Deep Learning

Ulf Brefeld & Soham Majumder

build: June 3, 2024

Machine Learning Group
Leuphana University of Lüneburg

Transposed Convolutions

So far, we saw neural networks that *reduce* the dimensionality of the data as the inputs are successively transformed to compute the output of the model.

Some models, however, need to *increase* the dimensionality of the information being processed.

Increasing the dimensionality using a fully-connected layer is relatively straightforward. Recall that a fully-connected layer with d_{input} inputs and d_{output} outputs is represented by the transformation

$$x^{(\ell)} = \sigma(Wx^{(\ell-1)} + b)$$

where $W \in \mathbb{R}^{d_{\text{output}} \times d_{\text{input}}}$, $x^{(\ell-1)} \in \mathbb{R}^{d_{\text{input}}}$, and $b \in \mathbb{R}^{d_{\text{output}}}$.

Increasing the dimensionality using a fully-connected layer is relatively straightforward. Recall that a fully-connected layer with d_{input} inputs and d_{output} outputs is represented by the transformation

$$x^{(\ell)} = \sigma(Wx^{(\ell-1)} + b)$$

where $W \in \mathbb{R}^{d_{\text{output}} \times d_{\text{input}}}$, $x^{(\ell-1)} \in \mathbb{R}^{d_{\text{input}}}$, and $b \in \mathbb{R}^{d_{\text{output}}}$.

Thus, we reduce the dimensionality by choosing $d_{\text{input}} > d_{\text{output}}$ and vice-versa.

It is also possible to increase the dimensionality of the output relative to input with a convolution.

To see this, consider how a convolution is represented as a matrix-vector product (recall from Task 17b):

$$x^{(\ell-1)} = \mathrm{vec}\left[\begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \\ x_{10} & x_{11} & x_{12} \end{pmatrix}\right] = \begin{pmatrix} x_1 \\ x_4 \\ x_7 \\ x_{10} \\ \vdots \\ x_9 \\ x_{12} \end{pmatrix}$$

$$Wx^{(\ell-1)} = s^{(\ell)}$$

$$\begin{pmatrix} f_1 & f_2 & f_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & f_1 & f_2 & f_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & f_1 & f_2 & f_3 & 0 \\ 0 & f_1 & f_2 & f_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & f_1 & f_2 & f_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & f_1 & f_2 & f_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_4 \\ x_7 \\ x_{10} \\ \vdots \\ x_9 \\ x_{12} \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{pmatrix}$$

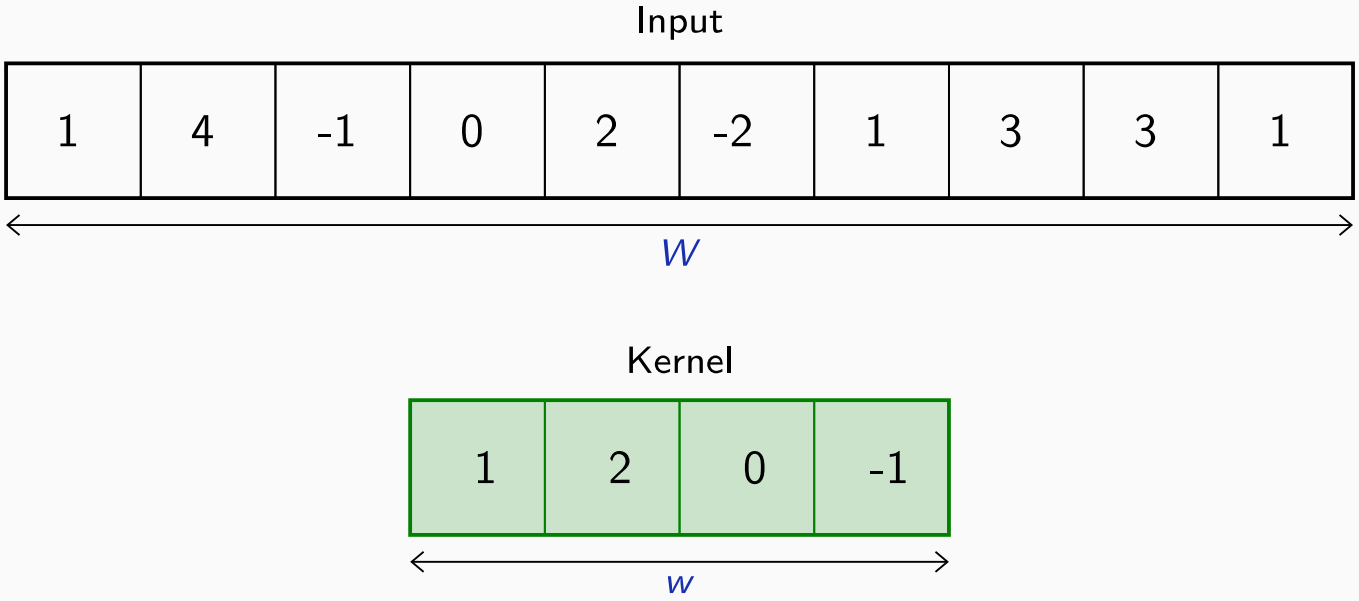
If $W^\top$ is used instead of W, this would represent an increase in the dimensionality of the inputs from 6 to 12:

$$W^\top x'^{(\ell-1)} = s^{(\ell)}$$

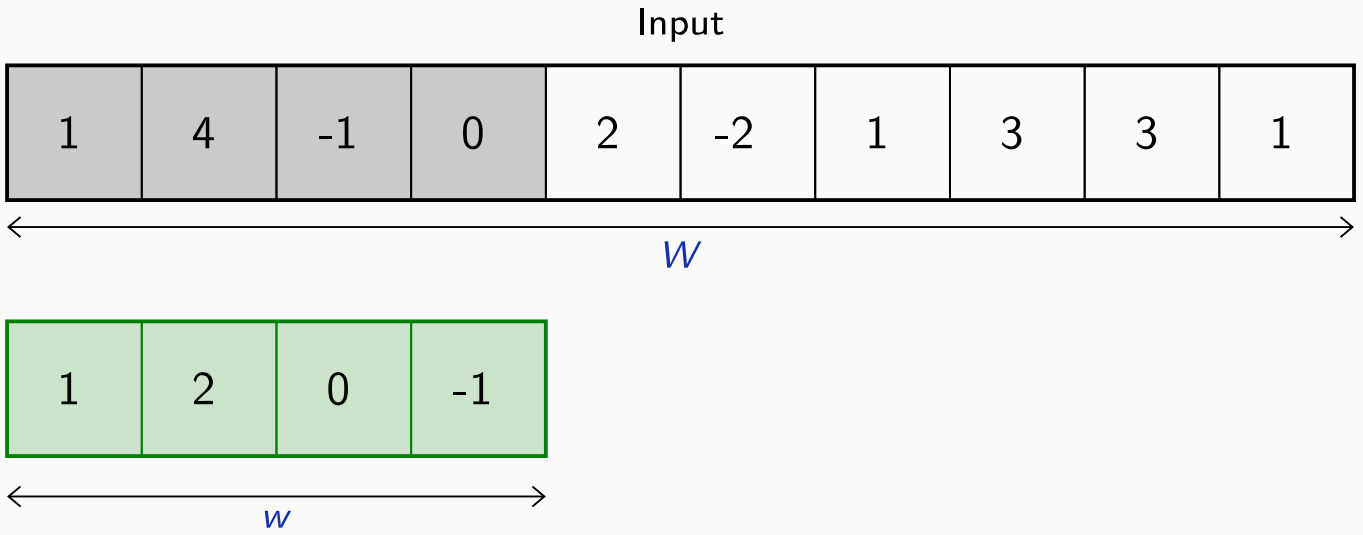
$$\begin{pmatrix} f_1 & 0 & 0 & 0 & 0 \\ f_2 & 0 & 0 & f_1 & 0 \\ f_3 & 0 & 0 & f_2 & 0 \\ 0 & 0 & 0 & f_3 & 0 \\ 0 & f_1 & 0 & 0 & 0 \\ 0 & f_2 & 0 & 0 & 0 \\ 0 & f_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & f_1 & 0 & 0 \\ 0 & 0 & f_2 & 0 & f_1 \\ 0 & 0 & f_3 & 0 & f_2 \\ 0 & 0 & 0 & 0 & f_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_4 \\ x_2 \\ x_5 \\ x_3 \\ x_6 \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ s_{11} \\ s_{12} \end{pmatrix}$$

Due to this “transposed” perspective, the operation equivalent to this is referred to as a *transposed convolution*.

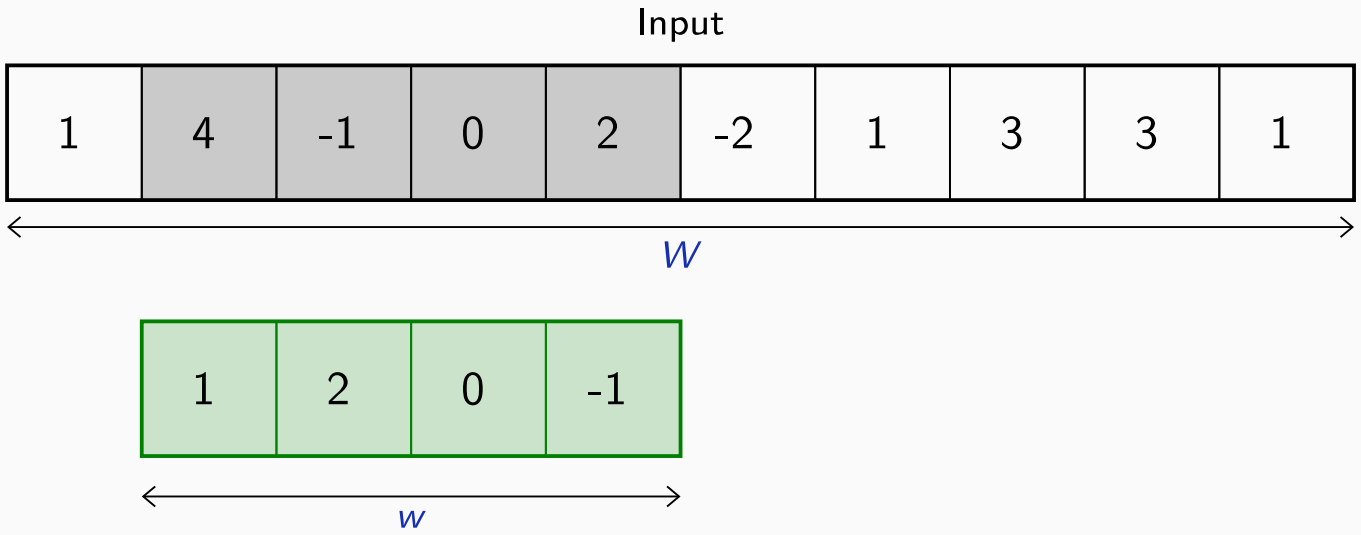
Convolution



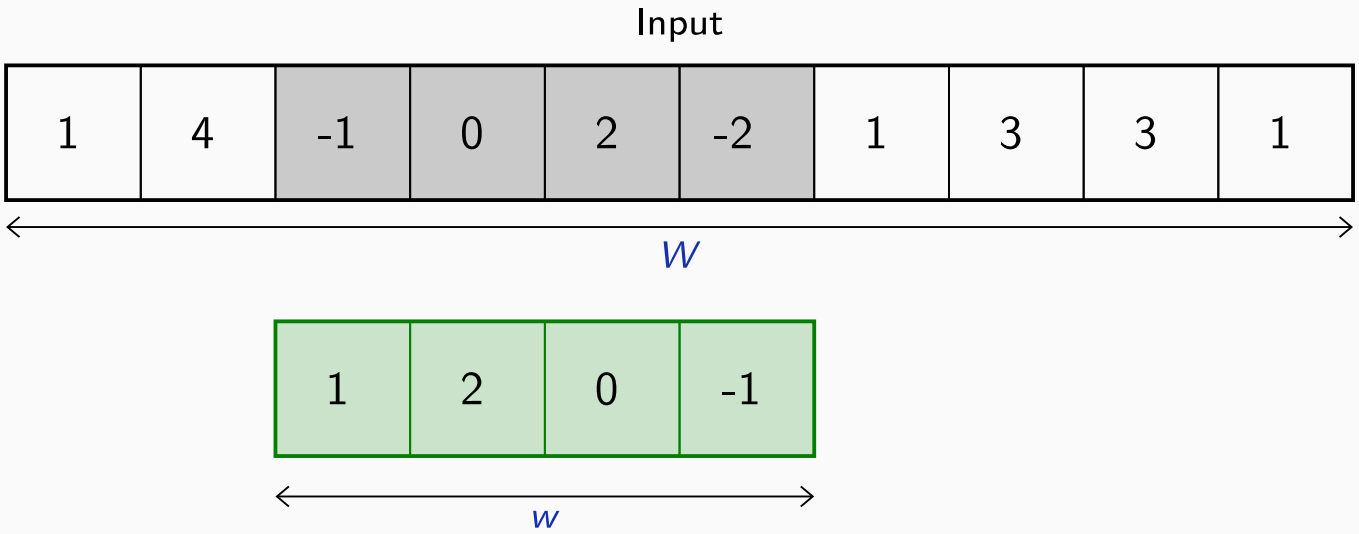
Convolution



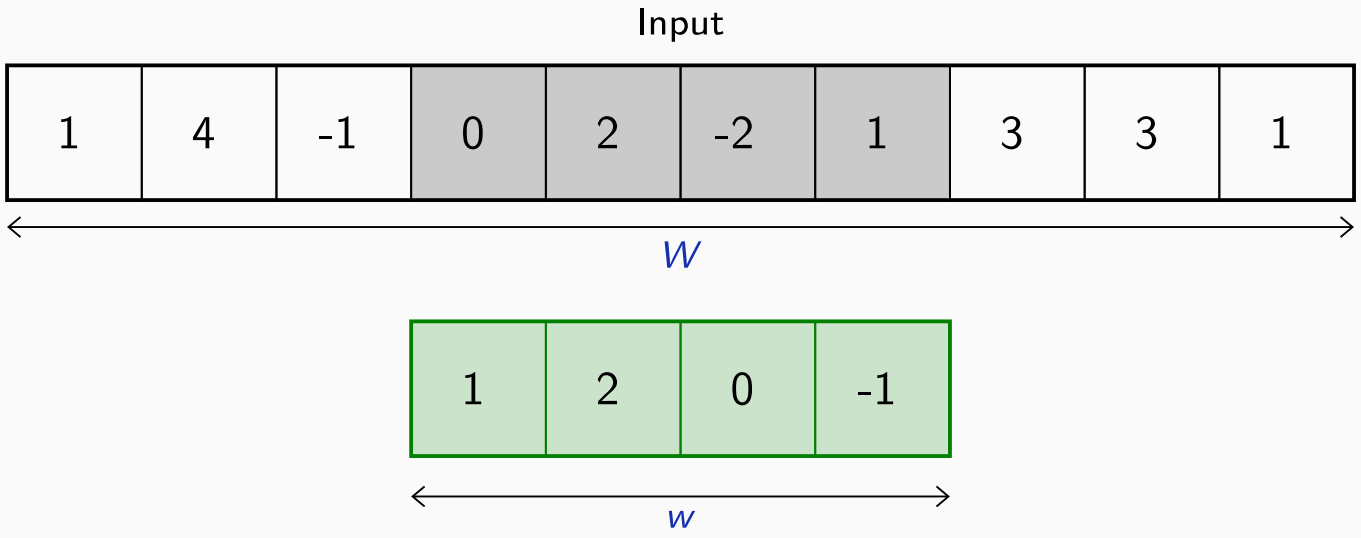
Convolution



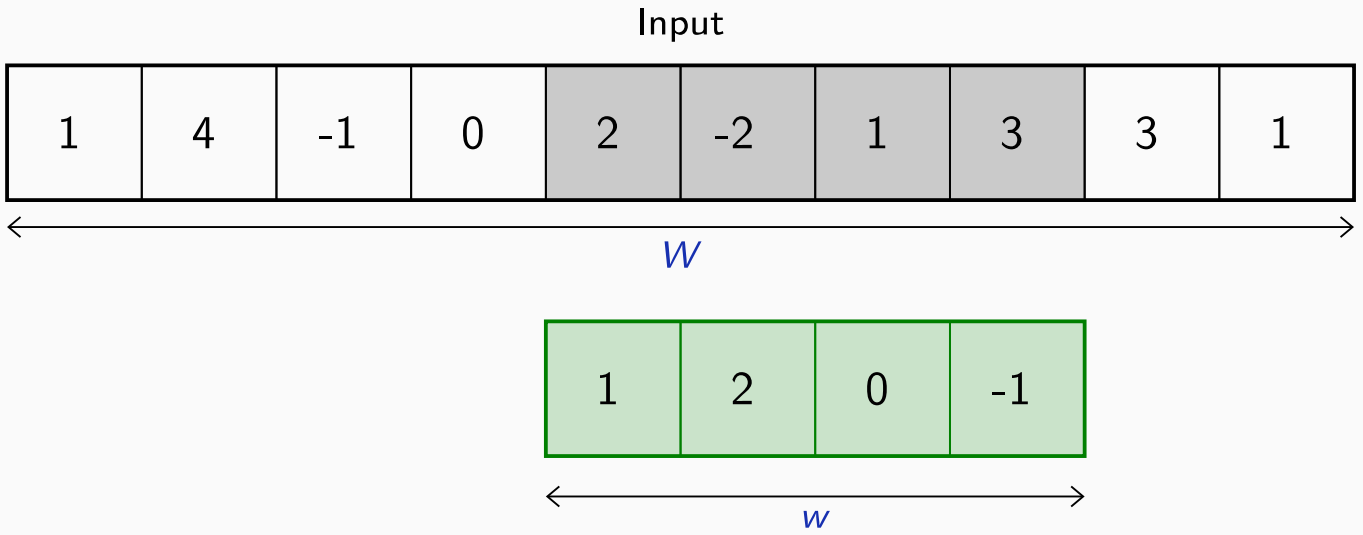
Convolution



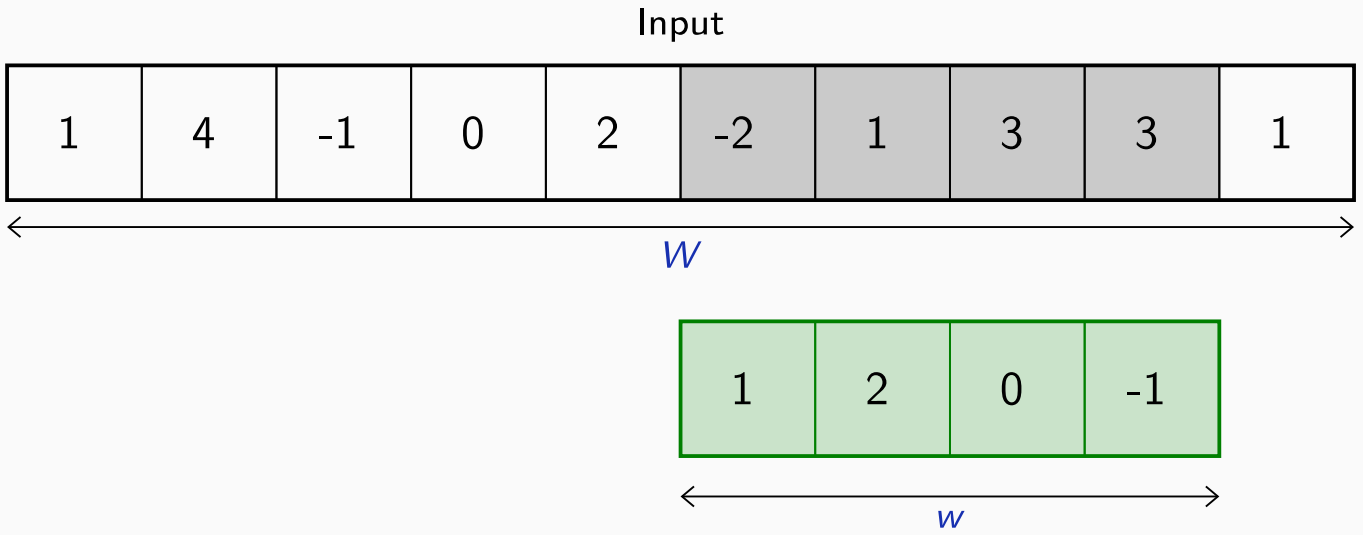
Convolution



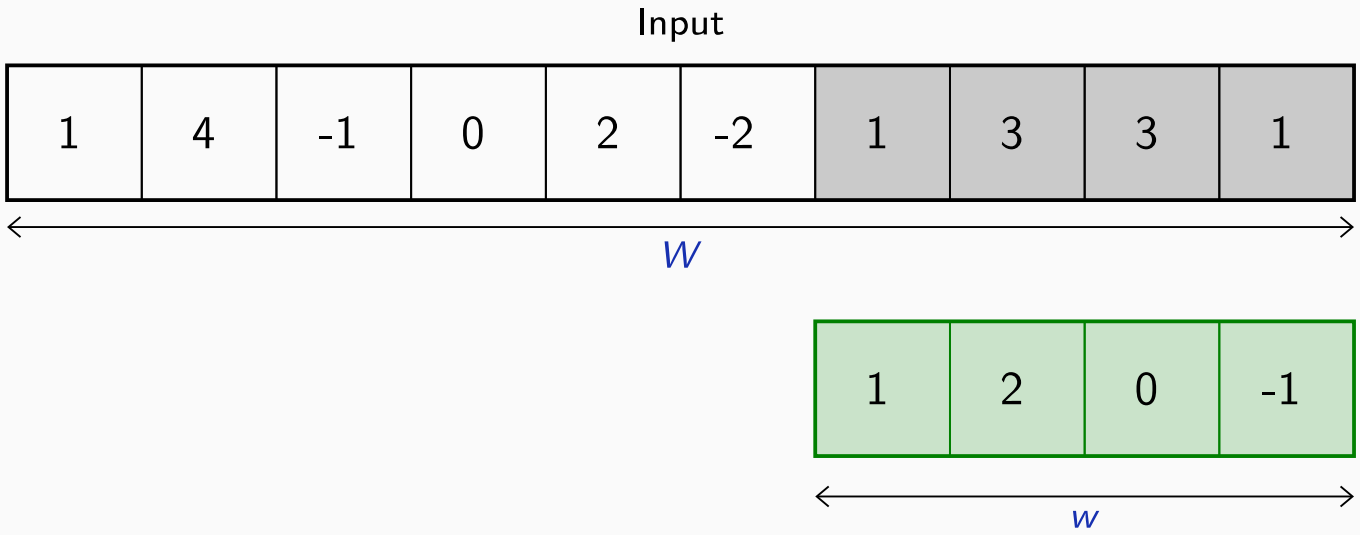
Convolution

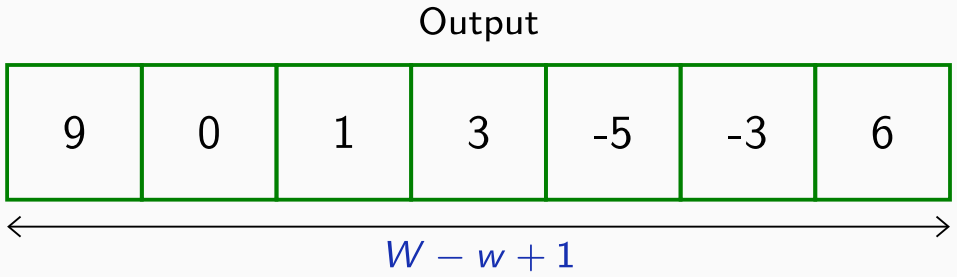


Convolution

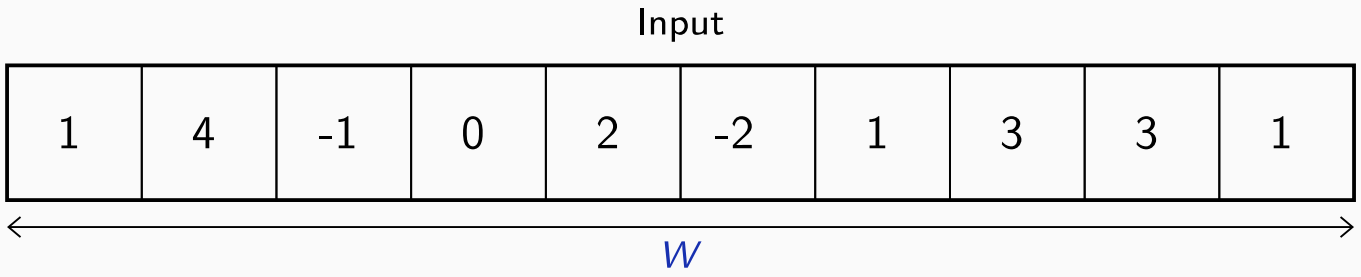


Convolution

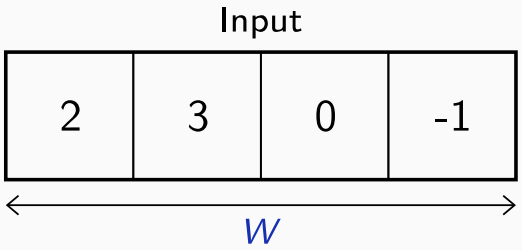


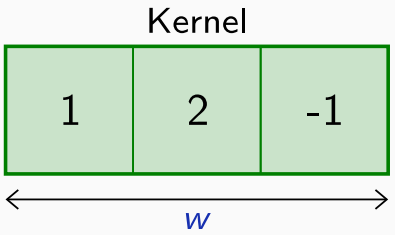


Convolution

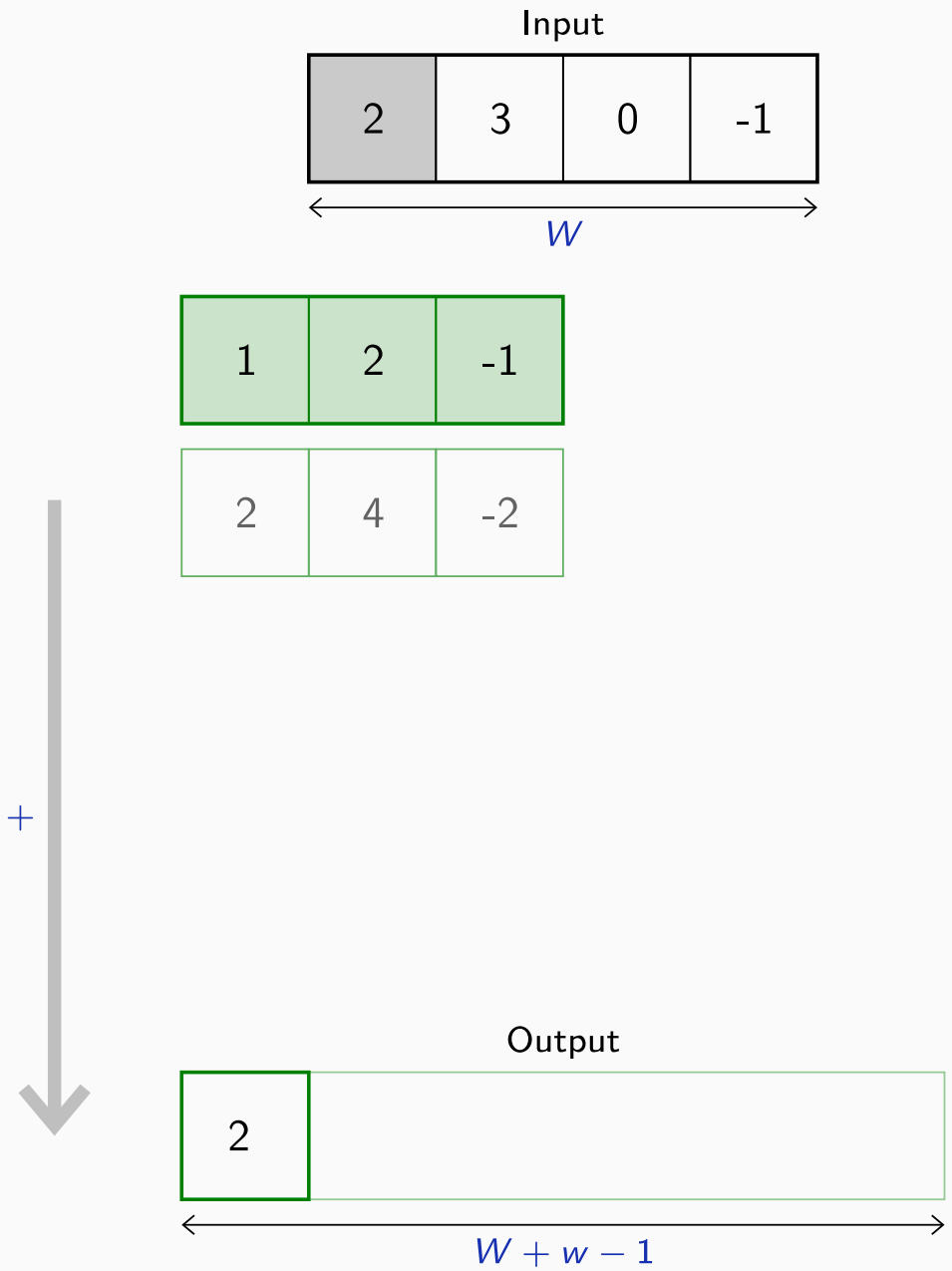


Transposed Convolution

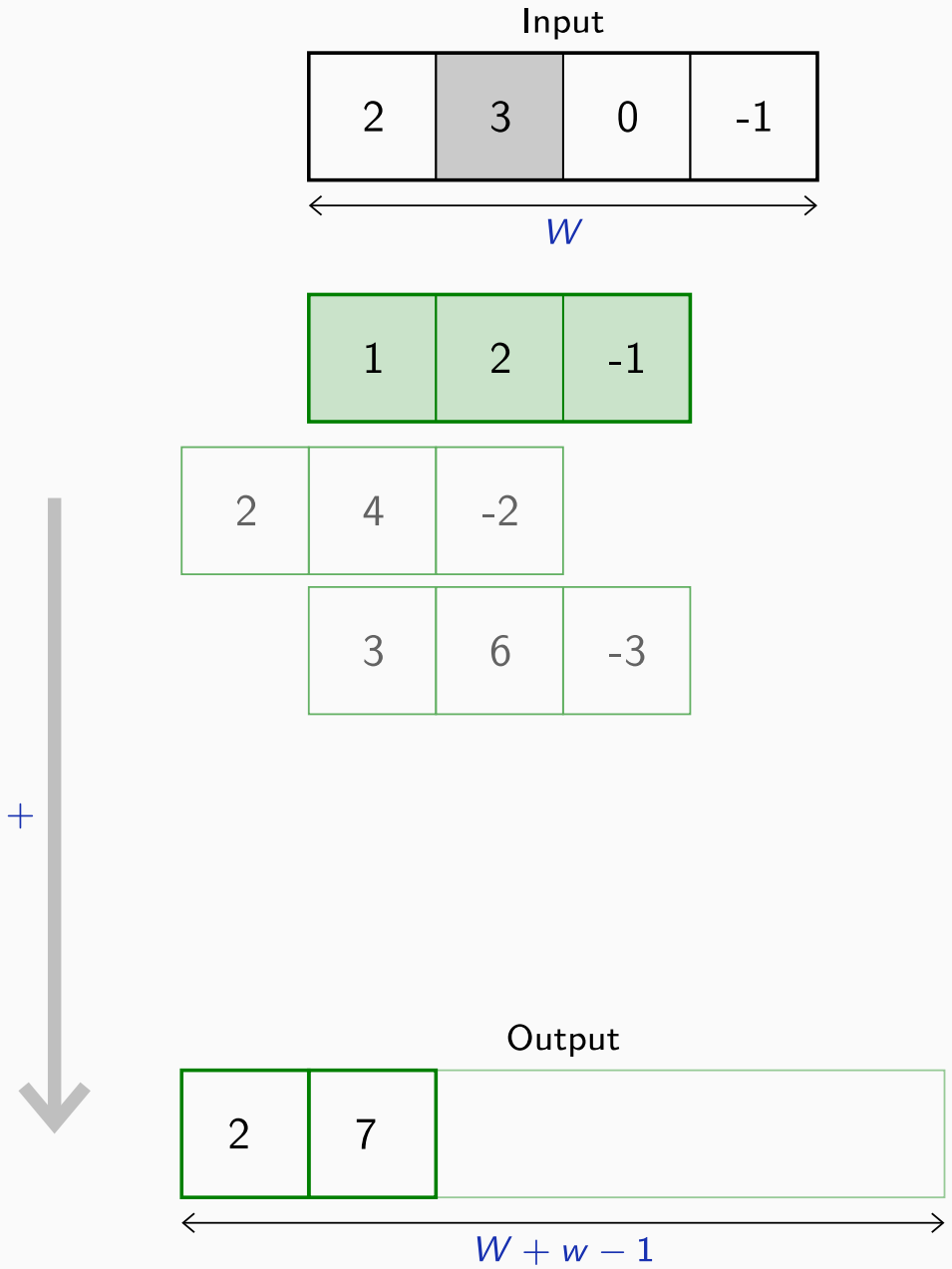




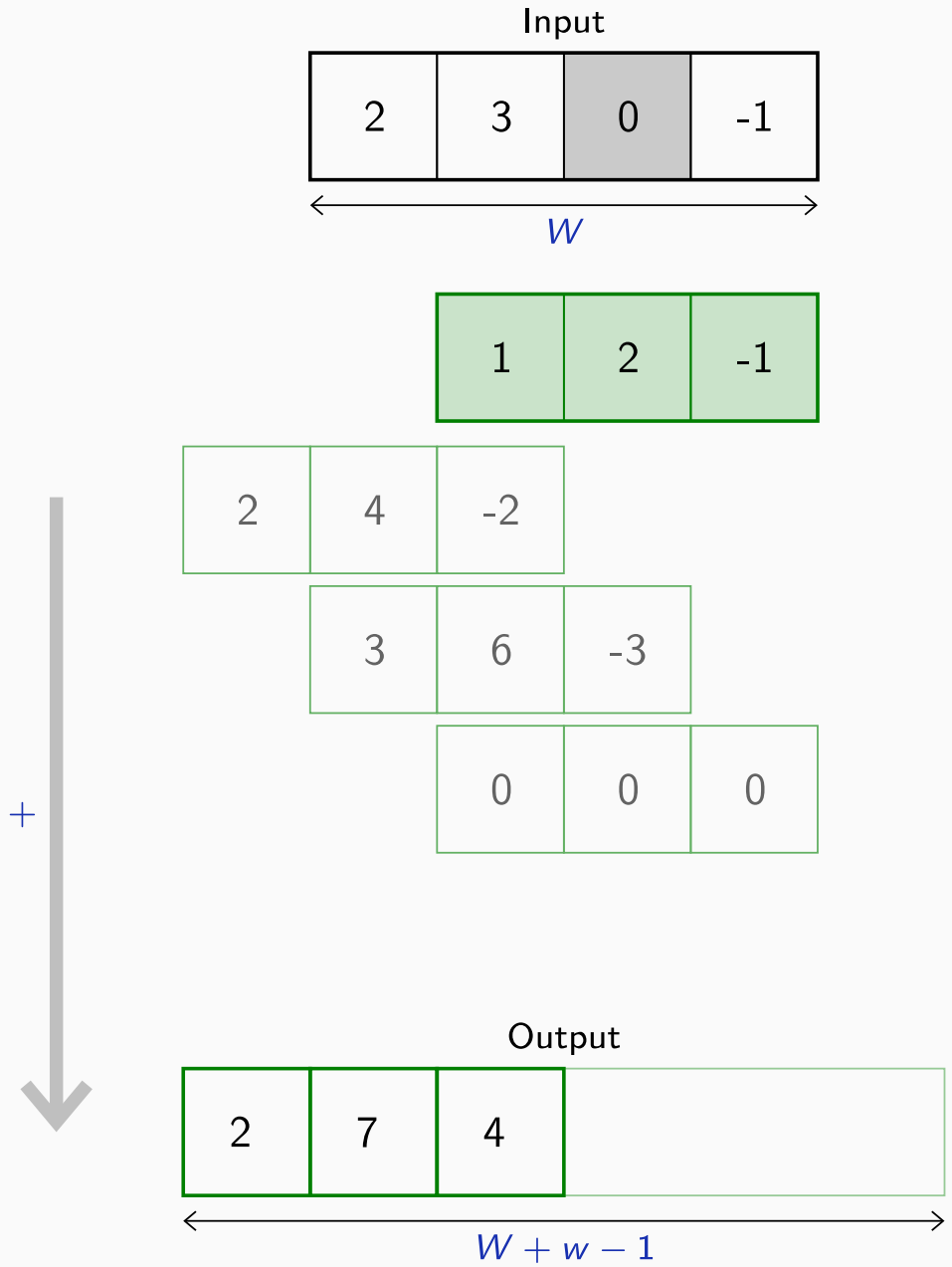
Transposed Convolution



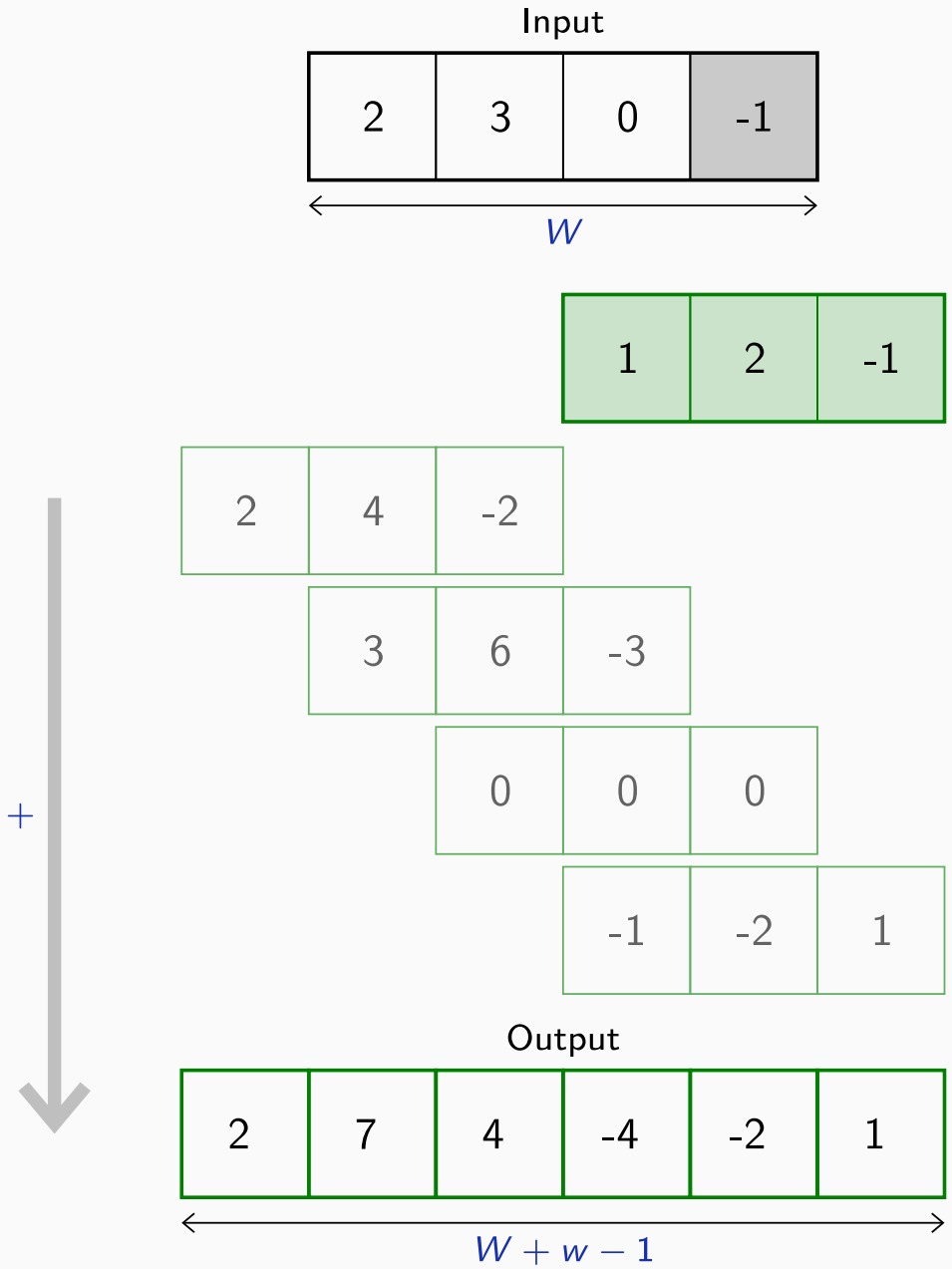
Transposed Convolution



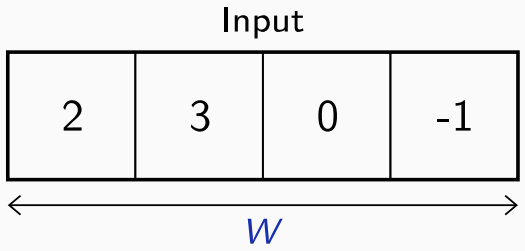
Transposed Convolution

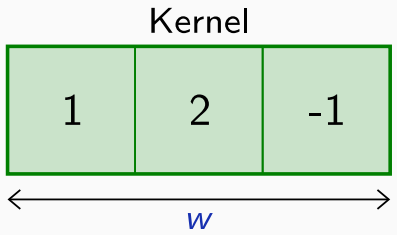


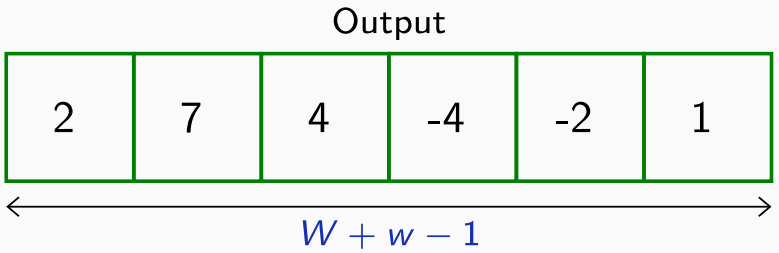
Transposed Convolution



Transposed Convolution





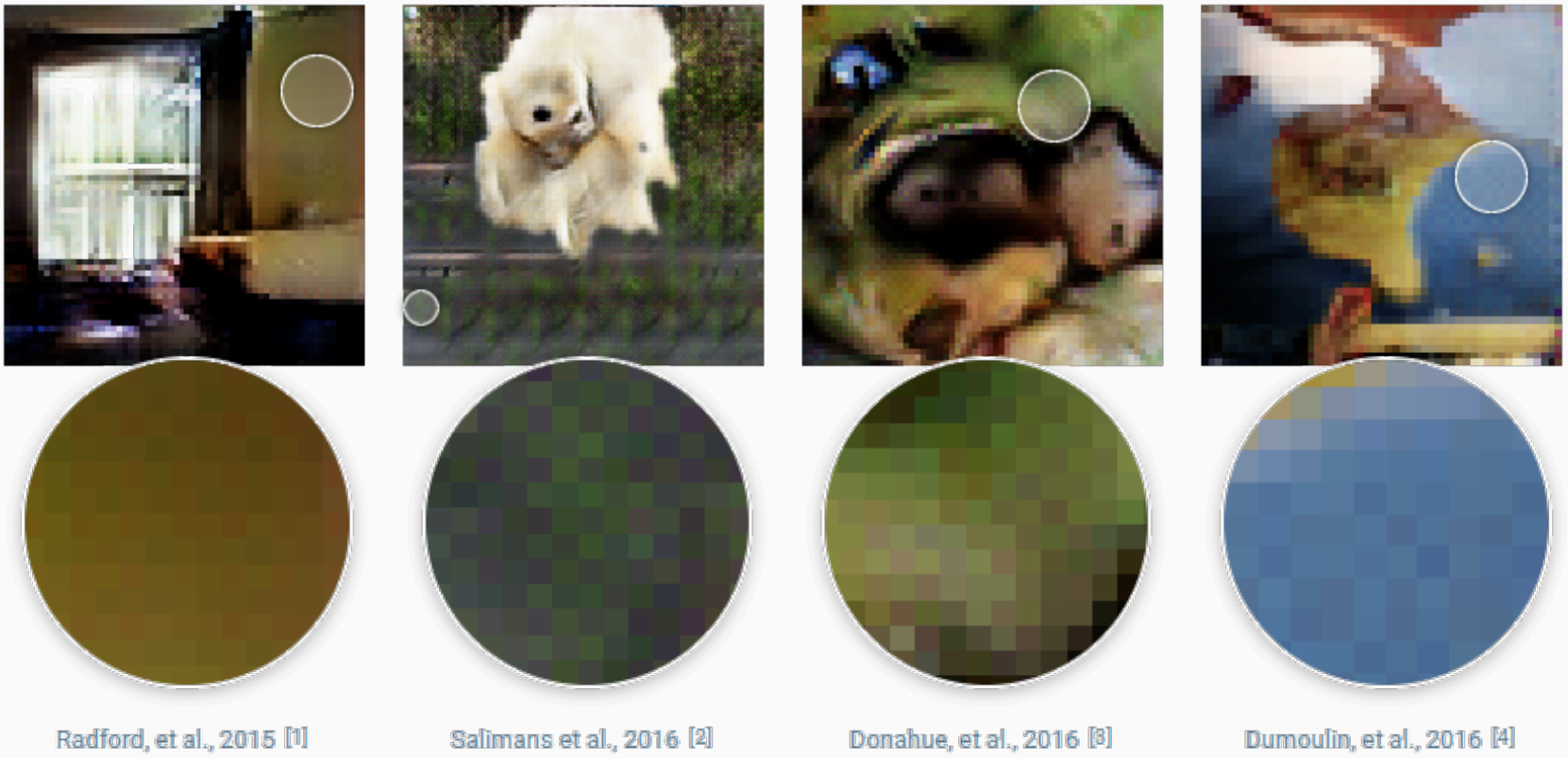


Because they are defined with convolutions in mind, transposed convolutions also have dilation, stride, and padding.

- dilation: spacing between kernel elements
- stride: spacing between consecutive steps (!)
- padding: removes a ring around the output (!)

A convolution followed by a transposed convolution, where both have the same settings (kernel size, dilation, stride, and padding), will preserve the input shape in the output, up to rounding.

Caveat: transposed convolutions can introduce artifacts.



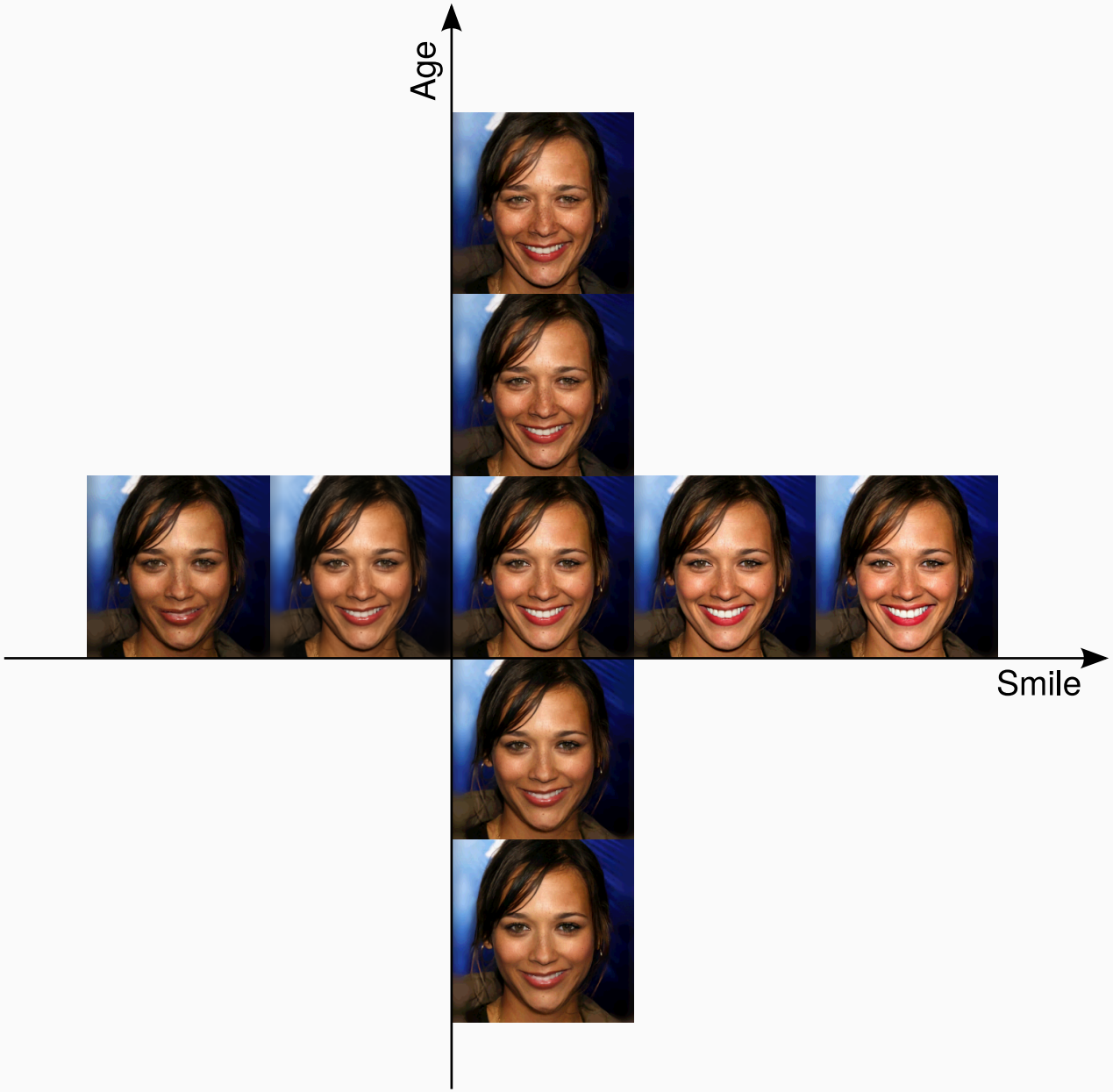
Alternative: nearest neighbor upsampling followed by regular convolutions.

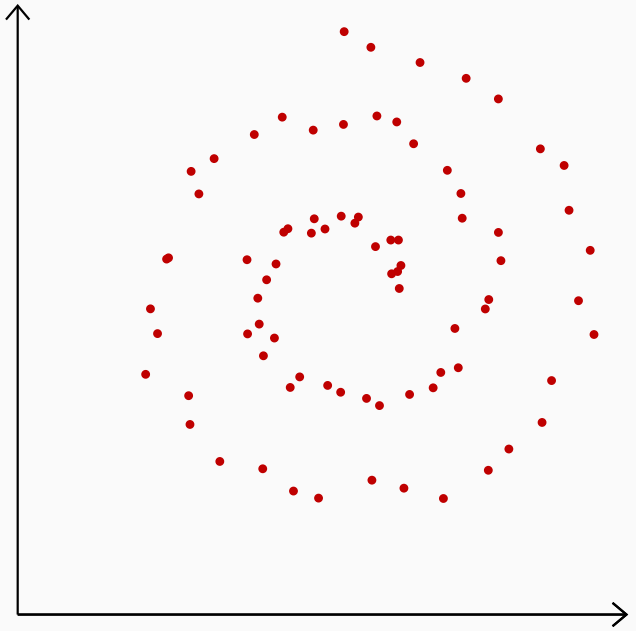
Autoencoders

Applications such as image or speech synthesis, denoising, super-resolution, or compression requires us to go beyond classification and regression.

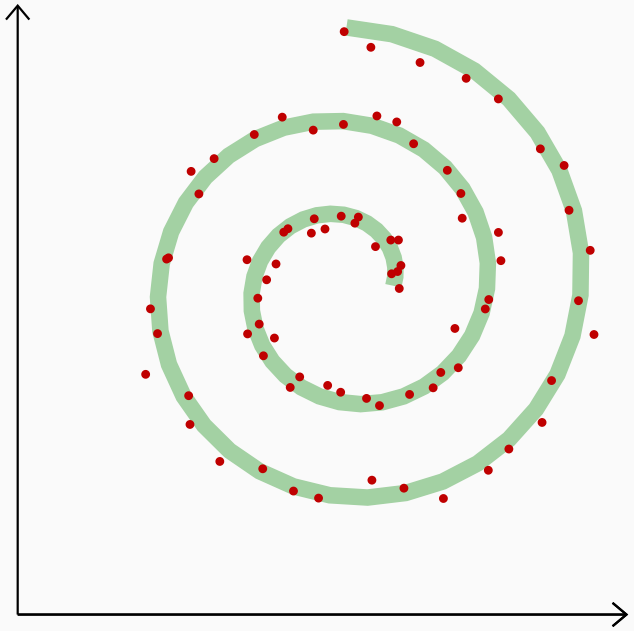
In those, modeling the high-dimensional input signal is crucial to modeling the processing generating it.

We can see it as finding the *meaningful degrees of freedom* that describe the signal, which are of lower dimension.

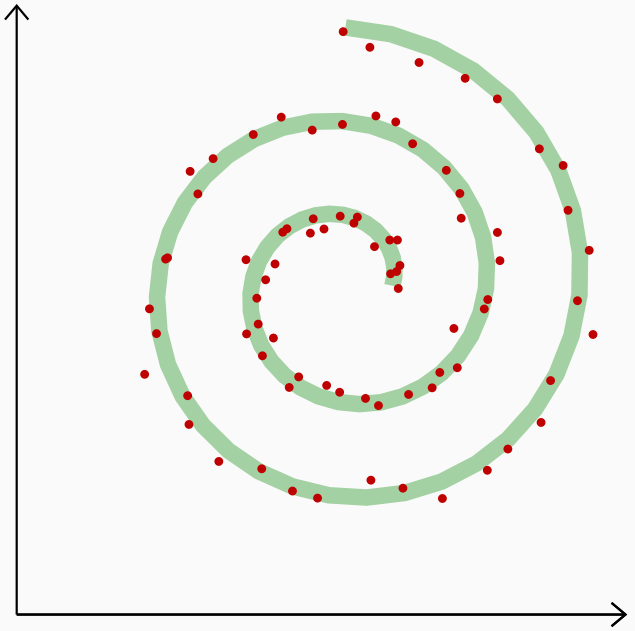

Age
Smile



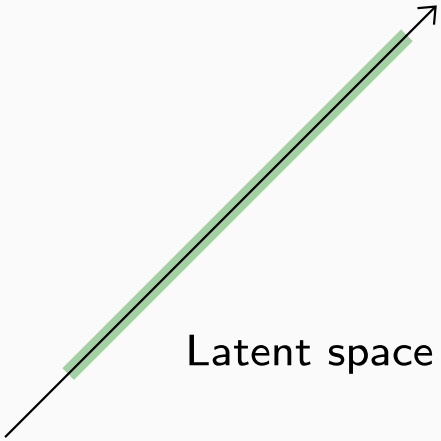
Original space



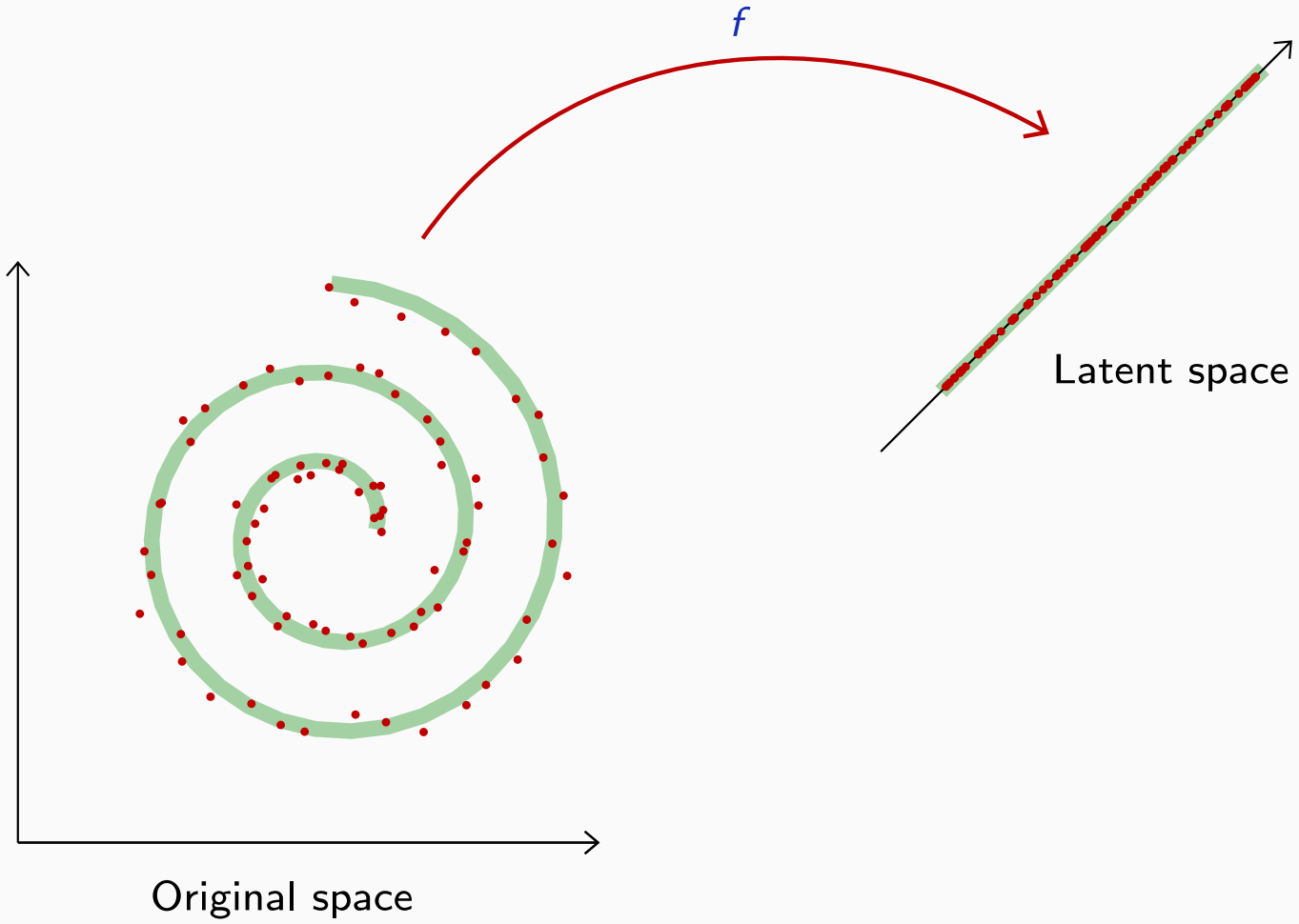
Original space

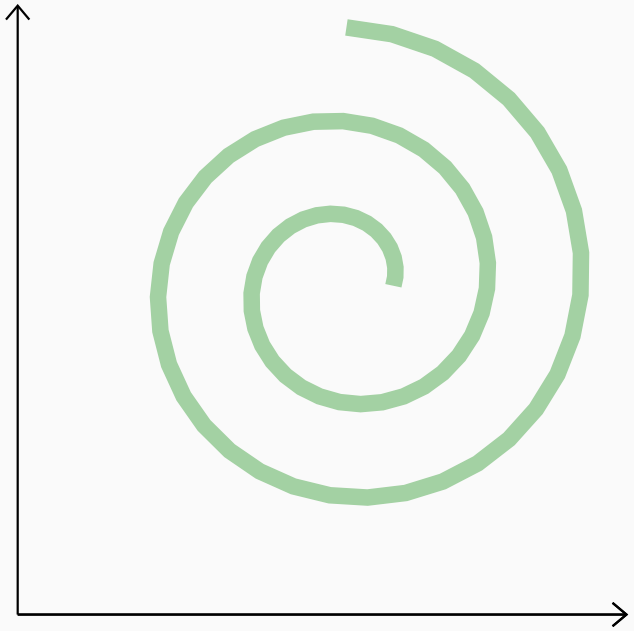


Original space

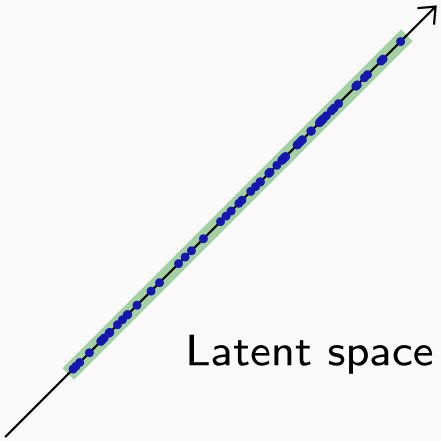


Latent space

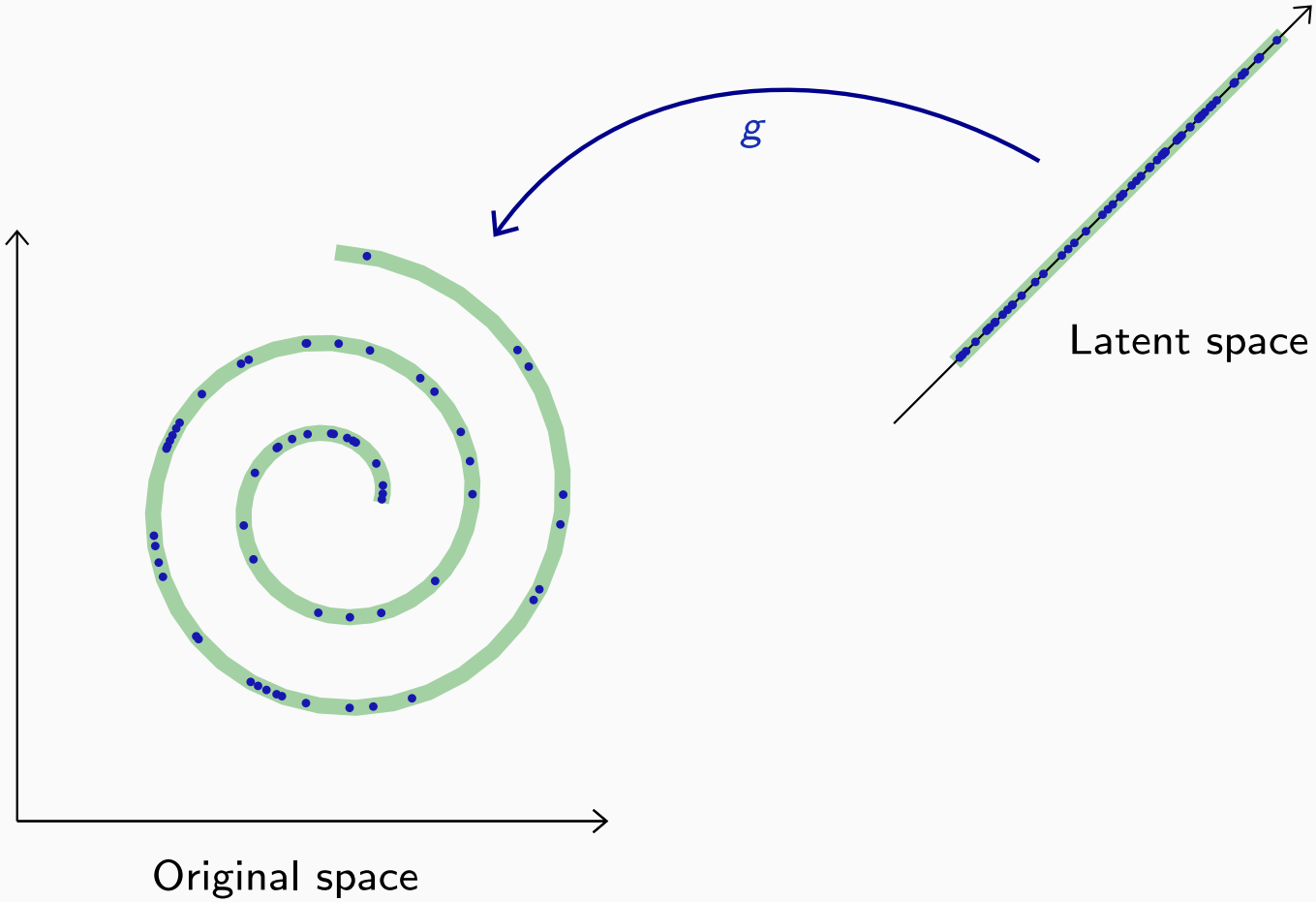

f
Latent space
Original space

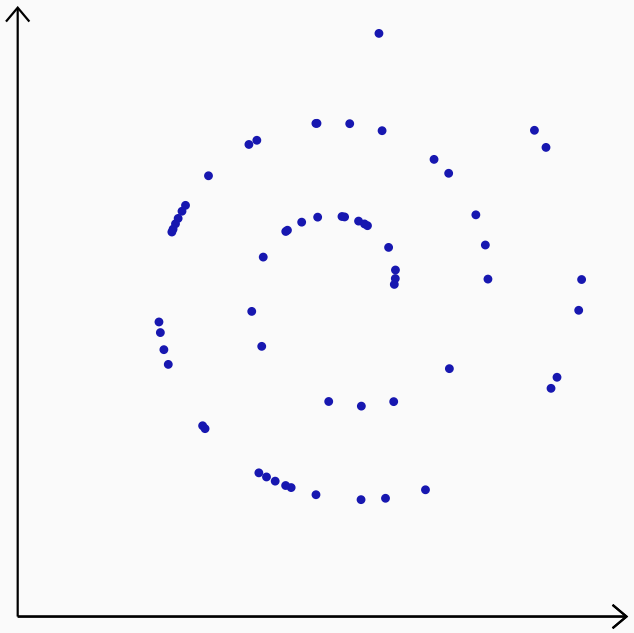


Original space



Latent space


g
Latent space
Original space

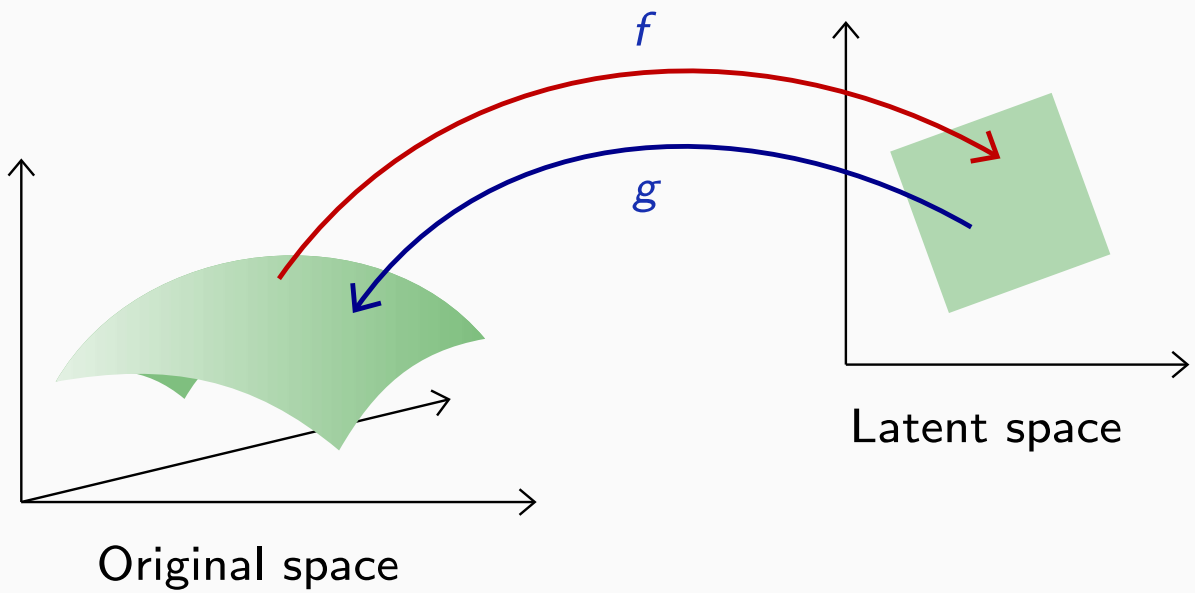


Original space

When dealing with high-dimensional inputs, the challenges in this task are similar to classification or regression: defining the right class of models (architectures) and optimizing them.

An *autoencoder* deals with this task by mapping a space onto itself, attemping to make the whole transformation as close as possible to identity.

The autoencoder decomposes this transformation into two parts. One **encoder** f maps from the original space into some latent space, while a **decoder** g maps back to the original space.



If the latent space is of lower dimension, the encoder must extract the information necessary for the decoder to reconstruct the original observation from it.

Formally, this is expressed as the following objective. Given two parametrized mappings $f(\cdot; w_{\text{enc}})$ and $g(\cdot; w_{\text{dec}})$ we want

$$\hat{w}_{\text{enc}}, \hat{w}_{\text{dec}} = \underset{w_{\text{enc}}, w_{\text{dec}}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \| x_i - g(f(x_i; w_{\text{enc}}); w_{\text{dec}}) \|^2$$

Importantly, if we restrict f and g to be linear, the optimal solution is given by PCA.

Naturally, using neural networks we can model more complex mappings.

For instance, a deep autoencoder for image data is composed of convolutional layers, with a decoder composed of transposed convolutions or other interpolating transformations.

```
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)`

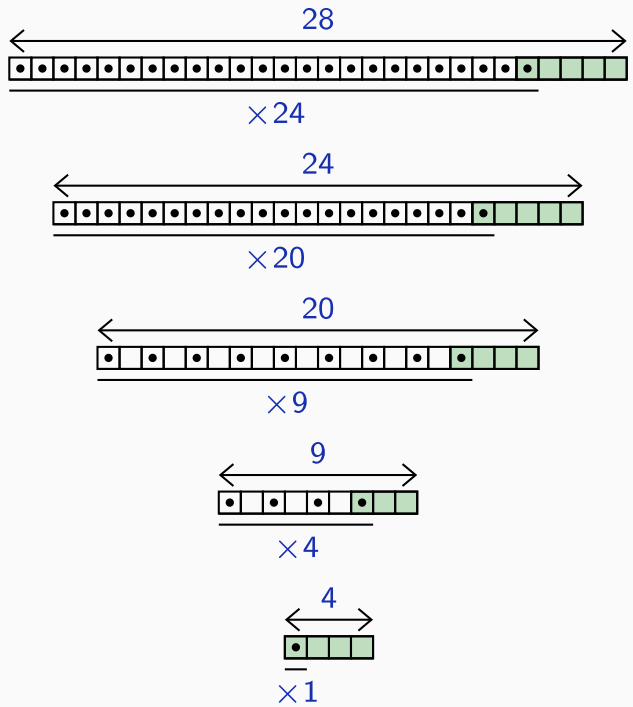
$32 \times 9 \times 9$

`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)`

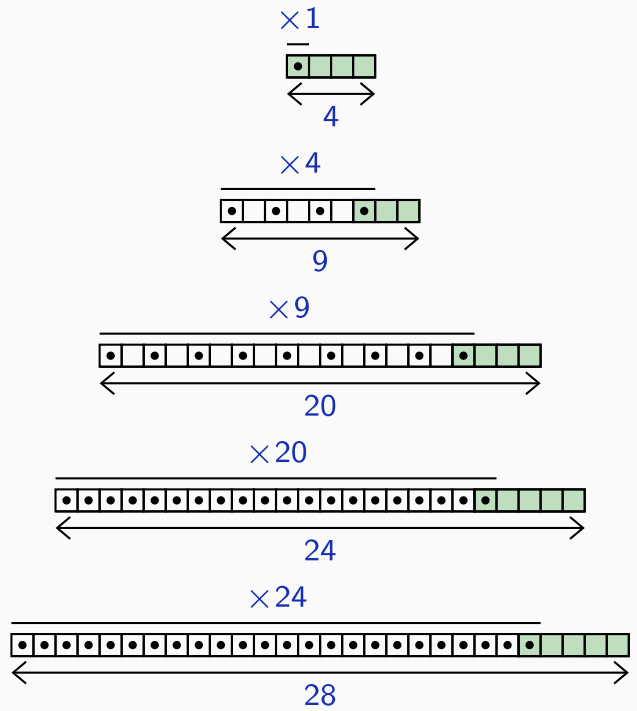
$32 \times 20 \times 20$

`nn.ConvTranspose2d(32, 32, kernel_size=5, stride=1)`

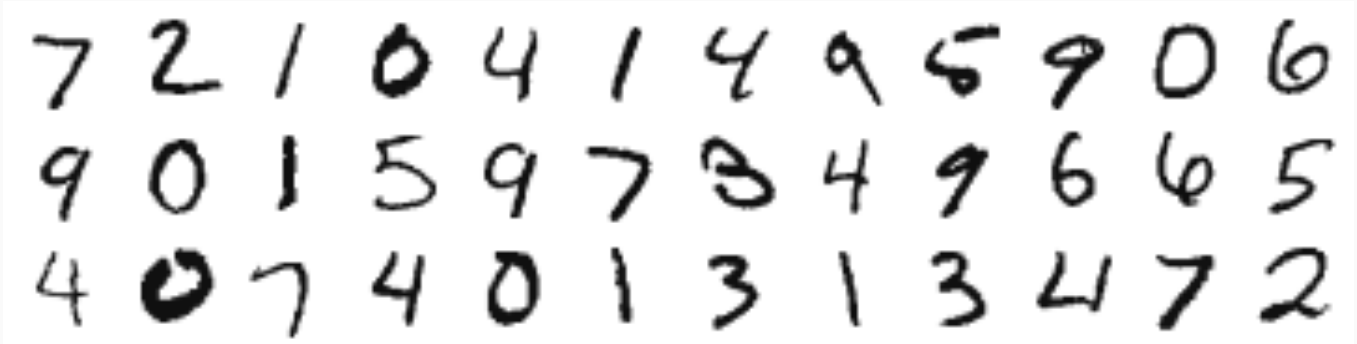
$32 \times 24 \times 24$

`nn.ConvTranspose2d(32, 1, kernel_size=5, stride=1)`

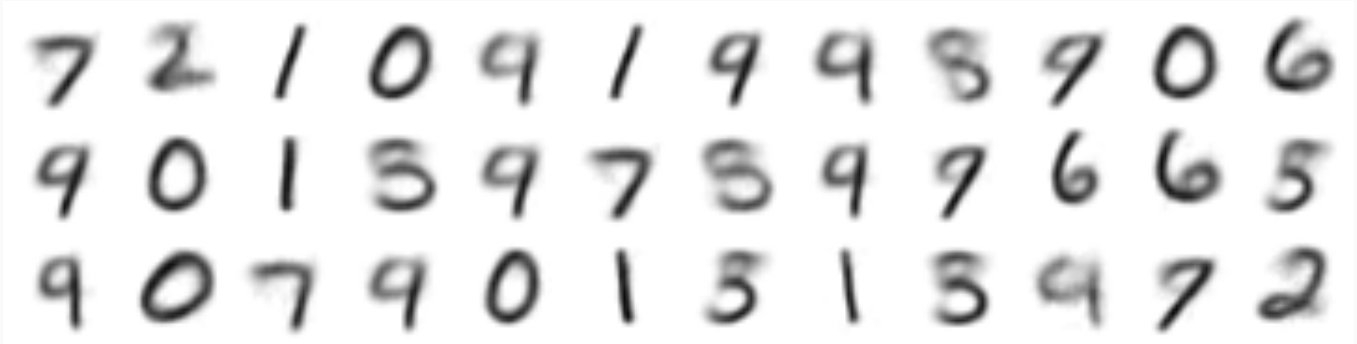
$1 \times 28 \times 28$



X (original samples)

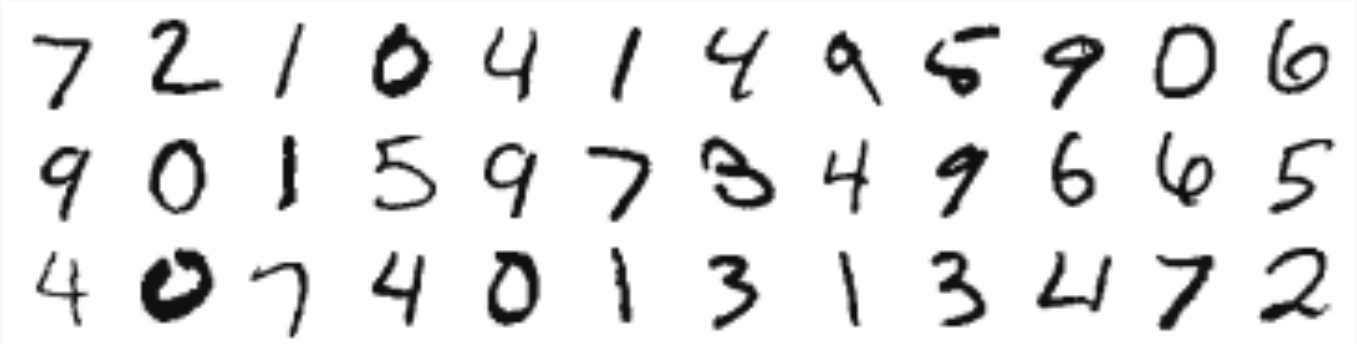


$g \circ f(X)$ (CNN, $d = 2$)

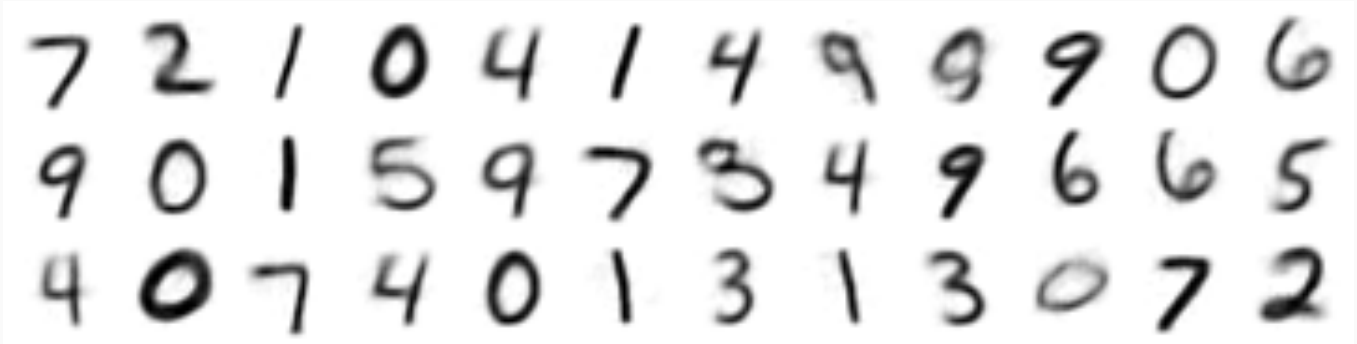


$g \circ f(X)$ (PCA, $d = 2$)

X (original samples)

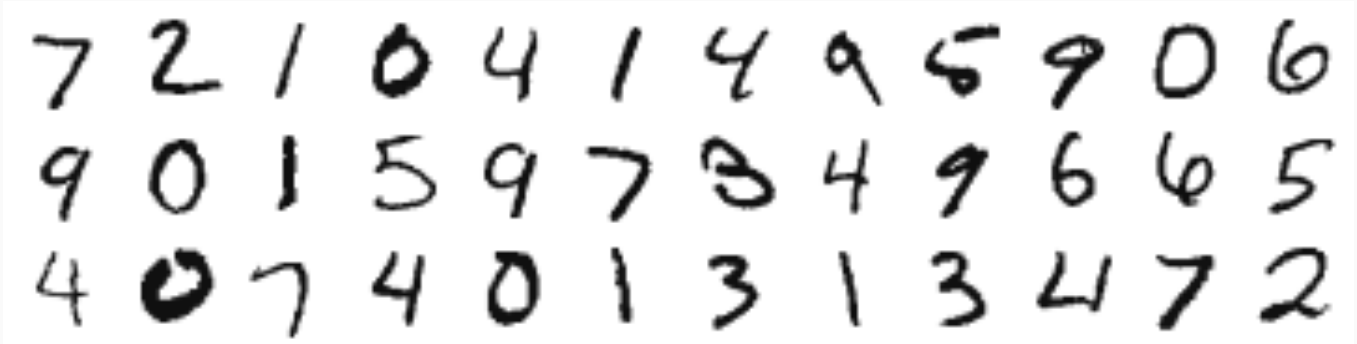


$g \circ f(X)$ (CNN, $d = 4$)

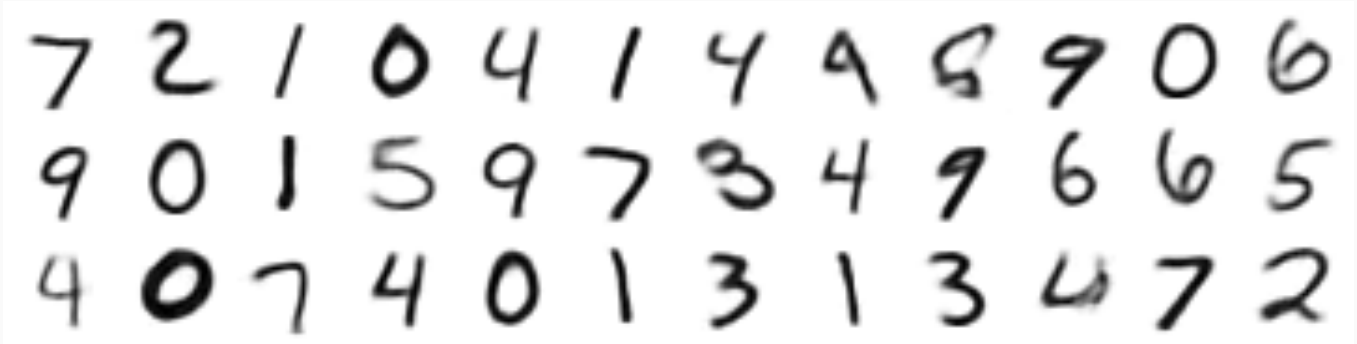


$g \circ f(X)$ (PCA, $d = 4$)

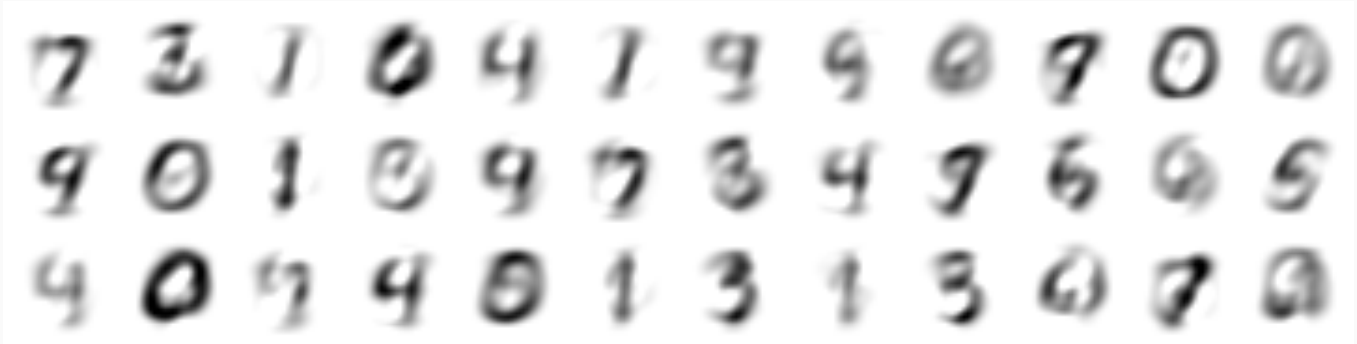
X (original samples)



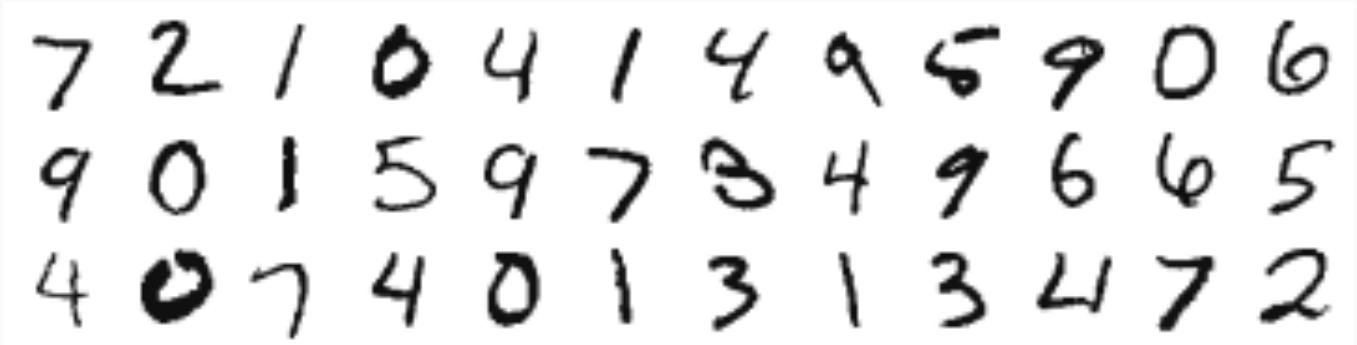
$g \circ f(X)$ (CNN, $d = 8$)



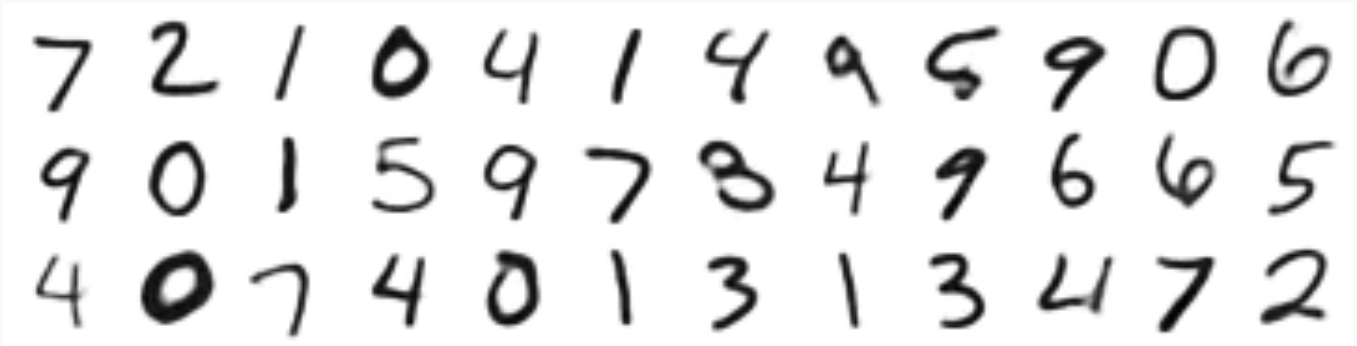
$g \circ f(X)$ (PCA, $d = 8$)



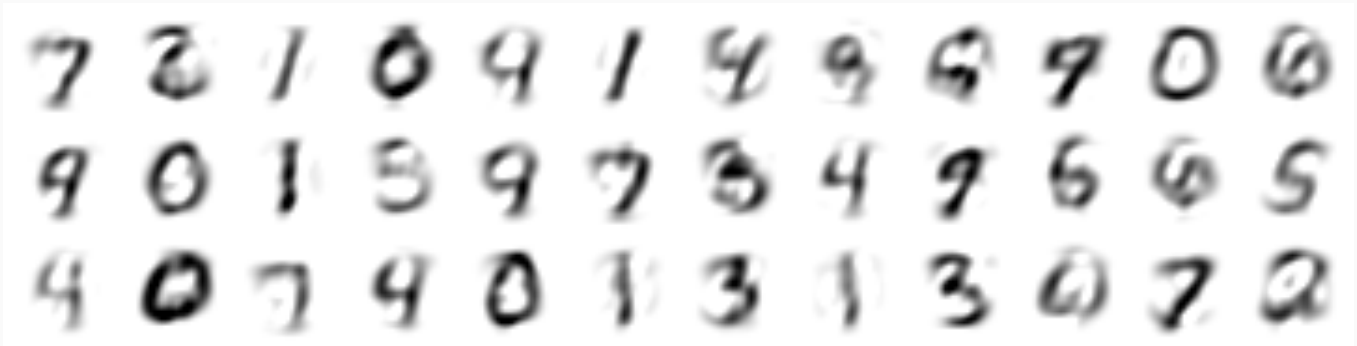
X (original samples)



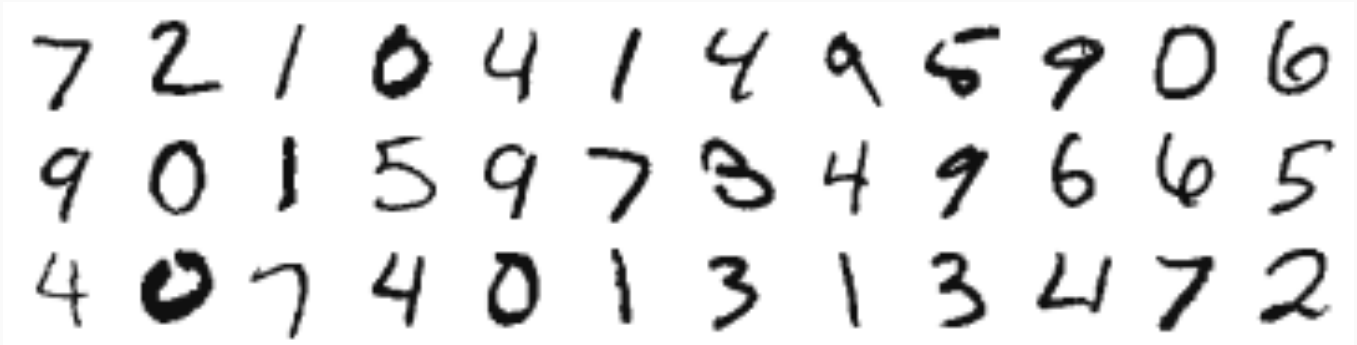
$g \circ f(X)$ (CNN, $d = 16$)



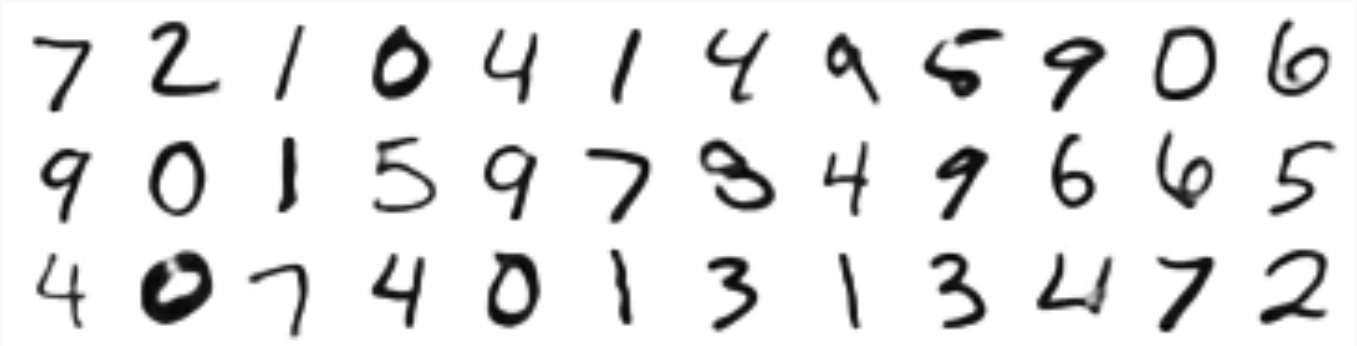
$g \circ f(X)$ (PCA, $d = 16$)



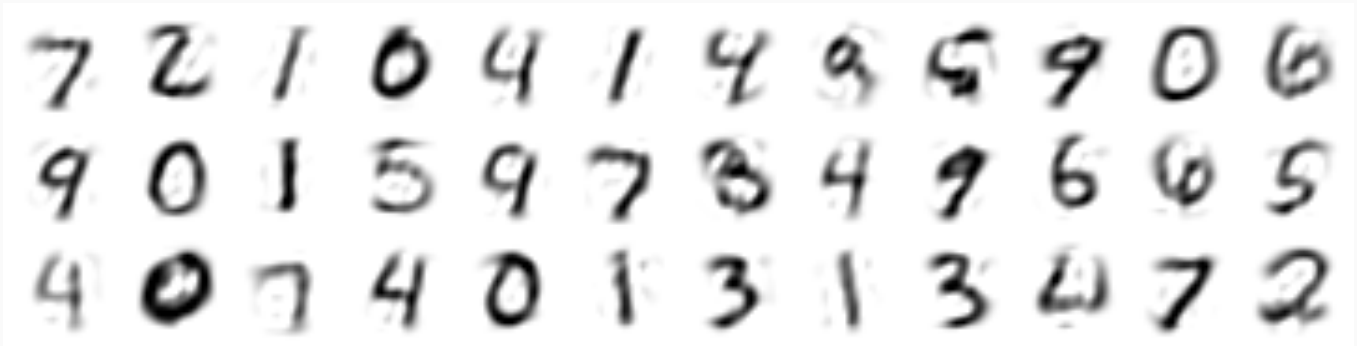
X (original samples)



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



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

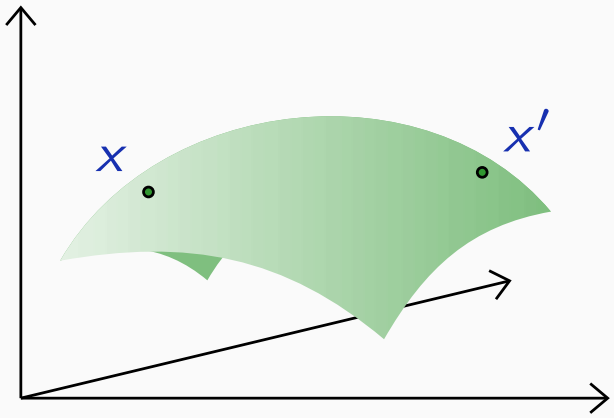


One common operation that yields us some intuition about the latent space of autoencoders (and other generative models) works as follows.

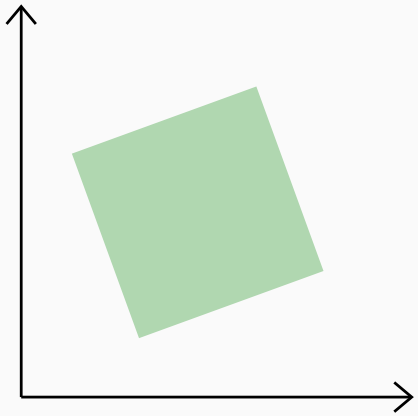
We pick two samples x and x' at random and then interpolate them, in the latent space, along a line:

$$x_{\text{lerp}} = g((1-\alpha)f(x) + \alpha f(x')).$$

$$x_{\text{lerp}} = g((1-\alpha)f(x) + \alpha f(x'))$$

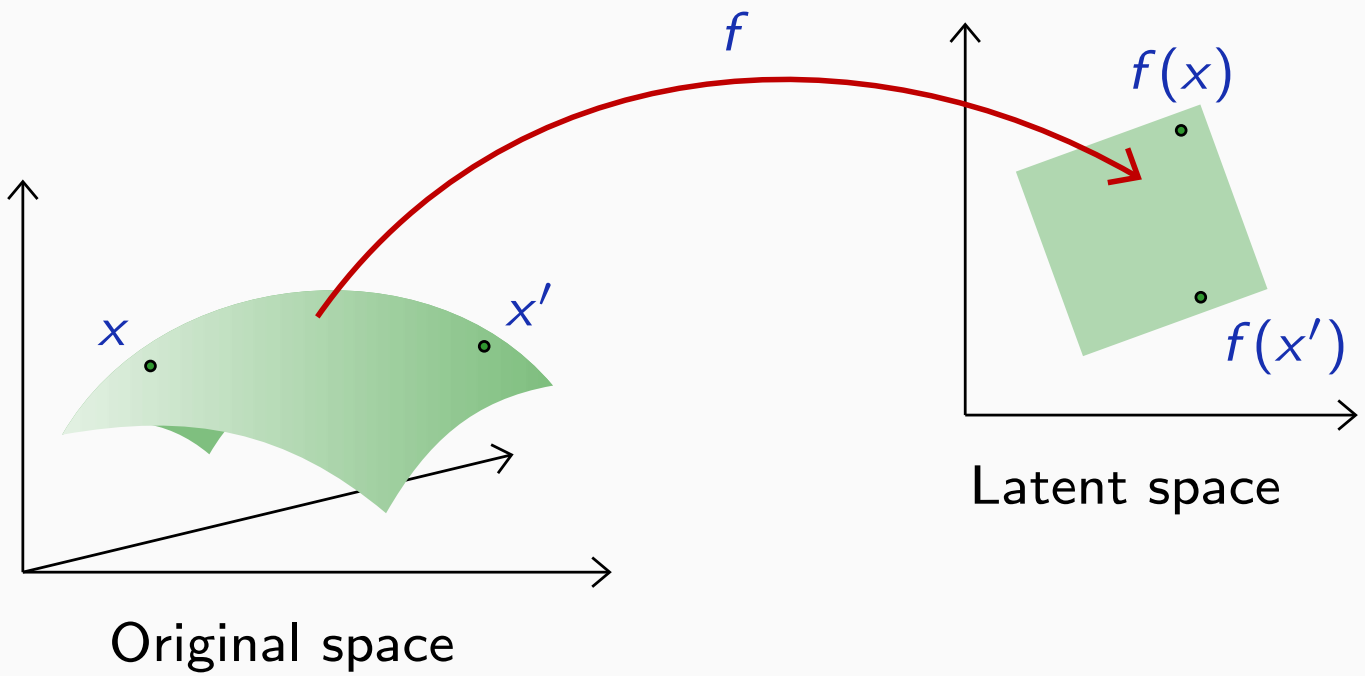


Original space

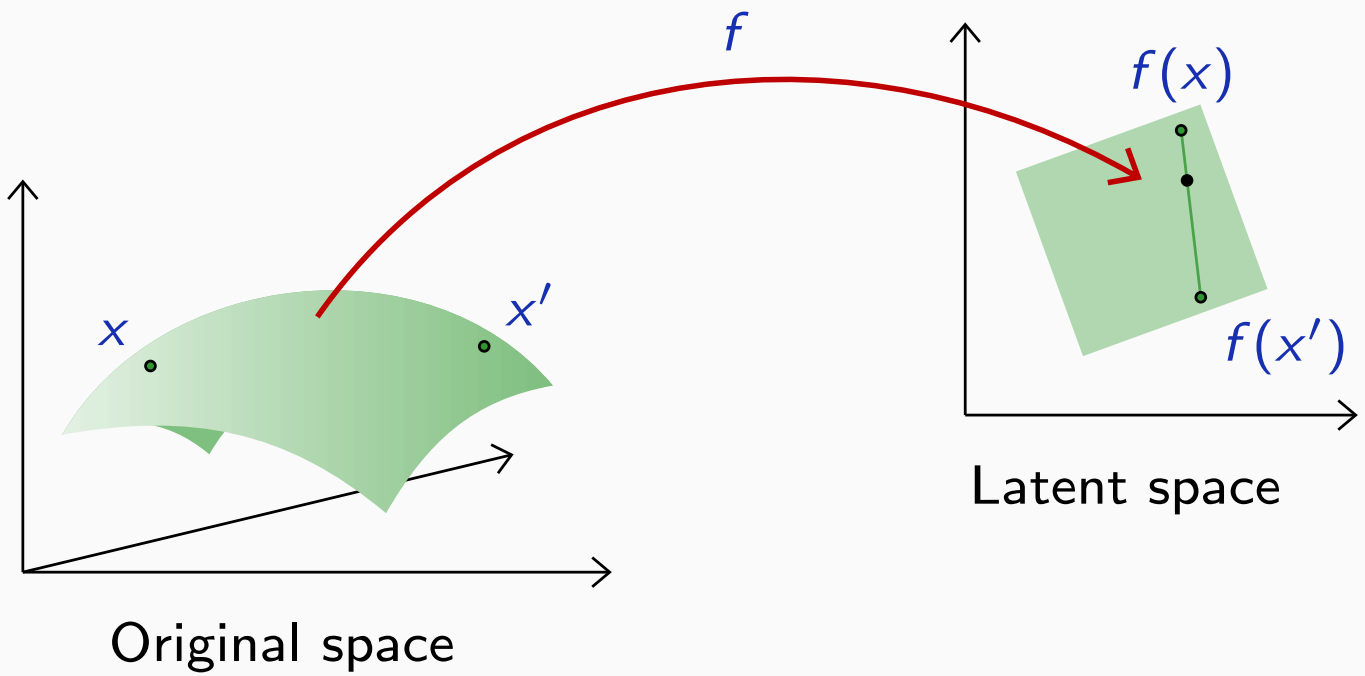


Latent space

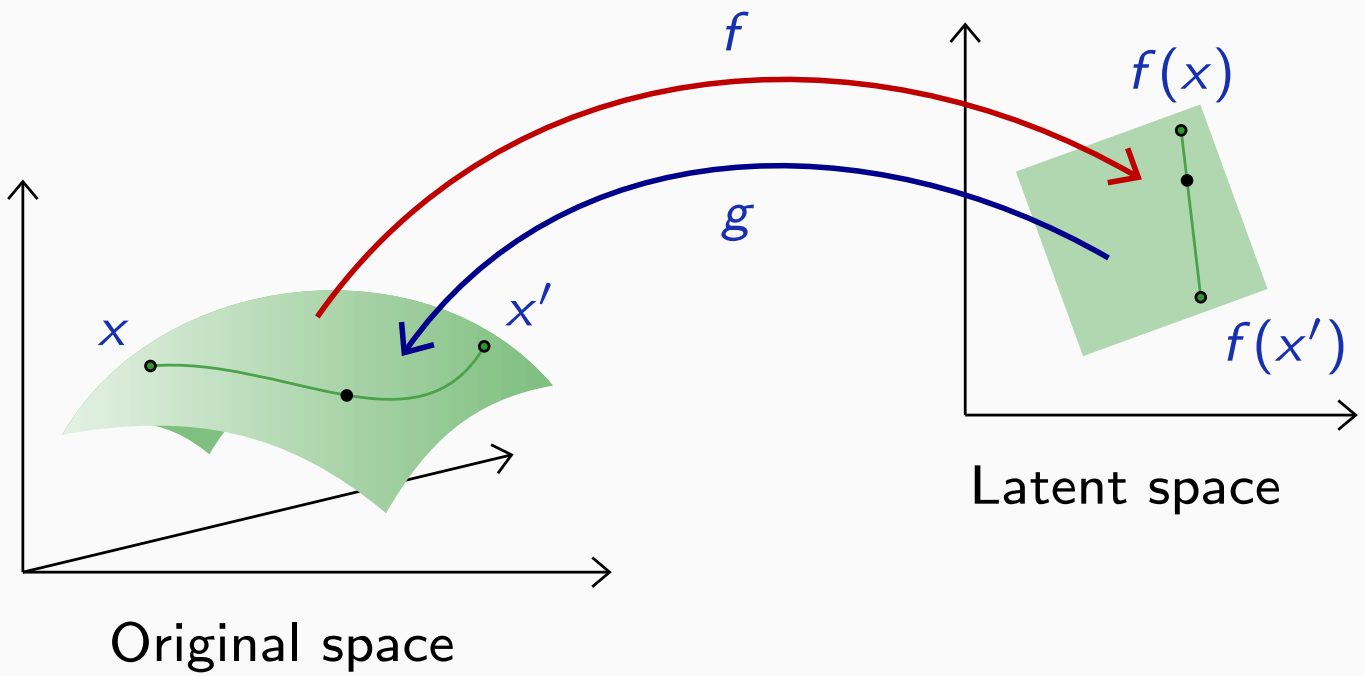
$$x_{\text{lerp}} = g((1-\alpha)f(x) + \alpha f(x'))$$



$$x_{\text{lerp}} = g((1-\alpha)f(x) + \alpha f(x'))$$

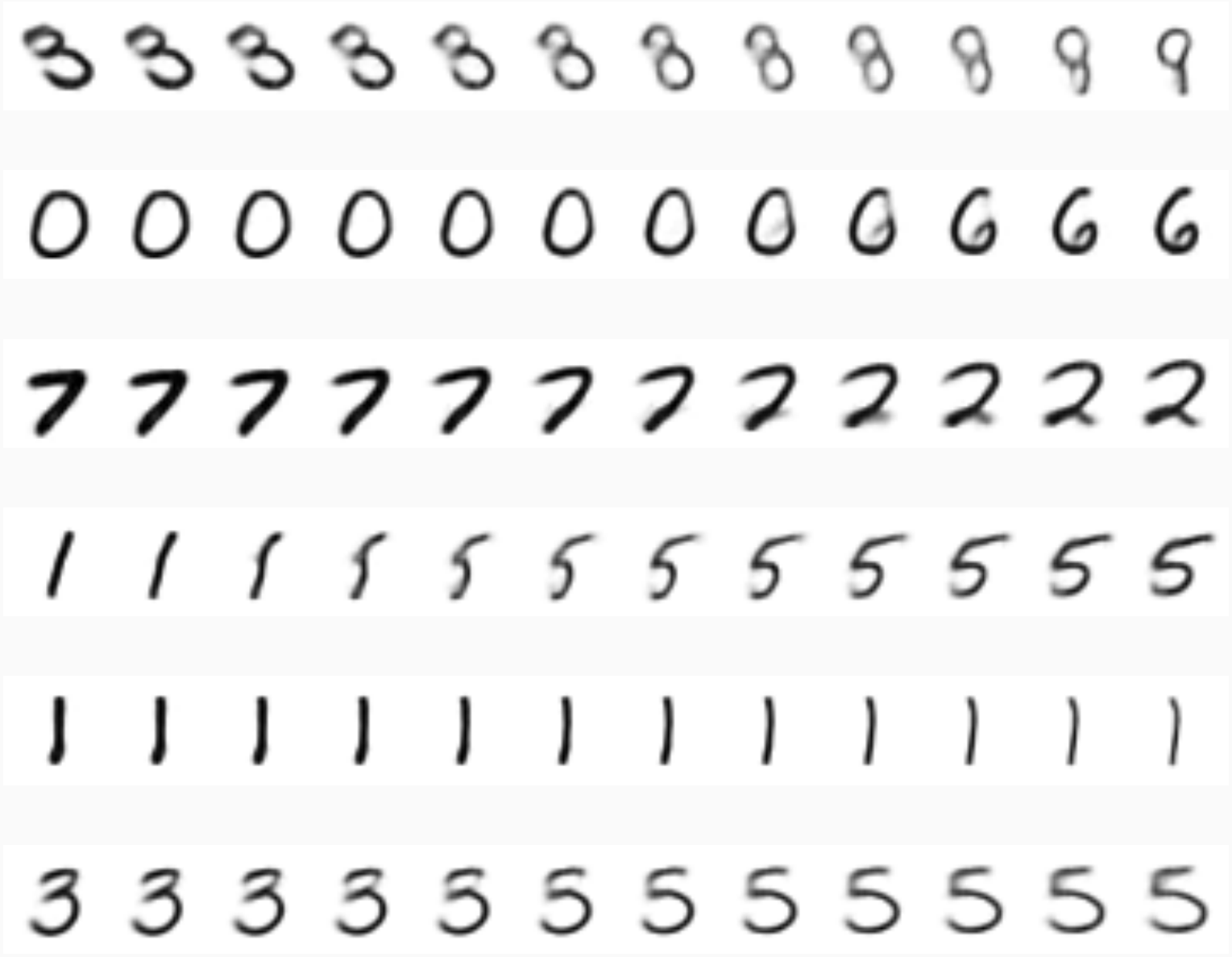


$$x_{\text{lerp}} = g((1-\alpha)f(x) + \alpha f(x'))$$

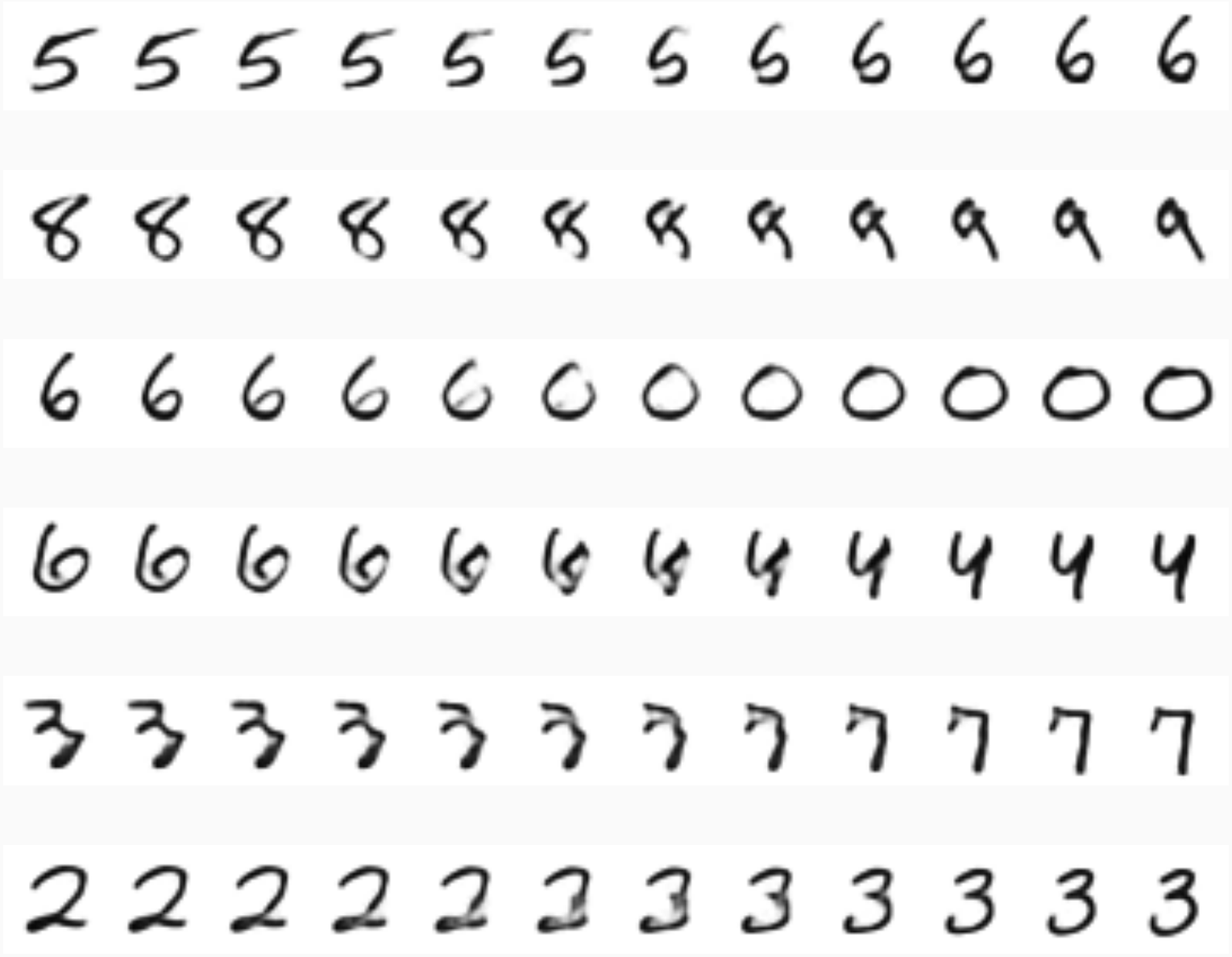


PCA interpolation ($d = 32$)

Autoencoder interpolation ($d = 8$)

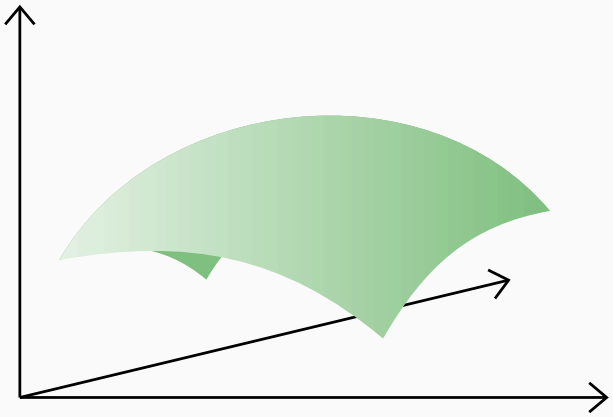


Autoencoder interpolation ($d = 32$)

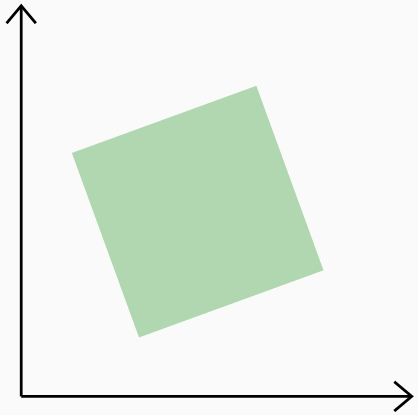


Another way of inspecting the latent space of an autoencoder, which also allows us to generate new data, is to introduce some density model over the latent space. Then, we could sample from this density and map them back to the original space to "see" them using g.

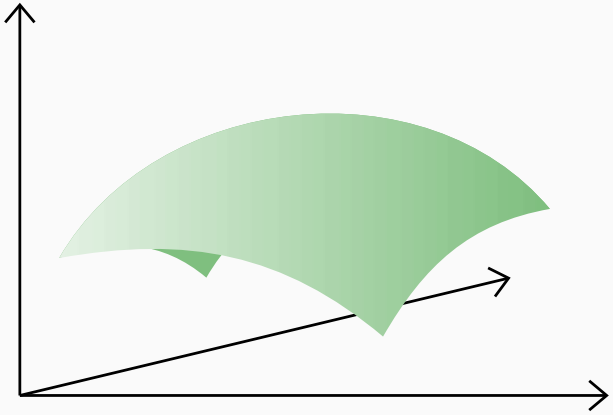
We can, for instance, use a simple Gaussian with a diagonal covariance matrix, and estimate both the covariance matrix and the mean from the training data.



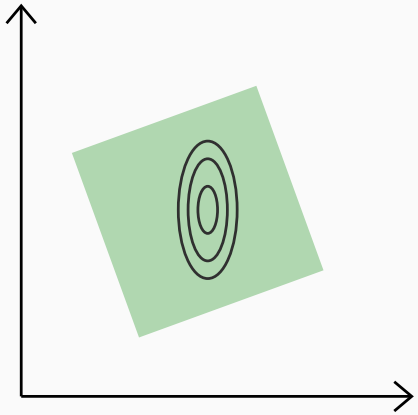
Original space



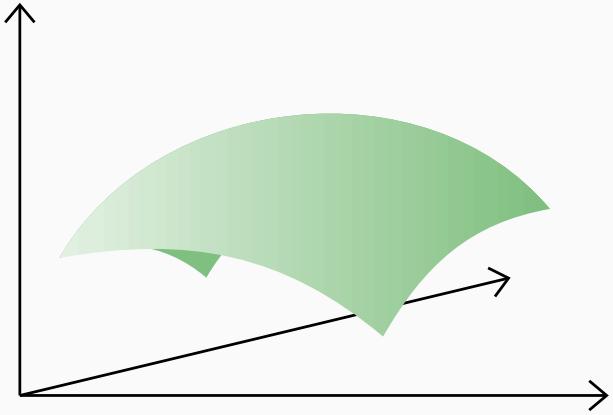
Latent space



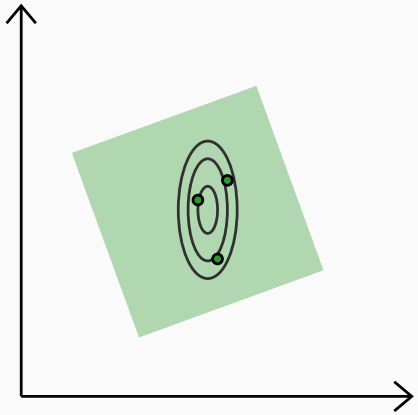
Original space



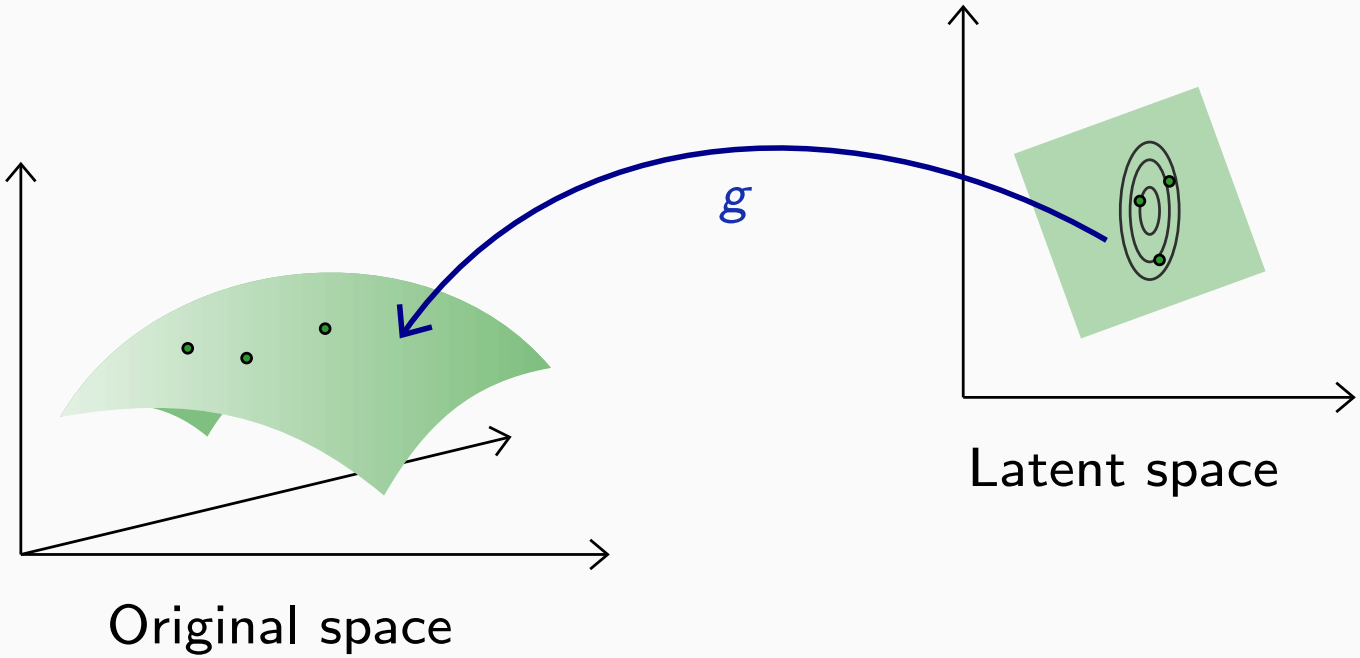
Latent space



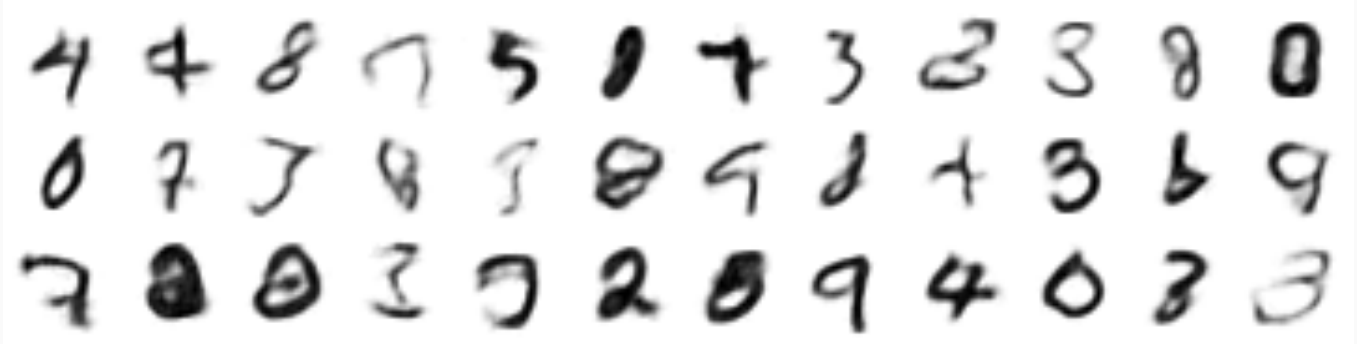
Original space



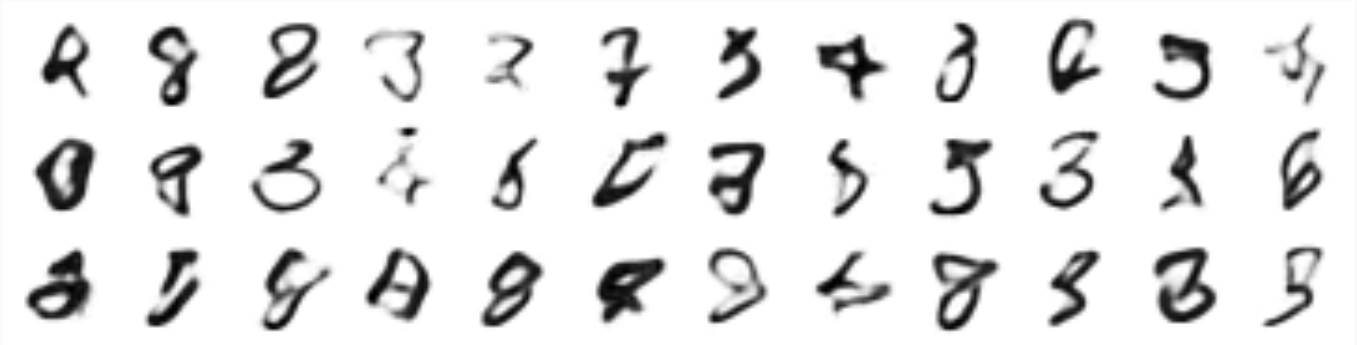
Latent space


g
Latent space
Original space

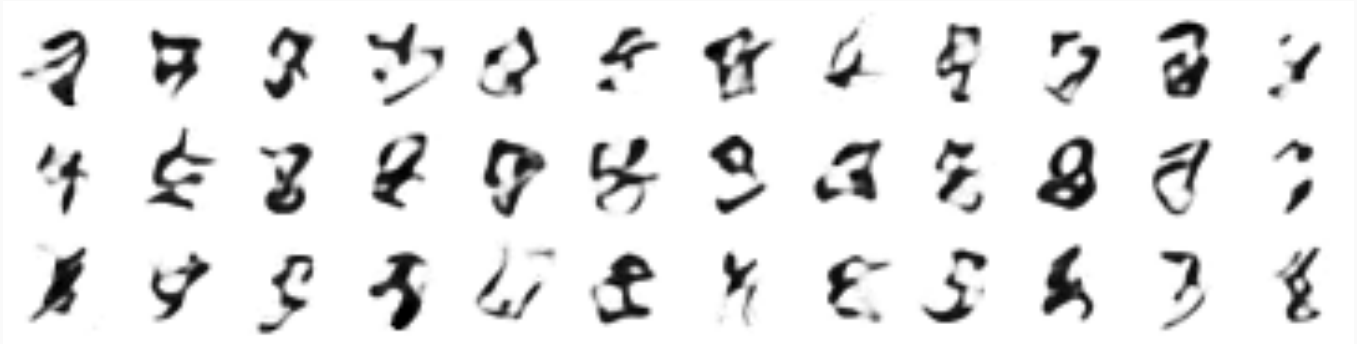
Autoencoder sampling ($d = 8$)



Autoencoder sampling ($d = 16$)



Autoencoder sampling ($d = 32$)



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.

Odena, A., Dumoulin, V., and Olah, C. Deconvolution and Checkerboard Artifacts, Distill, 2016. DOI: 10.23915/distill.00003.