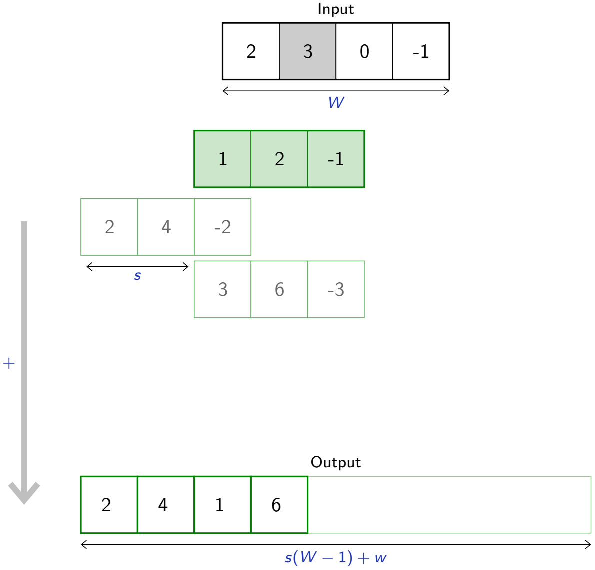
Transposed convolution layer (stride $= 2$)



from [Fle22]

1D: with strides

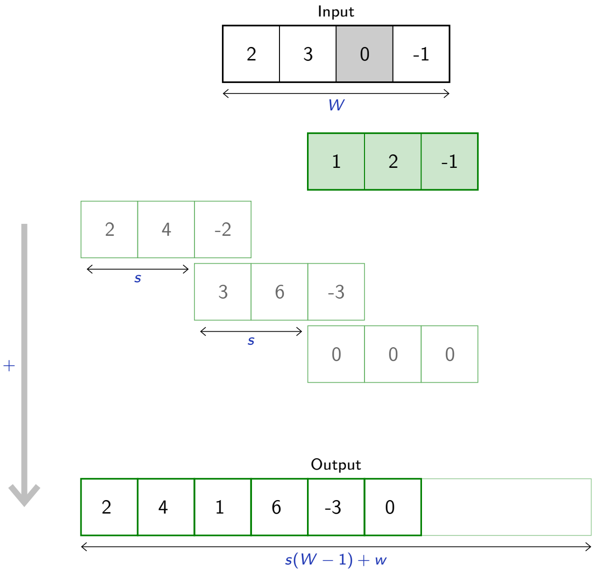
Transposed convolution layer (stride $= 2$)



from [Fle22]

1D: with strides

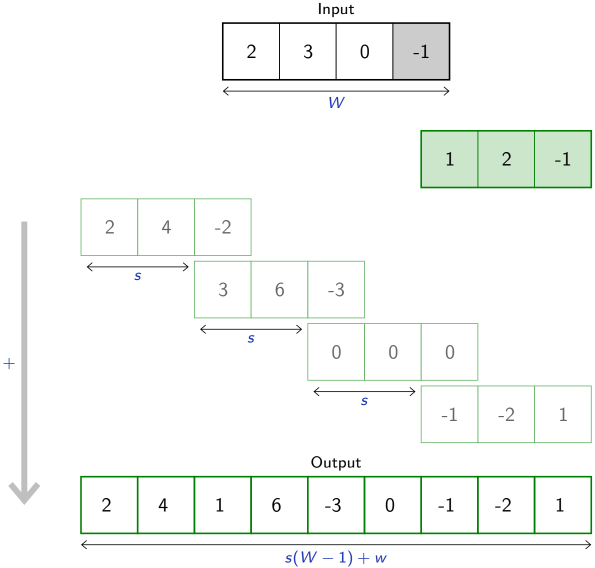
Transposed convolution layer (stride $= 2$)



from [Fle22]

1D: with strides

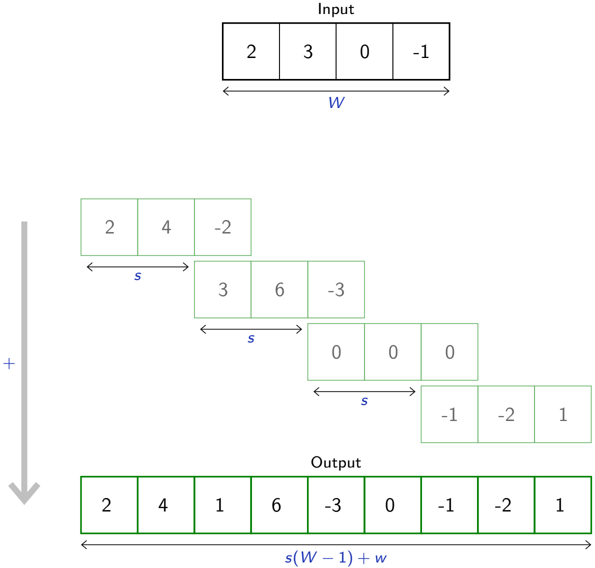
Transposed convolution layer (stride $= 2$)



from [Fle22]

1D: with strides

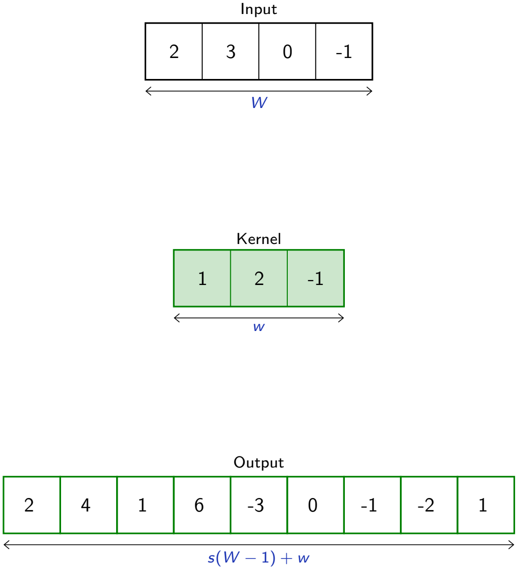
Transposed convolution layer (stride $= 2$)



from [Fle22]

1D: with strides

Transposed convolution layer (stride $= 2$)



from [Fle22]

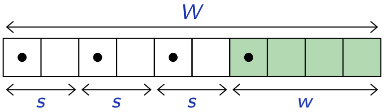
Convolution and Transposed Convolition back-to-back

The composition of a convolution and a transposed convolution of same parameters keep the signal size [roughly] unchanged.

A convolution with a stride greater than one may ignore parts of the signal. Its composition with the corresponding transposed convolution generates a map **of the size of the observed area**.

For instance, a 1d convolution of kernel size w and stride s composed with the transposed convolution of same parameters maintains the signal size W, only if

$$\exists q \in \mathbb{N}, \ \ W = w + s\,q.$$

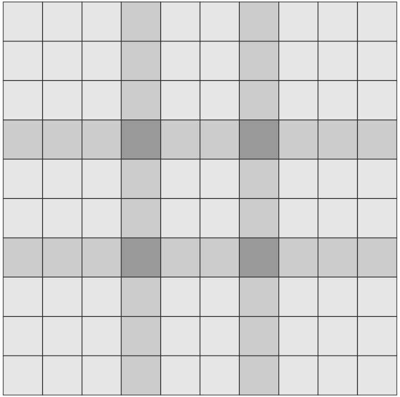


from [Fle22]

Artefacts

It has been observed that transposed convolutions may create some grid-structure artifacts, since generated pixels are not all covered similarly.

For instance with a 4×4 kernel and stride 3



from [Fle22]

Equivalent Operations: Transposed Conv or Conv+Interpolation

An alternative is to use an analytic up-scaling, implemented in the PyTorch functional `F.interpolate`.

```
>>> x = torch.tensor([[[[ 1., 2. ], [ 3., 4. ]]]])
>>> F.interpolate(x, scale_factor = 3, mode = 'bilinear')
tensor([[[[1.0000, 1.0000, 1.3333, 1.6667, 2.0000, 2.0000],
          [1.0000, 1.0000, 1.3333, 1.6667, 2.0000, 2.0000],
          [1.6667, 1.6667, 2.0000, 2.3333, 2.6667, 2.6667],
          [2.3333, 2.3333, 2.6667, 3.0000, 3.3333, 3.3333],
          [3.0000, 3.0000, 3.3333, 3.6667, 4.0000, 4.0000],
          [3.0000, 3.0000, 3.3333, 3.6667, 4.0000, 4.0000]]]])

>>> F.interpolate(x, scale_factor = 3, mode = 'nearest')
tensor([[[[1., 1., 1., 2., 2., 2.],
          [1., 1., 1., 2., 2., 2.],
          [1., 1., 1., 2., 2., 2.],
          [3., 3., 3., 4., 4., 4.],
          [3., 3., 3., 4., 4., 4.],
          [3., 3., 3., 4., 4., 4.]]]])
```

from [Fle22]

autoencoders

autoencoders: Goals

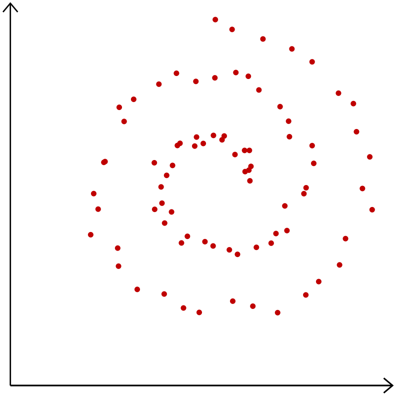
Many applications such as image synthesis, denoising, super-resolution, speech synthesis, compression, etc. require to go beyond classification and regression, and **model explicitly a high dimension signal**.

autoencoders: Goals

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**.

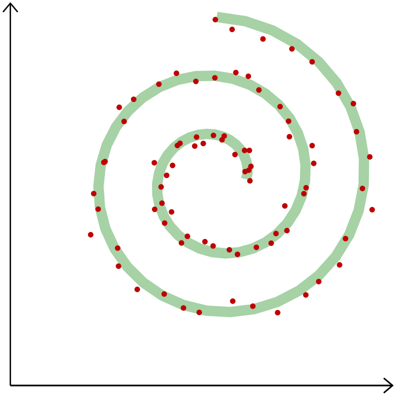
A complex signal



Original space $\mathscr{X}$

from [Fle22]

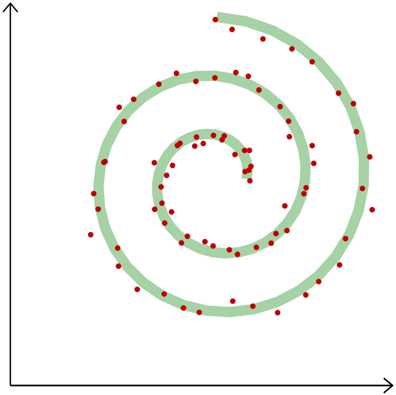
A complex signal has hidden structure

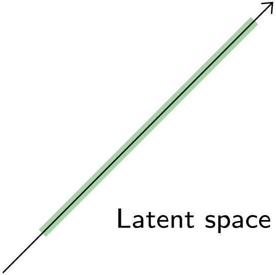


Original space $\mathscr{X}$

from [Fle22]

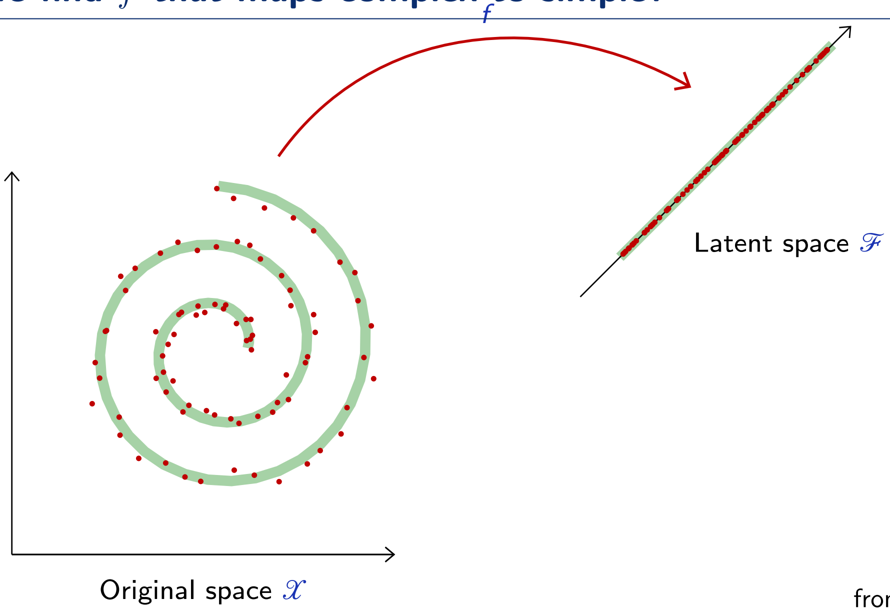
A latent space is simpler





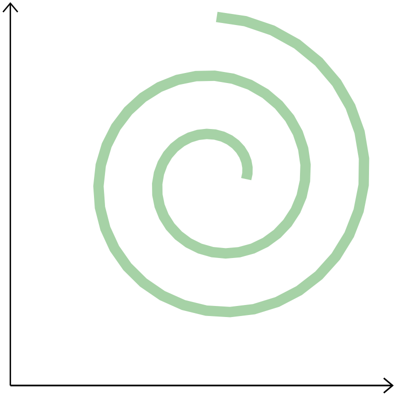
from [Fle22]

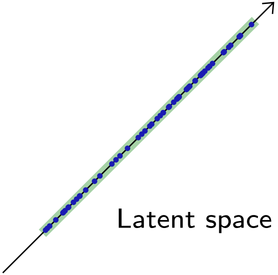
Can we find f that maps complex to simple?



from [Fle22]

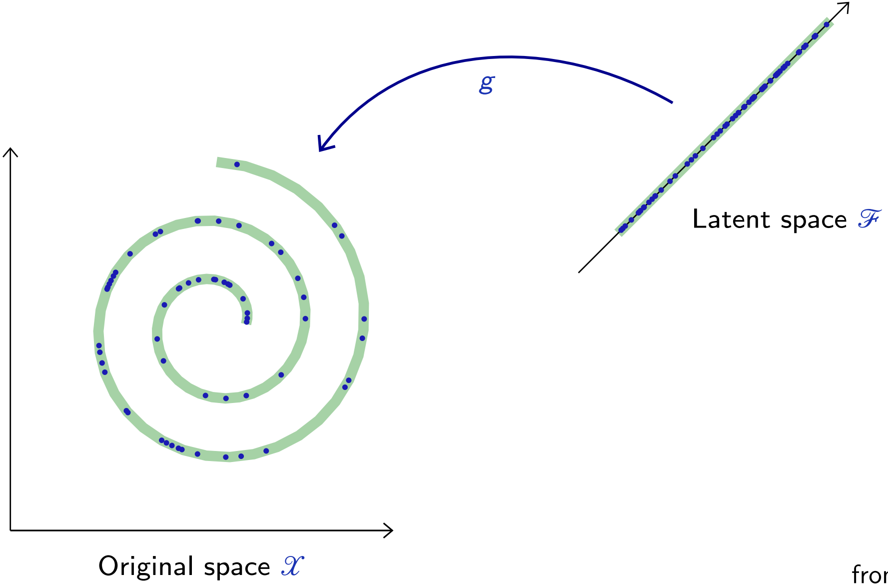
represent complex data in a simpler way





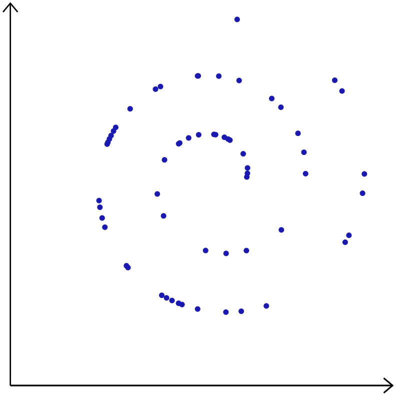
from [Fle22]

And for f, can we find g to invert that mapping?



from [Fle22]

Reconstruct the complex signal



Original space $\mathscr{X}$

from [Fle22]

autoencoders: Objectives

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

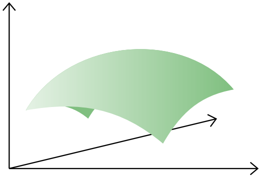
autoencoders: Objectives

When dealing with real-world signals, the objective of autoencoders 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*. This is called a **generative model**!

autoencoders: Definition [BK88] [HZ93]

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 $\mathscr{X}$ to a **latent** space $\mathscr{F}$, and a **decoder** g to map back to $\mathscr{X}$ (Bourlard and Kamp, 1988; Hinton and Zemel, 1994).

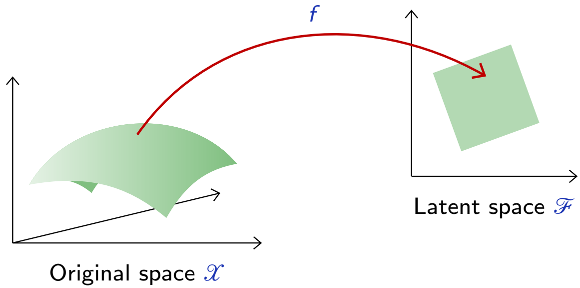


Original space $\mathscr{X}$

autoencoders: Definition [BK88] [HZ93]

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 $\mathscr{X}$ to a **latent** space $\mathscr{F}$, and a **decoder** g to map back to $\mathscr{X}$ (Bourlard and Kamp, 1988; Hinton and Zemel, 1994).



autoencoders: Loss functions

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

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

from [Fle22]

autoencoders: Loss functions

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

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

from [Fle22]

autoencoders: Loss functions

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 $f(\cdot\,; w_f)$ and $g(\cdot\,; w_g)$, 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.

from [Fle22]

autoencoders: Loss functions

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 $f(\cdot\,; w_f)$ and $g(\cdot\,; w_g)$, 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.

from [Fle22]

deep autoencoders: an example with MNIST[1]

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))
  )
)
```

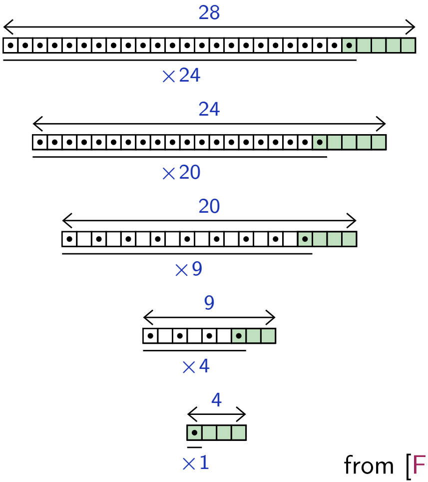
from [Fle22]

[1] 28×28 greyscale images of handwritten digits.

autoencoders

Encoder

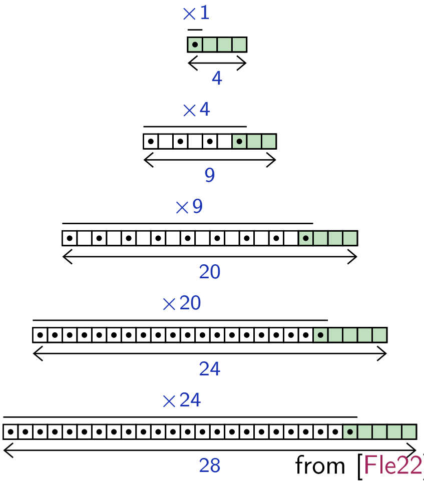
Tensor sizes / operations
$1 \times 28 \times 28$
`nn.Conv2d(1, 32, kernel_size=5, stride=1)`
$32 \times 24 \times 24$
`nn.Conv2d(32, 32, kernel_size=5, stride=1)`
$32 \times 20 \times 20$
`nn.Conv2d(32, 32, kernel_size=4, stride=2)`
$32 \times 9 \times 9$
`nn.Conv2d(32, 32, kernel_size=3, stride=2)`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



from [Fle22]

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)`
$1 \times 28 \times 28$



from [Fle22]

autoencoders in pytorch

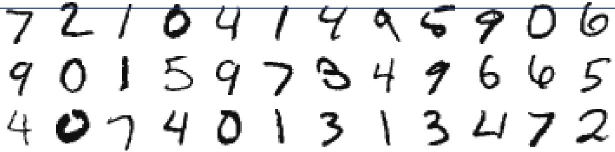
Training is achieved with quadratic loss and Adam

```
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()
```

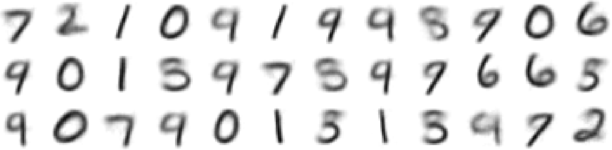
from [Fle22]

Results

X (original samples)



$g \circ f(X)$ (CNN, $d = 2$)

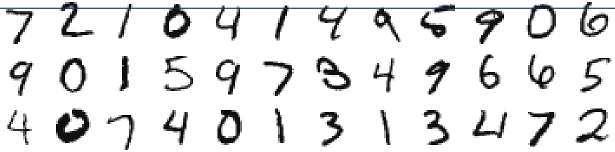


$g \circ f(X)$ (PCA, $d = 2$)

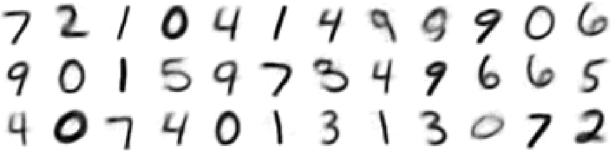
from [Fle22]

Results

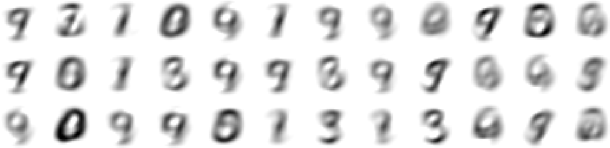
X (original samples)



$g \circ f(X)$ (CNN, $d = 4$)



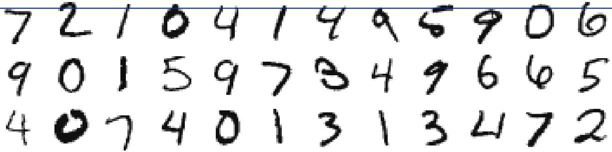
$g \circ f(X)$ (PCA, $d = 4$)



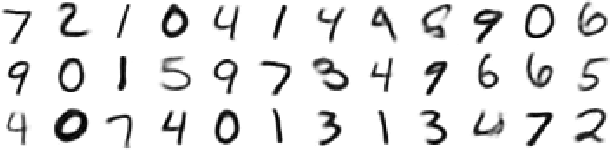
from [Fle22]

Results

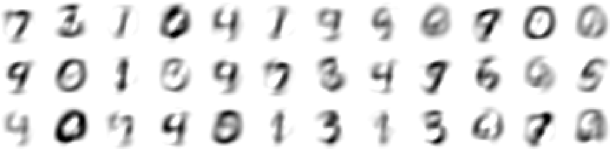
X (original samples)



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



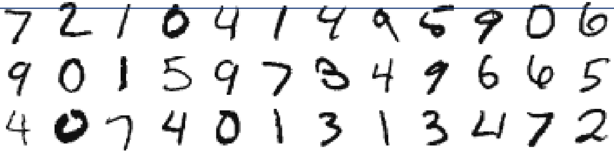
$g \circ f(X)$ (PCA, $d = 8$)



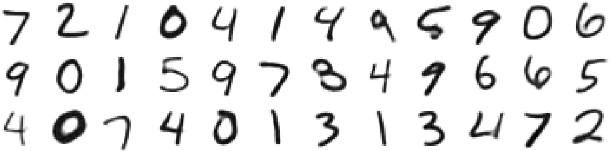
from [Fle22]

Results

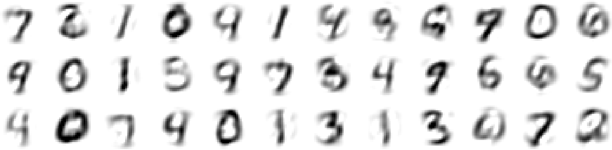
X (original samples)



$g \circ f(X)$ (CNN, $d = 16$)



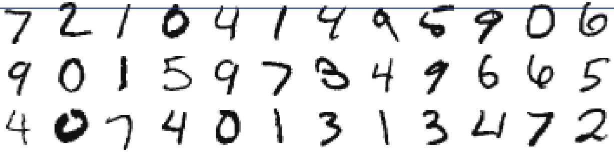
$g \circ f(X)$ (PCA, $d = 16$)



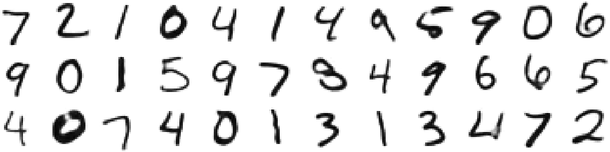
from [Fle22]

Results

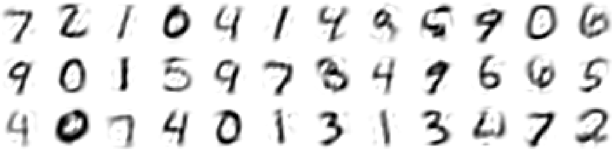
X (original samples)



$g \circ f(X)$ (CNN, $d = 32$)



$g \circ f(X)$ (PCA, $d = 32$)

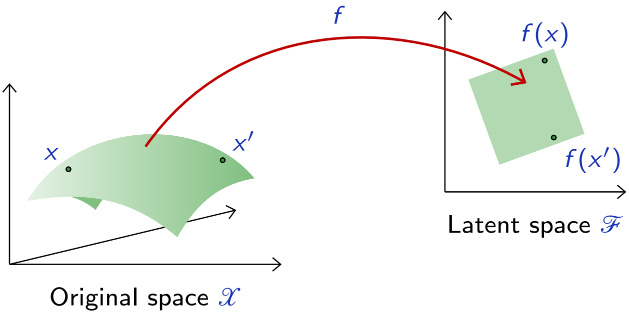


from [Fle22]

Exploring the latent space

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

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

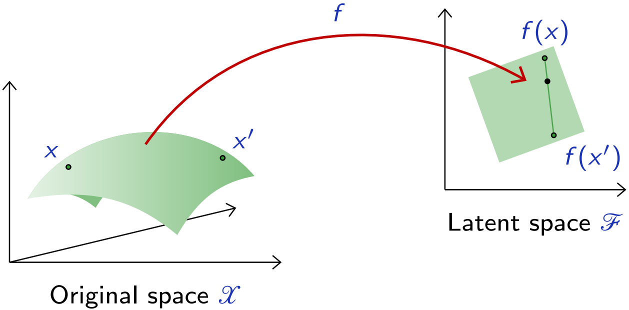


from [Fle22]

Exploring the latent space

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

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

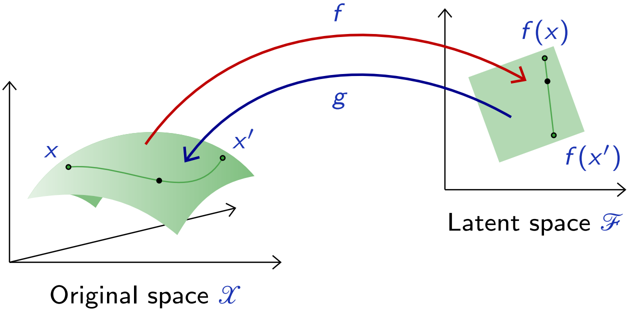


from [Fle22]

Exploring the latent space

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

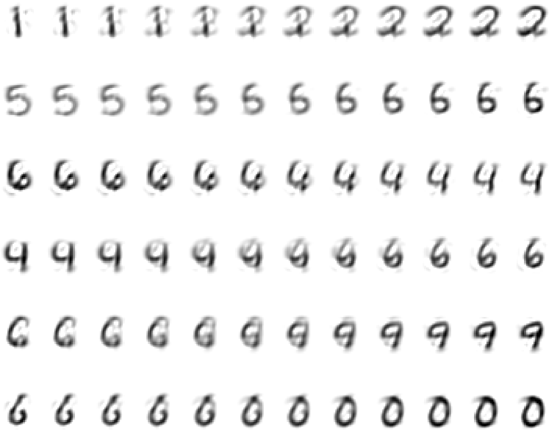
$$\forall x, x' \in \mathscr{X}^2,\ \ \alpha \in [0,1],\ \ \ \xi(x, x', \alpha) = g((1-\alpha)f(x) + \alpha f(x')).$$



from [Fle22]

Exploring the latent space: results

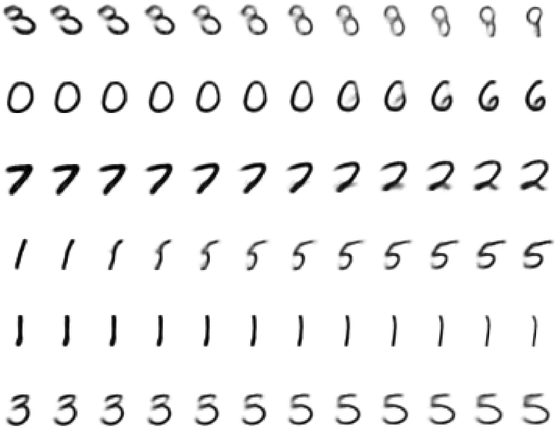
PCA interpolation ($d = 32$)



from [Fle22]

Exploring the latent space: results

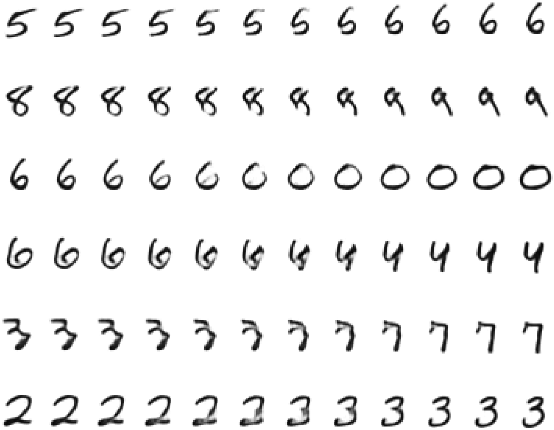
Autoencoder interpolation ($d = 8$)



from [Fle22]

Exploring the latent space: results

Autoencoder interpolation ($d = 32$)



from [Fle22]

Sample the latent space

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.

from [Fle22]

Sample the latent space

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.

from [Fle22]

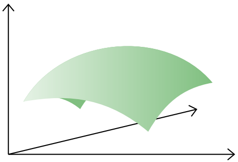
Sample the latent space

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.

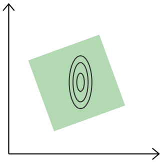
We can for instance use a Gaussian model with diagonal covariance matrix.

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

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



Original space $\mathscr{X}$



Latent space $\mathscr{F}$

from [Fle22]

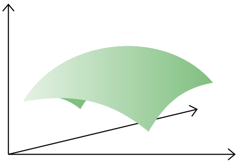
Sample the latent space

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.

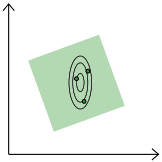
We can for instance use a Gaussian model with diagonal covariance matrix.

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

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



Original space $\mathscr{X}$



Latent space $\mathscr{F}$

from [Fle22]

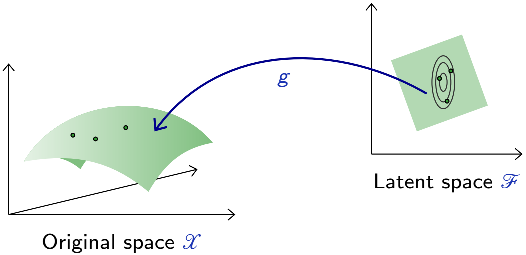
Sample the latent space

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.

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

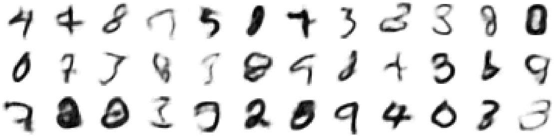
where $\hat{m}$ is a vector and $\hat{\Delta}$ a diagonal matrix, both estimated on training data.



from [Fle22]

Results ☹

Autoencoder sampling ($d = 8$)



Autoencoder sampling ($d = 16$)

Autoencoder sampling ($d = 32$)

from [Fle22]

Unsatisfying Results?

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.

from [Fle22]

Take Aways and Parting Thoughts

Transposed Convolutions and Upscaling

- complement regular convolutions
- either use transposed conv or conv+interpolate to upscale

Take Aways and Parting Thoughts

Transposed Convolutions and Upscaling

- complement regular convolutions
- either use transposed conv or conv+interpolate to upscale

Auto Encoders

- popular architecture with various applications
- latent space can be traversed and worked with
- restoration of many corruptions

Time for some exercises!

References I

[Fle22] F. Fleuret. *Deep Learning Course*. 2022. URL: https://fleuret.org/dlc/.

[BK88] H. Bourland and Y. Kamp. "ªAuto-Association by Multilayer Perceptrons and Singular Value Decomposition, º Biological Cybernetics, vol. 59". In: (1988).

[HZ93] G. E. Hinton and R. Zemel. "Autoencoders, minimum description length and Helmholtz free energy". In: *Advances in neural information processing systems* 6 (1993).