Deep Learning

A journey from feature extraction and engineering to end-to-end pipelines

Part 5: Generative models. Self-supervised learning

Andrei Bursuc

With slides from A. Karpathy, F. Fleuret, J. Johnson, S. Yeung, G. Louppe, Y. Avrithis ...

Outline

1. Computer Vision
   - before things went deep: handcrafted features
2. Neural networks
   - empirical risk minimization, stochastic gradient descent, backpropagation
   - multilayer perceptrons
3. Going deeper
   - convolutional layers, deep regularization, deep architectures
4. Under the hood
   - understanding and visualizing CNNs, adversarial attacks
   - CPU vs GPU, using CNNs in practice
5. Unsupervised and self-supervised learning
   - generative models: autoencoders, variational autoencoders, generative adversarial networks
   - self-supervised learning

Outline

1. Computer Vision
   - before things went deep: handcrafted features
2. Neural networks
   - empirical risk minimization, stochastic gradient descent, backpropagation
   - multilayer perceptrons
3. Going deeper
   - convolutional layers, deep regularization, deep architectures
4. Under the hood
   - understanding and visualizing CNNs, adversarial attacks
   - CPU vs GPU, using CNNs in practice
5. Unsupervised and self-supervised learning
   - generative models: autoencoders, variational autoencoders, generative adversarial networks
   - self-supervised learning

Supervised vs Unsupervised Learning

Supervised Learning

- Data: (data=x,y=label)
- Goal: learn f(x)=y
- Examples: classification, regression, image captioning, etc.

Unsupervised Learning

- Data: (data=x,y=None)
- Goal: learn underlying hiddend structure of data
- Examples: clustering, dimensionality reduction, density estimation, etc.

Generative models

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 dimensional signal.

Generative models

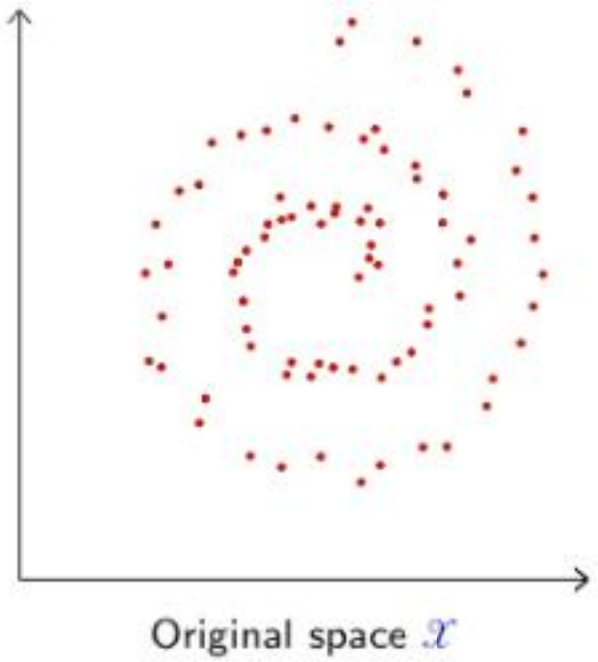
- Many applications such as image synthesis, denoising, super-resolution, speech synthesis, compression, etc. require to go beyond classification and regression, and model explicitly a high dimensional signal.
- This modeling consists of finding "meaningful degrees of freedom" that describe the signal, and are of lesser dimension.

Generative models

- Generative models take training samples from some data distribution and learn a model that represents that distribution
- Density estimation:

- Sample generation:

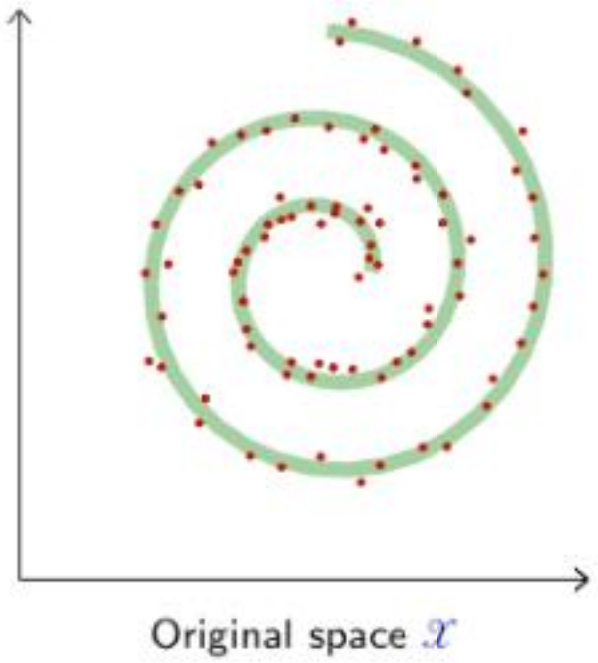
Generative models: intuition



Given seome data in an space $\mathcal{X}$

Figure credit: F. Fleuret, EE-559 Deep learning

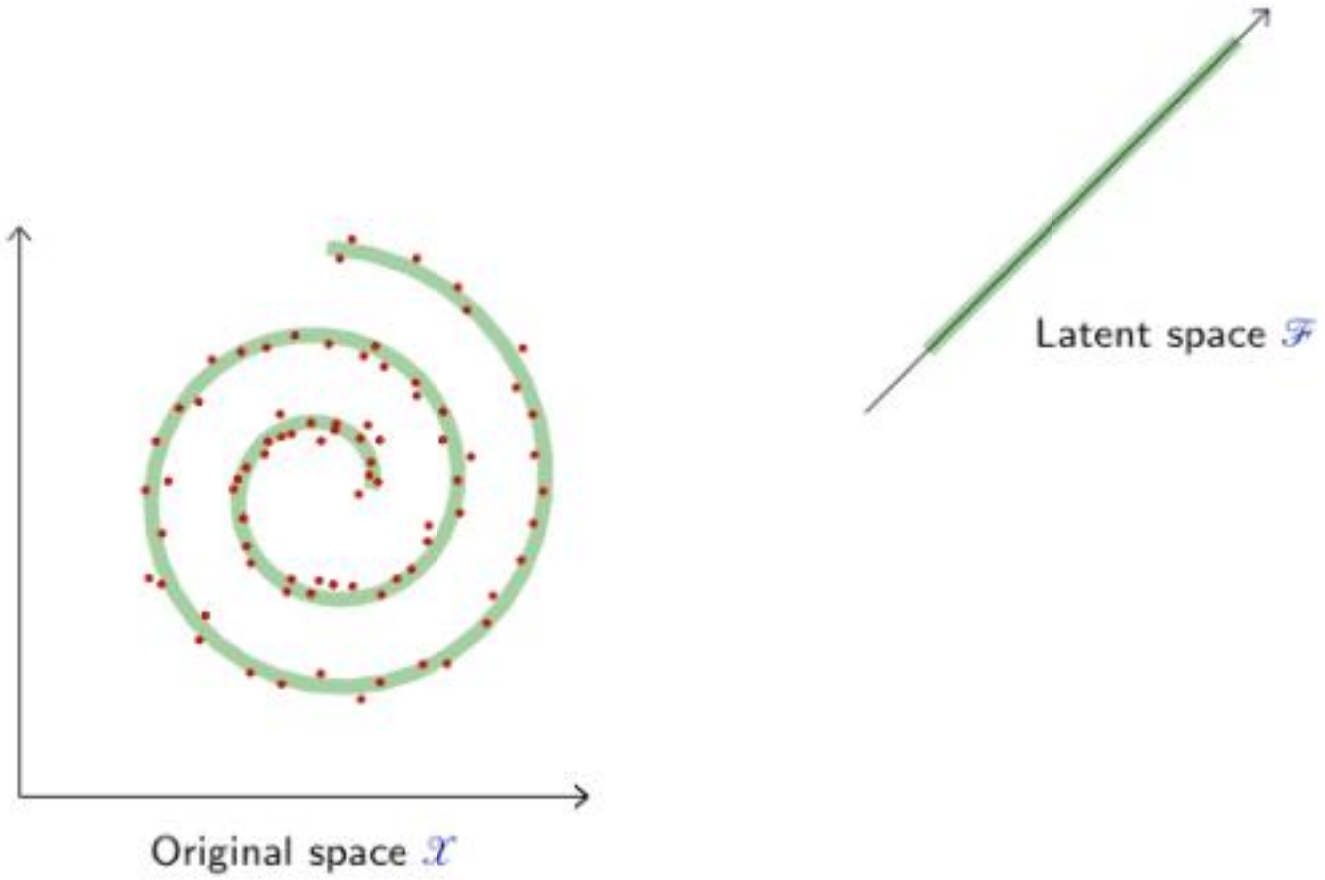
Generative models: intuition



We assume that data has an inner structure that is rather difficult to find and model in the space $\mathcal{X}$

Figure credit: F. Fleuret, EE-559 Deep learning

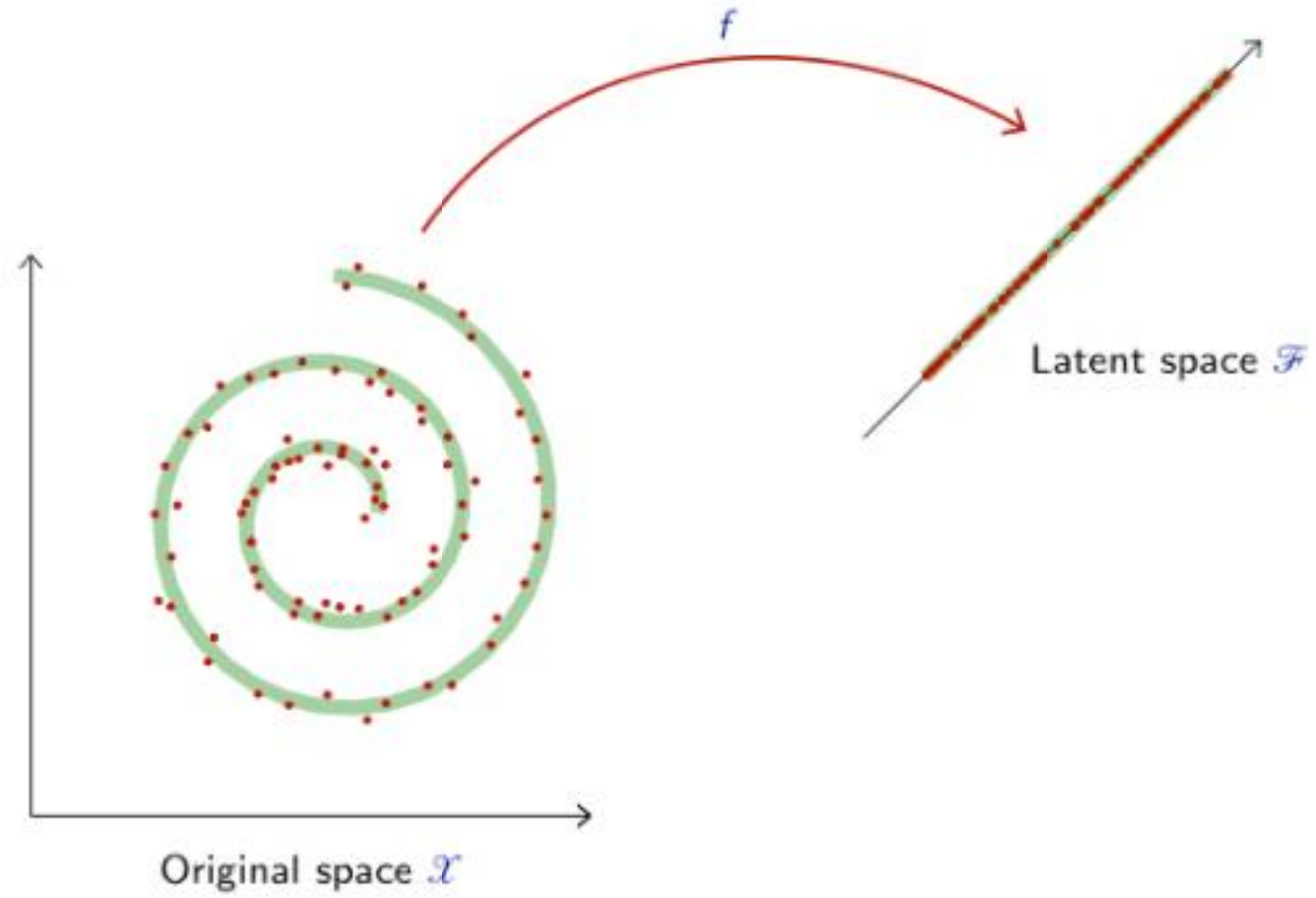
Generative models: intuition



We assume that there is another lower dimensional space $\mathcal{F}$ where the structure of the data in $\mathcal{X}$ is easier to find and model.

Figure credit: F. Fleuret, EE-559 Deep learning

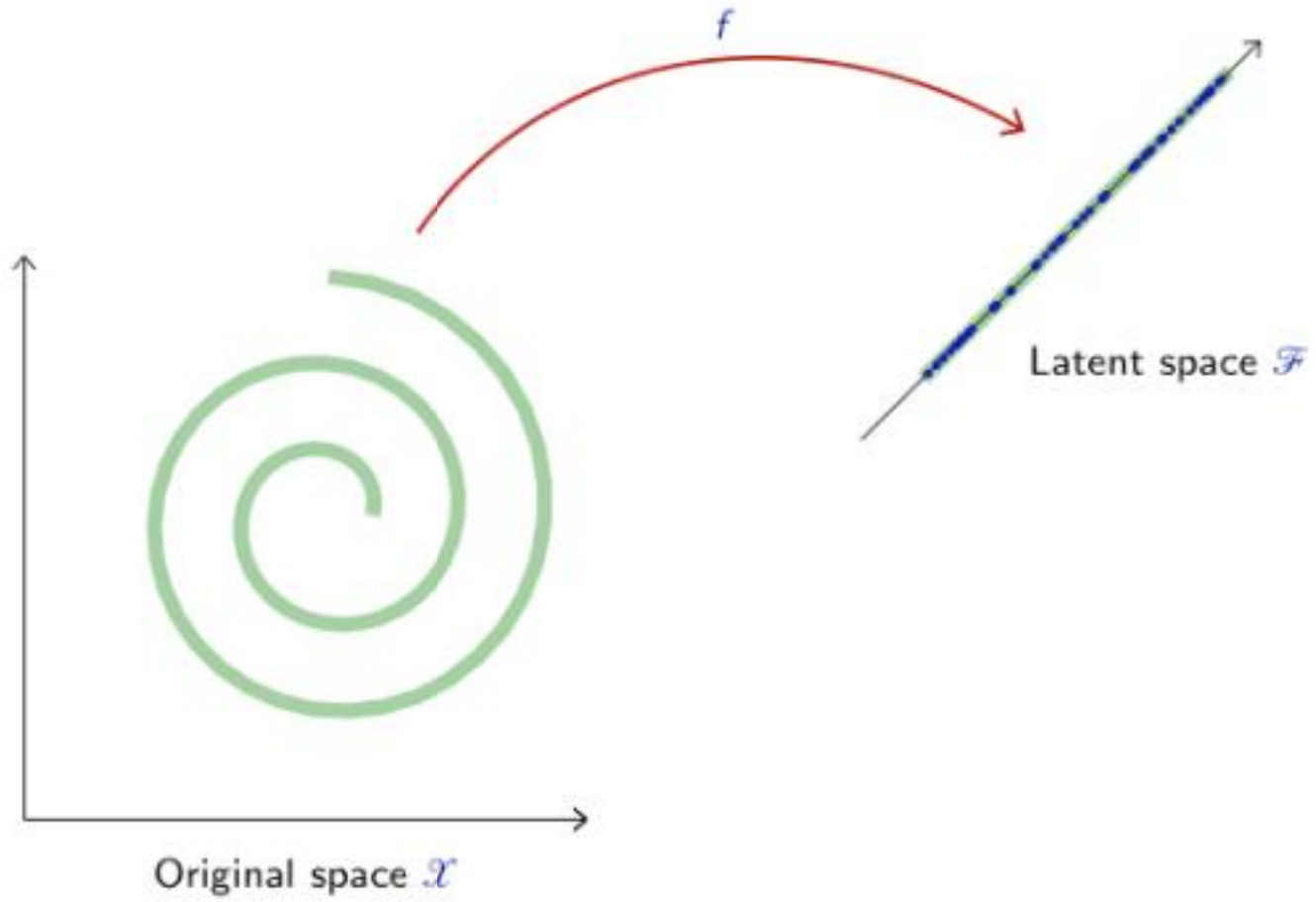
Generative models: intuition



We could then project data from $\mathcal{X}$ to $\mathcal{F}$ via a projection/encoding function f

Figure credit: F. Fleuret, EE-559 Deep learning

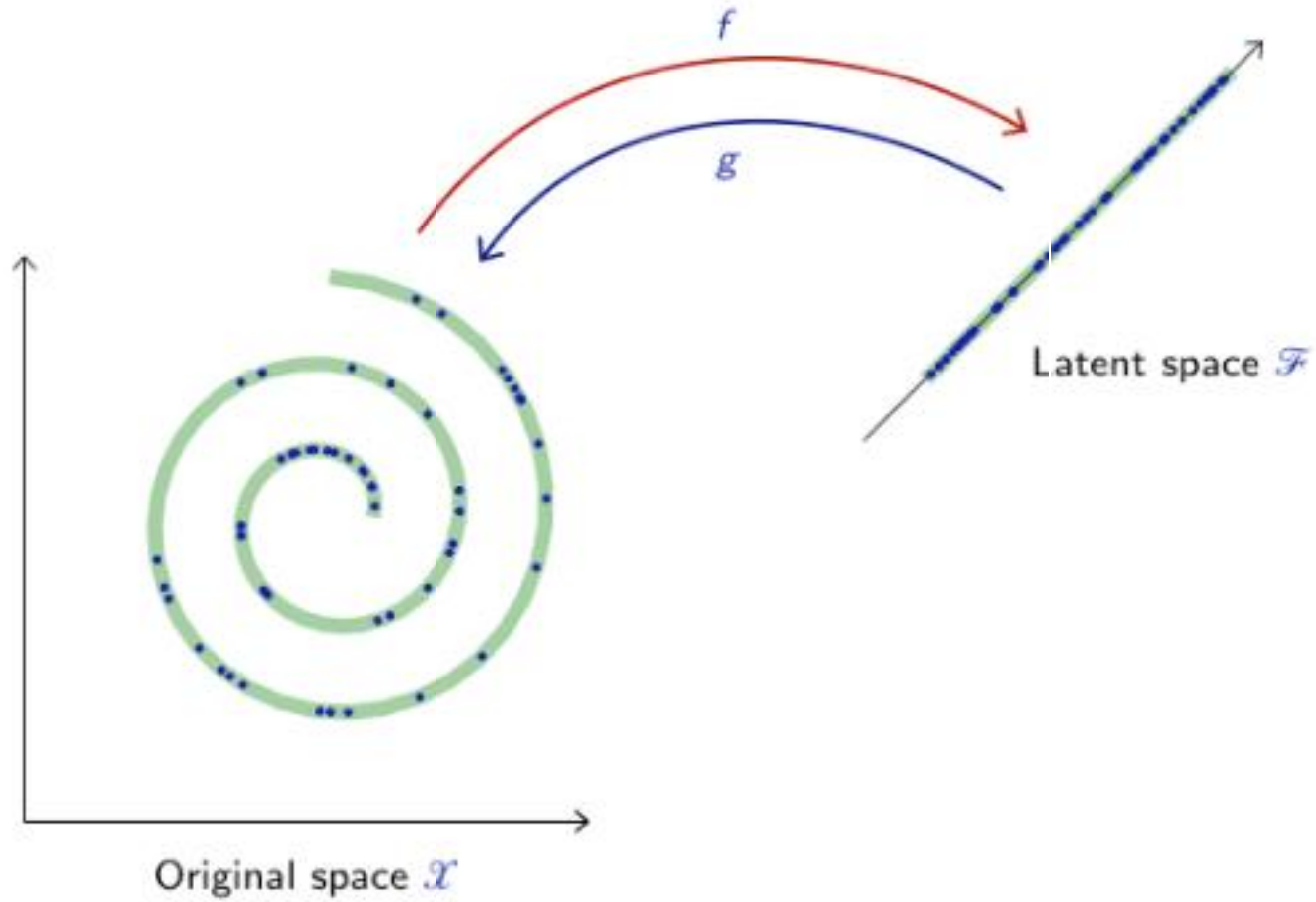
Generative models: intuition



Once our points are projected in $\mathcal{F}$ we can find easier the structure of the data and thus generate new points

Figure credit: F. Fleuret, EE-559 Deep learning

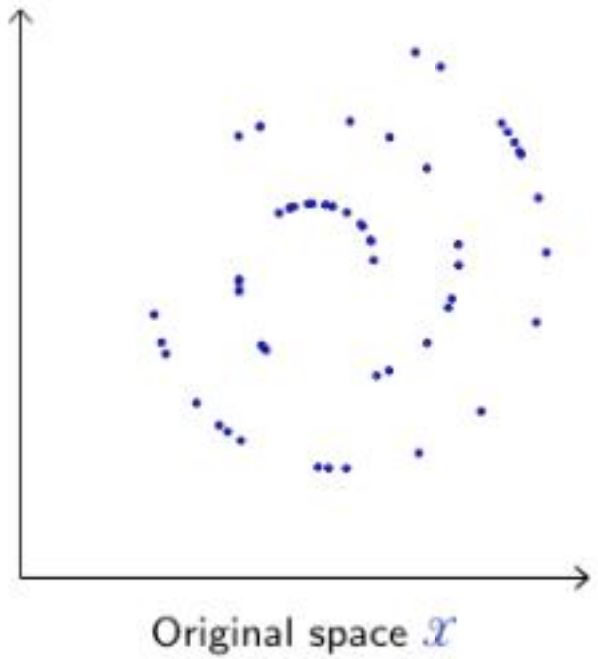
Generative models: intuition



We could then return to our original $\mathcal{X}$ using a projection function g. The newly identifed points in $\mathcal{F}$ will lead to new smaples in $\mathcal{X}$.

Figure credit: F. Fleuret, EE-559 Deep learning

Generative models: intuition



The newly generated samples in $\mathcal{X}$ follow the same structure and allow us to cover better the data space.

Figure credit: F. Fleuret, EE-559 Deep learning

Generative models

- 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.

Generative models

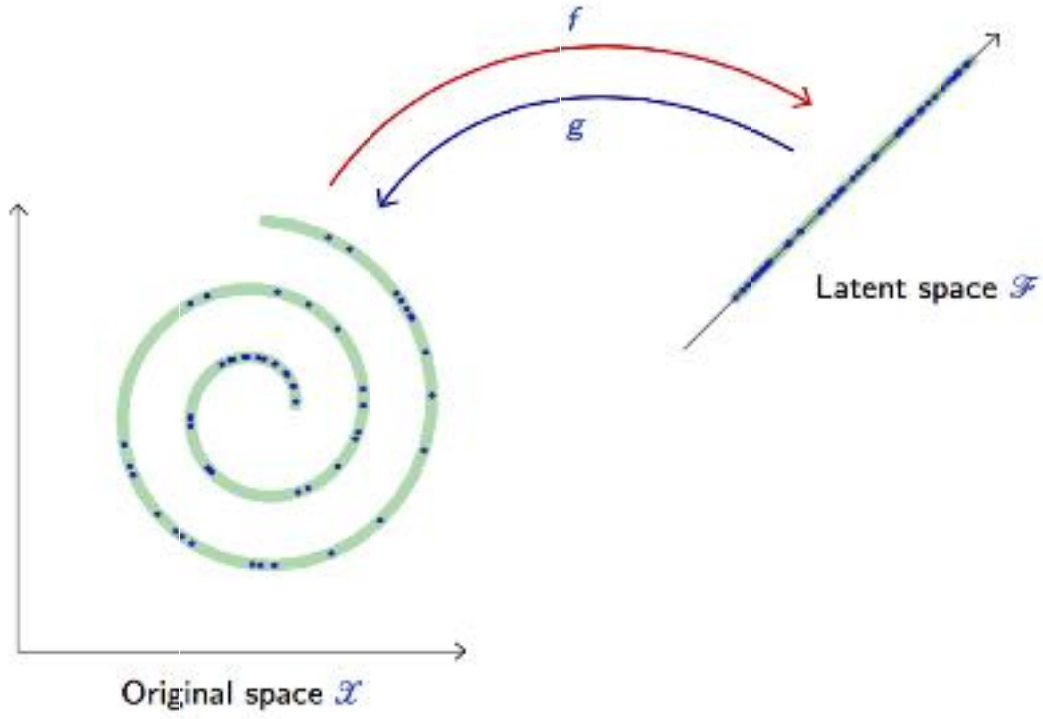
- When dealing with real-world signals, this objective involves the same theoretical and practical issues as for classification or regression: defining the right class of high-dimension models, and optimizing them.
- Regarding synthesis, we saw that deep feed-forward architectures exhibit good generative properties, which motivates their use explicitly for that purpose.

Autoencoders

Definition: An autoencoder combines an encoder f that embeds the original space $\mathcal{X}$ into a latent space of lower dimension $\mathcal{F}$, and a decoder g to map back to $\mathcal{X}$ such that their composition $g \circ f$ is (close to) the identity on the data

Autoencoders

Definition: An autoencoder combines an encoder f that embeds the original space $\mathcal{X}$ into a latent space of lower dimension $\mathcal{F}$, and a decoder g to map back to $\mathcal{X}$ such that their composition $g \circ f$ is (close to) the identity on the data



An autoencoder should capture the statistical dependencies between the signal

Slide credit: F. Fleuret, EE-559 Deep learning

Autoencoders

- A good autoencoder could be characterized with the Mean Squared Error/ L_2 loss:

$$\hat{x} = g \circ f(x)$$

$$L_2 = \|x - \hat{x}\|^2$$

- Train such that features can be used to reconstruct original data
- No labels used
- After training we can remove decoder g and use just features from encoder $f(x)$
  - raw
  - for fine-tuning f in supervised manner

Autoencoders

How can we generate images ?

1. Fully connected layers
    - limited in size of input/ouput images

Autoencoders

How can we generate images ?

1. Fully connected layers
   - limited in size of input/ouput images
2. Convolutions (to the rescue)
   - however convolutions keep or reduce the size of its inputs
   - we need a trick to upsample inputs

Convolutions (again)

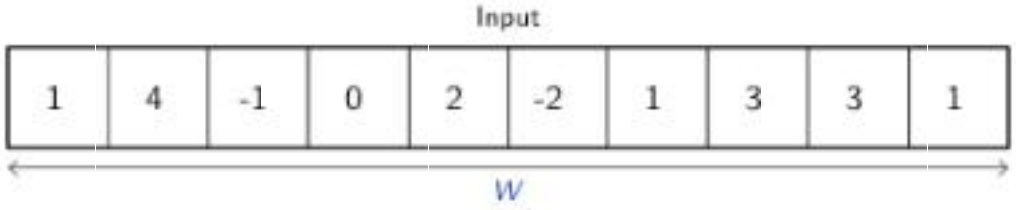


Figure credit: F. Fleuret, EE-559 Deep learning

Convolutions (again)

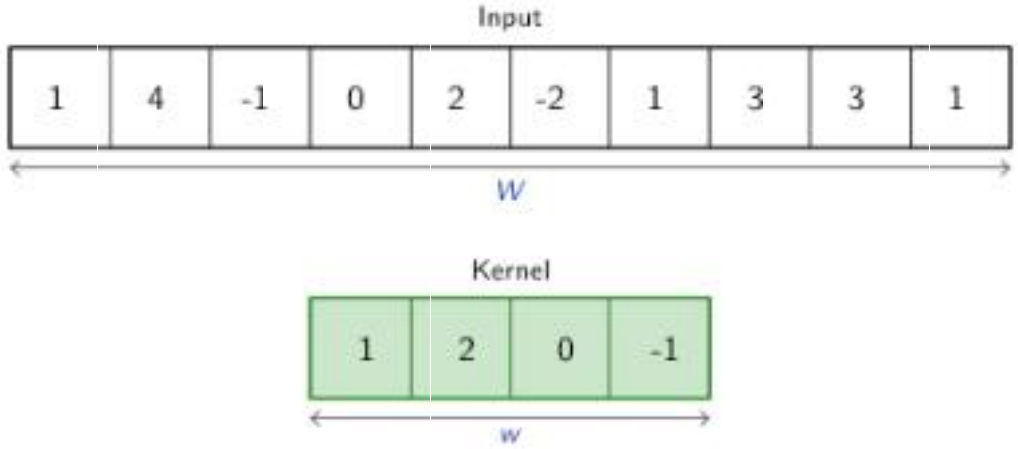


Figure credit: F. Fleuret, EE-559 Deep learning

Convolutions (again)

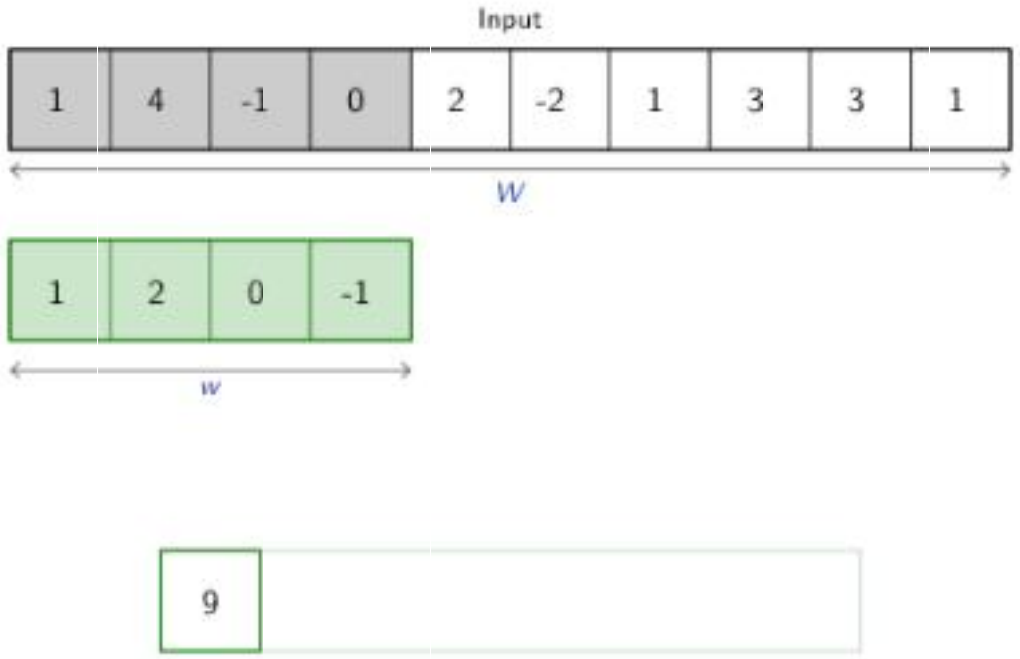


Figure credit: F. Fleuret, EE-559 Deep learning

Convolutions (again)

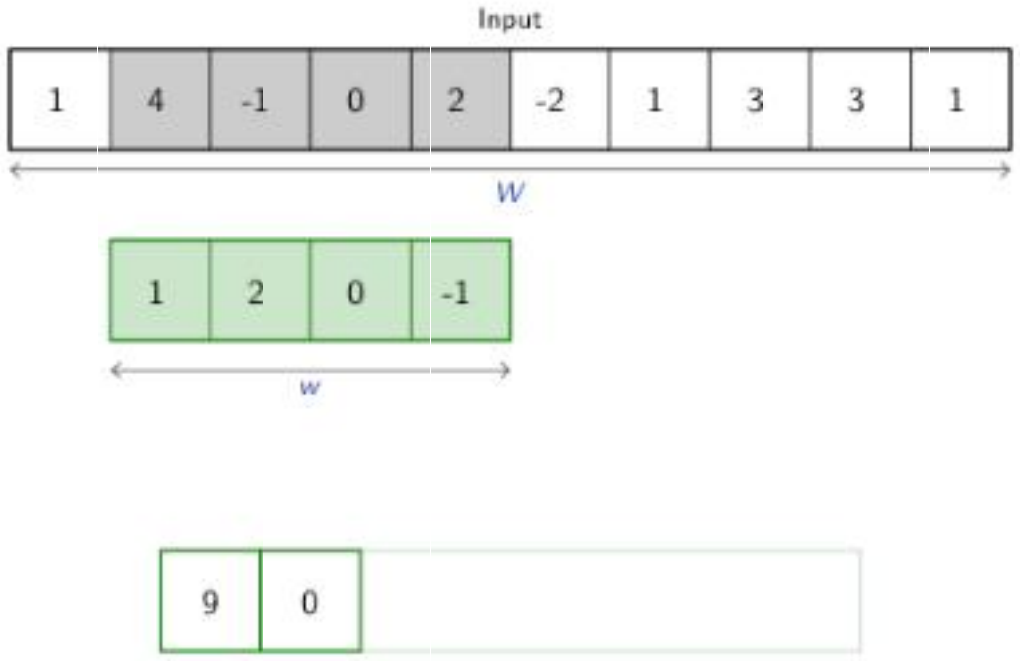


Figure credit: F. Fleuret, EE-559 Deep learning

Convolutions (again)

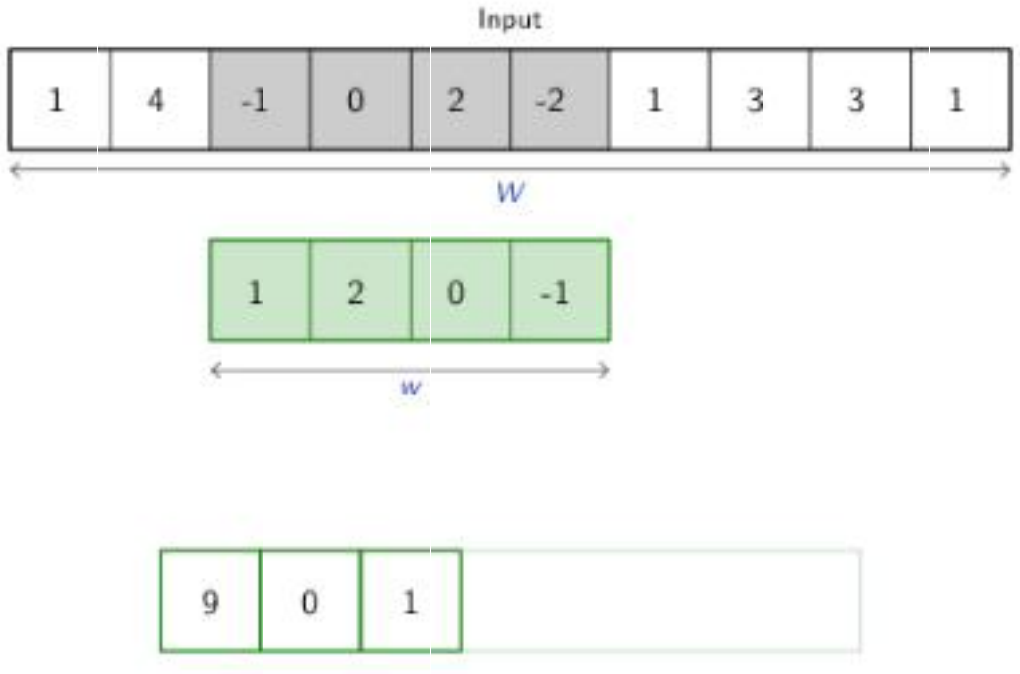


Figure credit: F. Fleuret, EE-559 Deep learning

Convolutions (again)

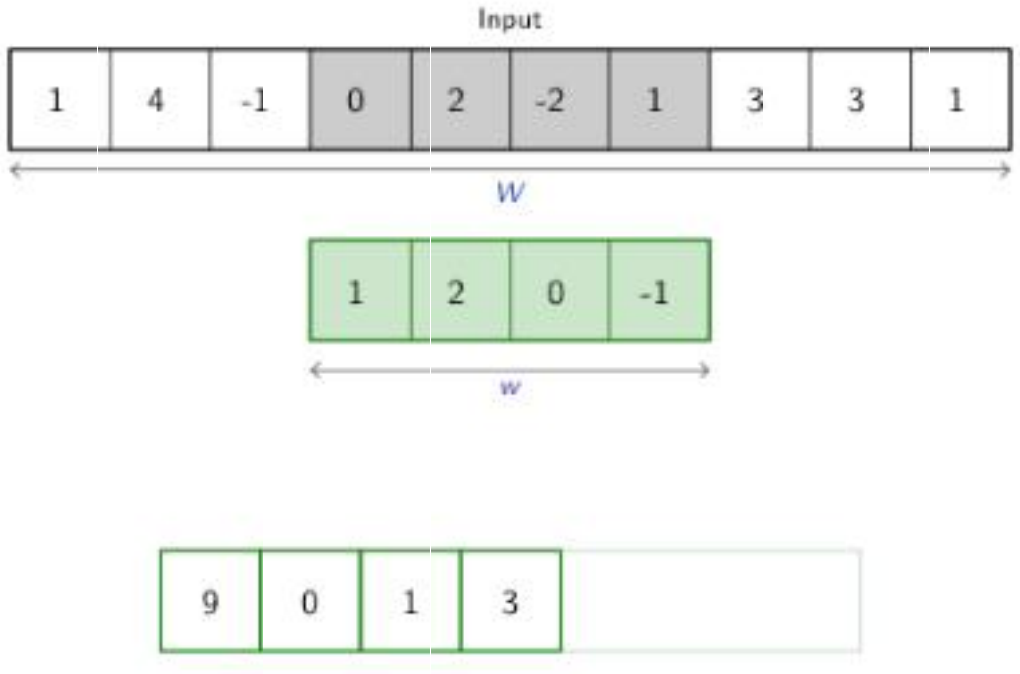


Figure credit: F. Fleuret, EE-559 Deep learning

Convolutions (again)

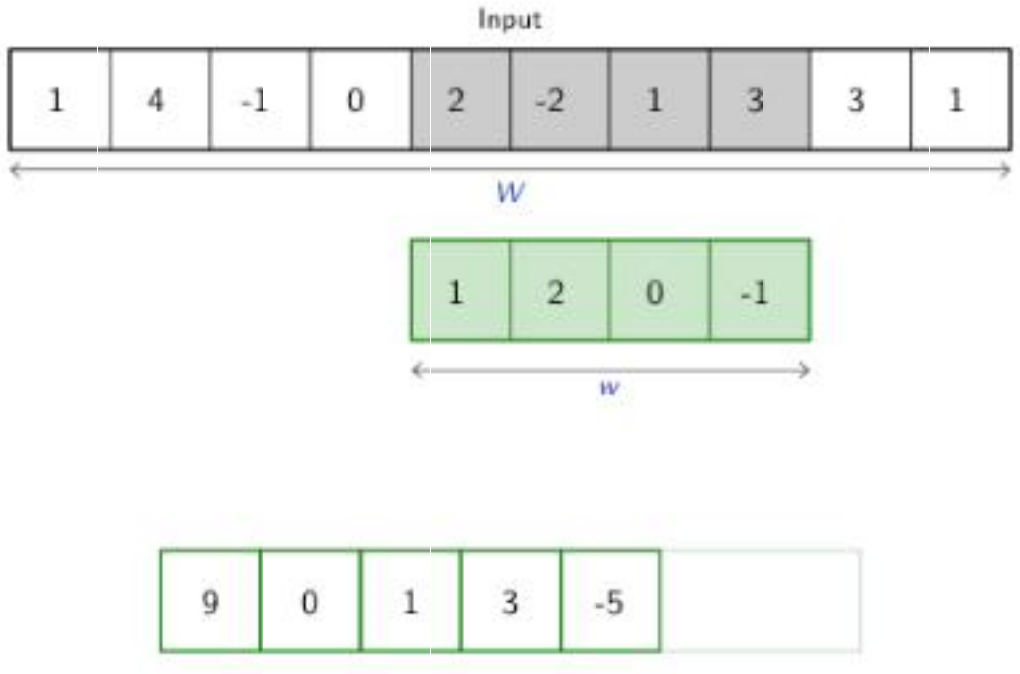


Figure credit: F. Fleuret, EE-559 Deep learning

Convolutions (again)

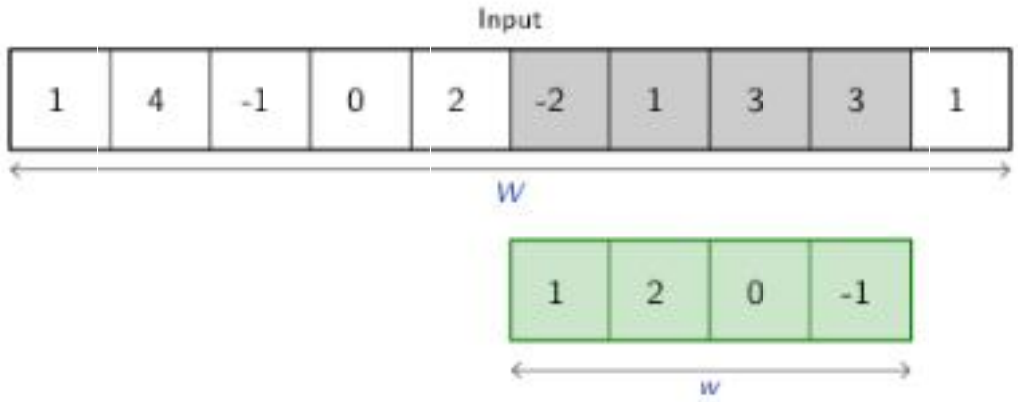


Figure credit: F. Fleuret, EE-559 Deep learning

Convolutions (again)

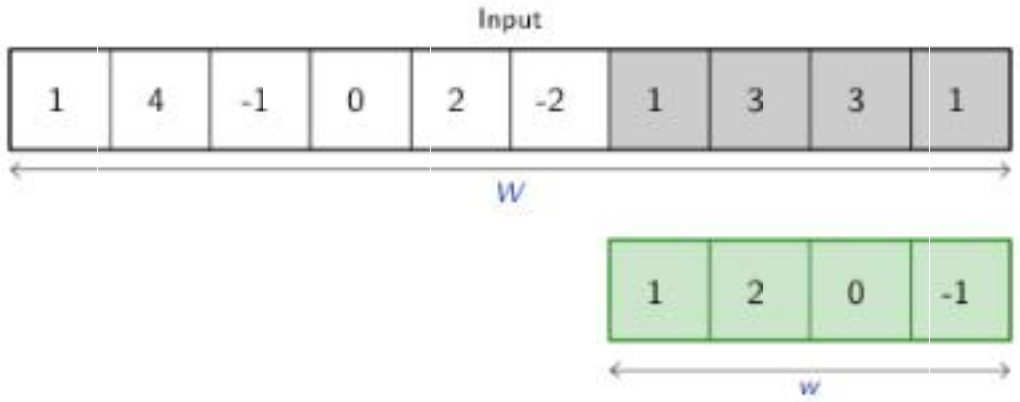
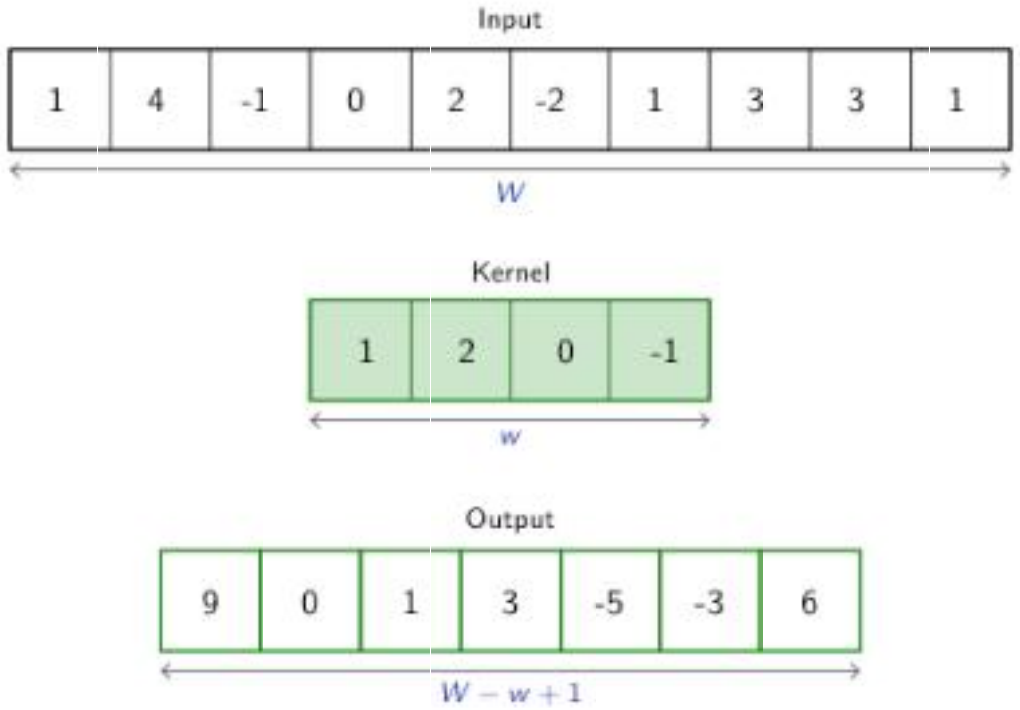


Figure credit: F. Fleuret, EE-559 Deep learning

Convolutions (again)



After convolution the size of the ouput signal $\leq$ size of input signal

Figure credit: F. Fleuret, EE-559 Deep learning

Transposed/UpSampling convolutions

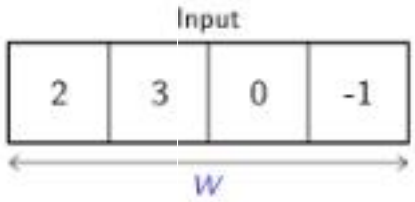
- We've seen that in CNNs in the forward pass the size of the input image was decreasing across convolutional layers
- However, during the backward / backpropagation pass we were returning from small ouputs to our original input
- During backprop we are practically distributing the gradients from one pixel in the feature map to multiple pixels in the previous layer according to the size of the convolutional filters and their stride

$$forward : y = wx$$

$$backprop : \frac{\partial L}{\partial x} = w^T \frac{\partial L}{\partial y}$$

- we apply a similar principle to generate bigger images with convolutions

Transposed/UpSampling convolutions



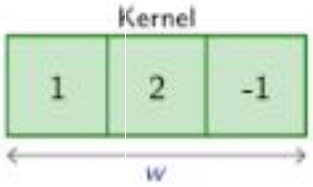


Figure credit: F. Fleuret, EE-559 Deep learning

Transposed/UpSampling convolutions

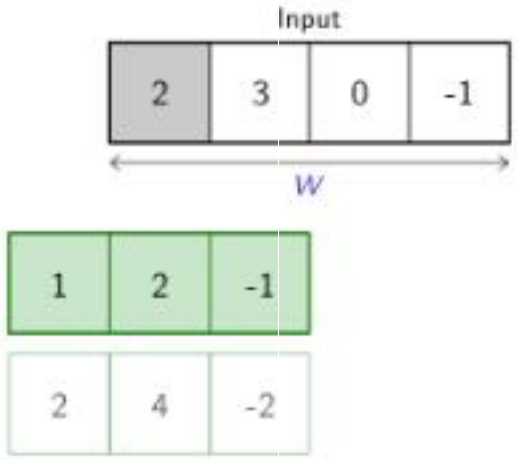


Figure credit: F. Fleuret, EE-559 Deep learning

Transposed/UpSampling convolutions

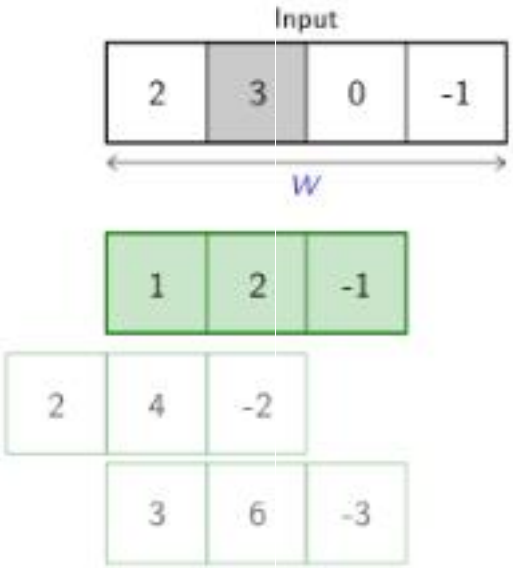


Figure credit: F. Fleuret, EE-559 Deep learning

Transposed/UpSampling convolutions

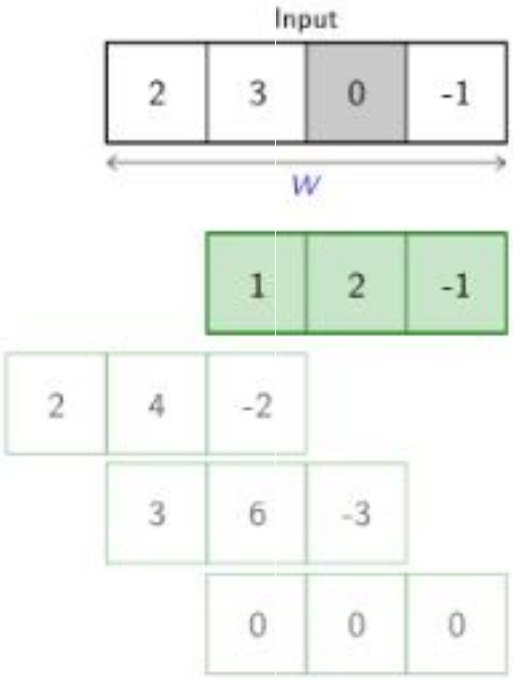


Figure credit: F. Fleuret, EE-559 Deep learning

Transposed/UpSampling convolutions

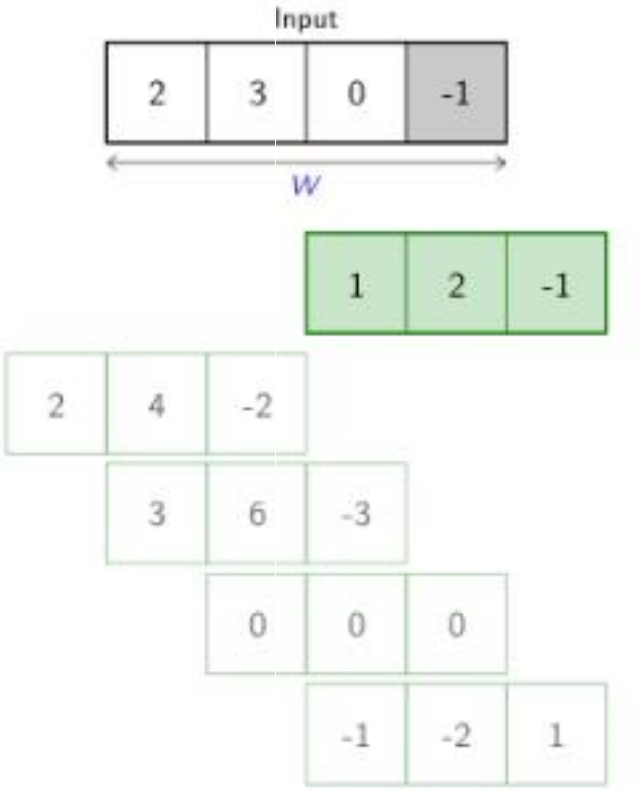


Figure credit: F. Fleuret, EE-559 Deep learning

Transposed/UpSampling convolutions

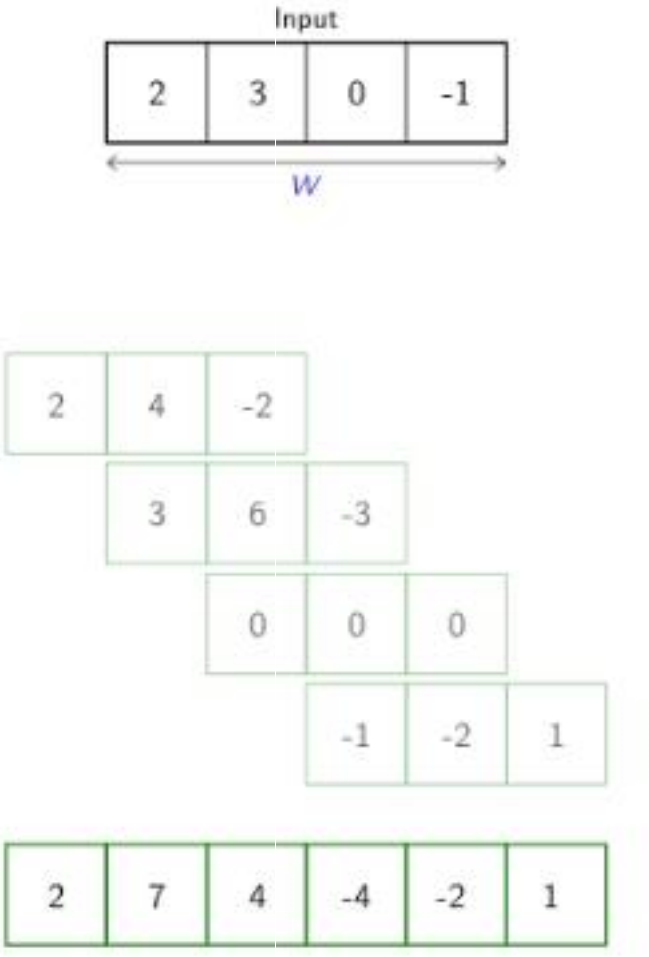
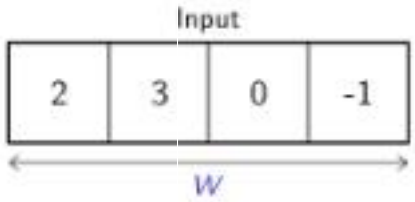
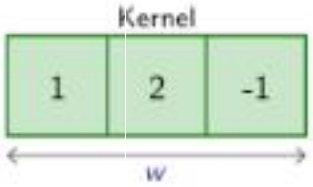


Figure credit: F. Fleuret, EE-559 Deep learning

Transposed/UpSampling convolutions





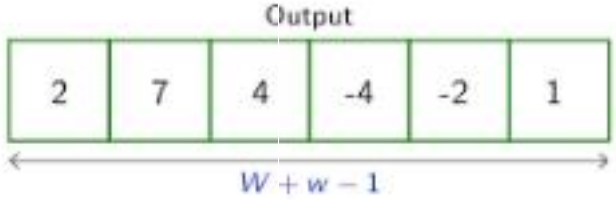
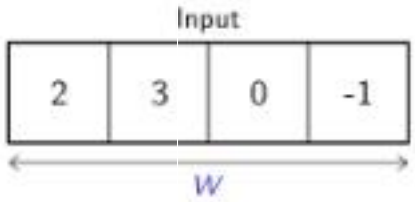
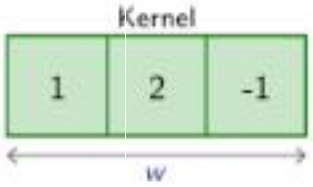


Figure credit: F. Fleuret, EE-559 Deep learning

Transposed/UpSampling convolutions





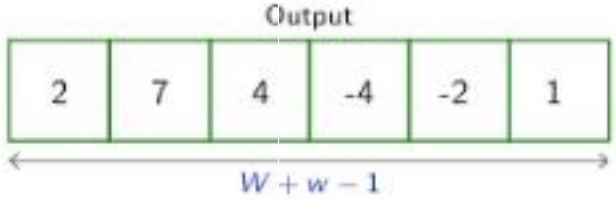


Figure credit: F. Fleuret, EE-559 Deep learning

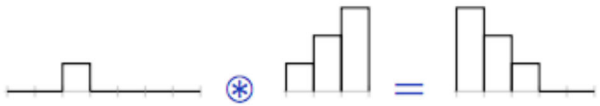
Transposed/UpSampling convolutions

Alternatively we can use normal upsampling of the images (via interpolation) followed by size preserving convolutions.

Transposed/UpSampling convolutions

In pytorch:

```
>>> x = Variable(Tensor([[[0, 0, 1, 0, 0, 0, 0]]]))
>>> k = Variable(Tensor([[[1, 2, 3]]]))
>>> F.conv1d(x, k)
Variable containing:
(0 ,.,.) =
  3 2 1 0 0
[torch.FloatTensor of size 1x1x5]
```



```
>>> F.conv_transpose1d(x, k)
Variable containing:
(0 ,.,.) =
  0 0 1 2 3 0 0 0 0 [torch.FloatTensor of size 1x1x9]
```

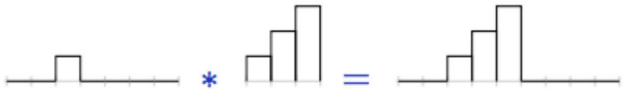


Figure credit: F. Fleuret, EE-559 Deep learning

Back to (deep) autoencoders

Autoencoders

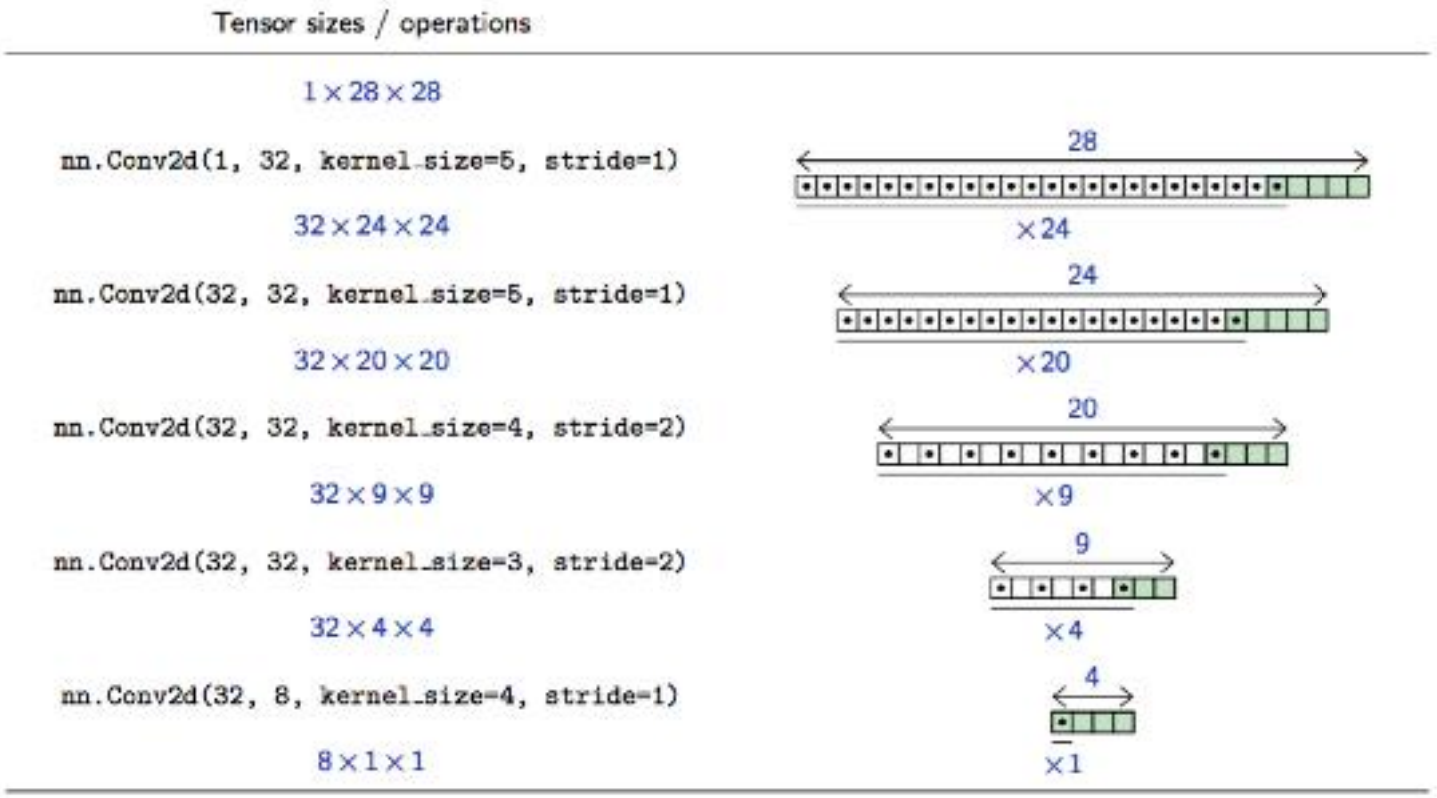
- A deep autoencoder combines an encoder composed of convolutional layers, and a decoder composed of the reciprocal transposed convolution layers.
- To run a simple example on MNIST, we consider the following model, where dimension reduction is obtained through filter sizes and strides > 1, avoiding max-pooling layers.

```
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,
```

```
(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))
```

Autoencoders

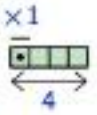
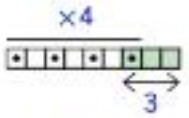
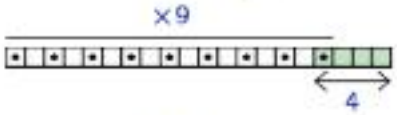
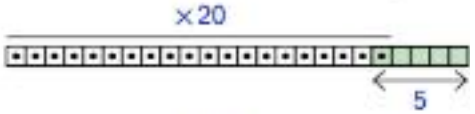
Encoder



Slide credit: F. Fleuret, EE-559 Deep learning

Autoencoders

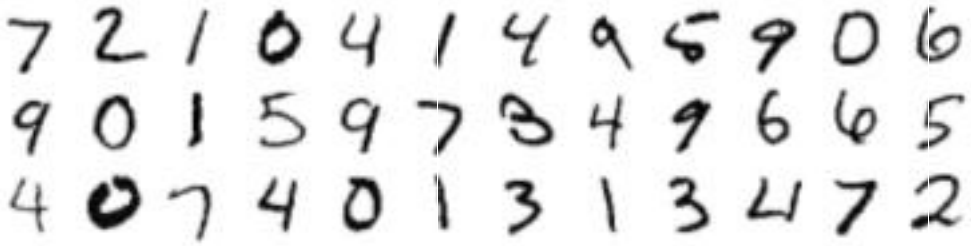
Decoder

Tensor sizes / operations	
$8 \times 1 \times 1$	
`nn.ConvTranspose2d(32, 8, 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(1, 32, kernel_size=5, stride=1)`	
$1 \times 28 \times 28$	

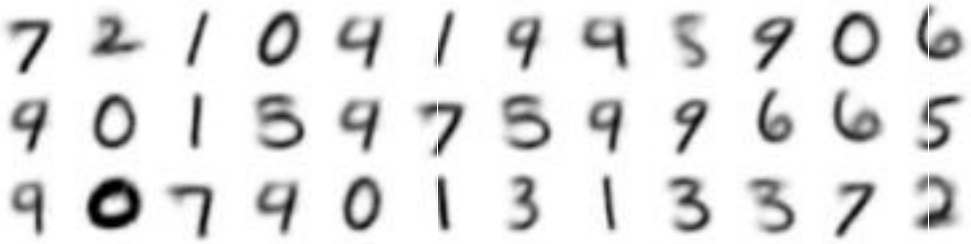
Slide credit: F. Fleuret, EE-559 Deep learning

Autoencoders

X (original samples)



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



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

Figure credit: F. Fleuret, EE-559 Deep learning

Autoencoders

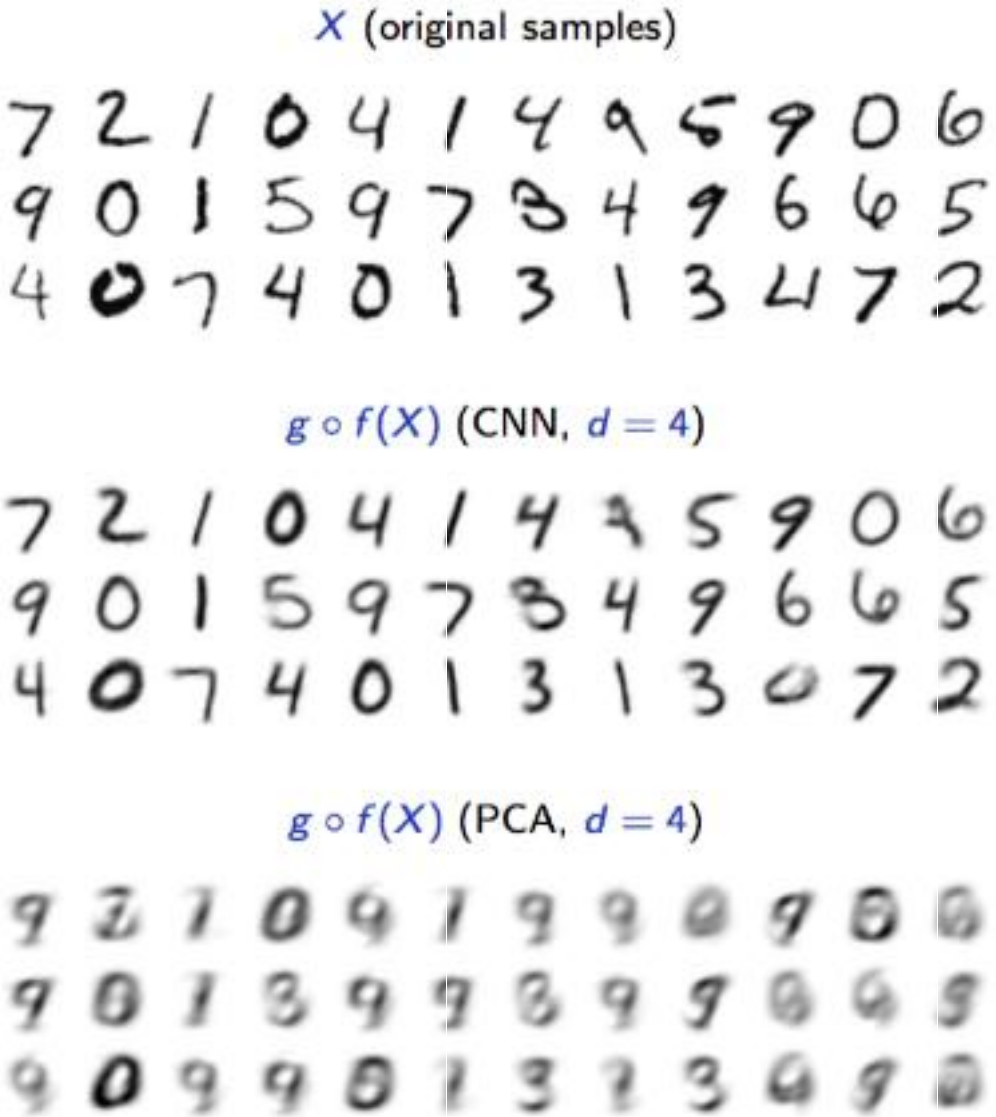


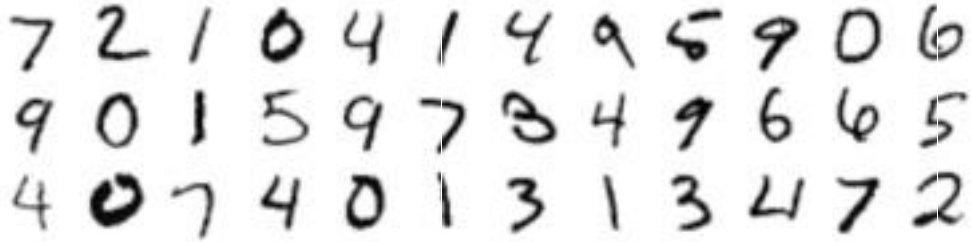
Figure credit: F. Fleuret, EE-559 Deep learning

Autoencoders

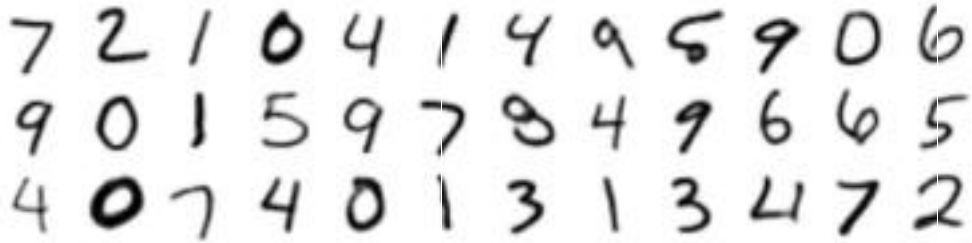
Figure credit: F. Fleuret, EE-559 Deep learning

Autoencoders

X (original samples)



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



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

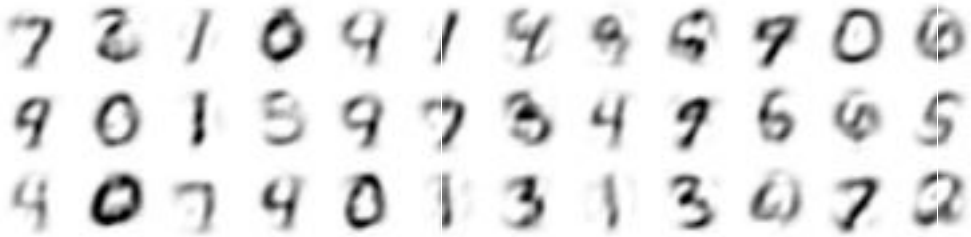
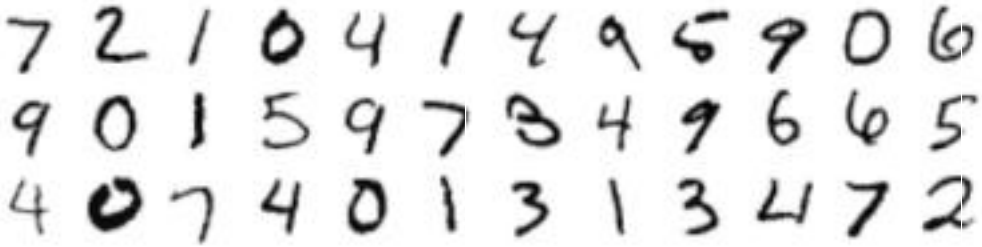


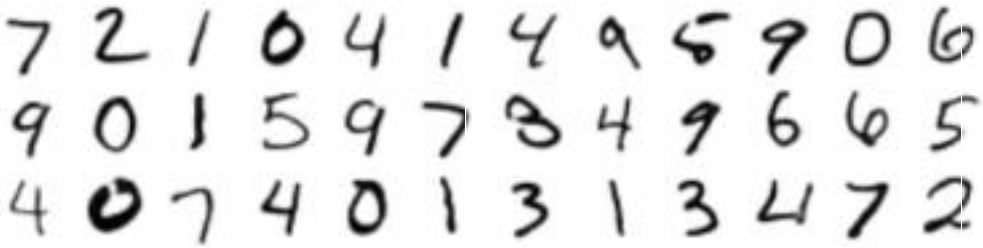
Figure credit: F. Fleuret, EE-559 Deep learning

Autoencoders

X (original samples)



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



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

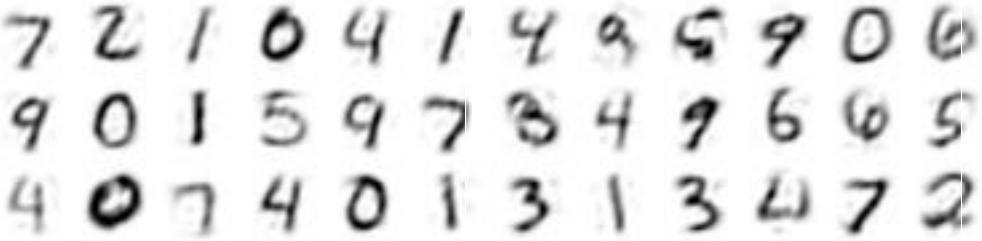
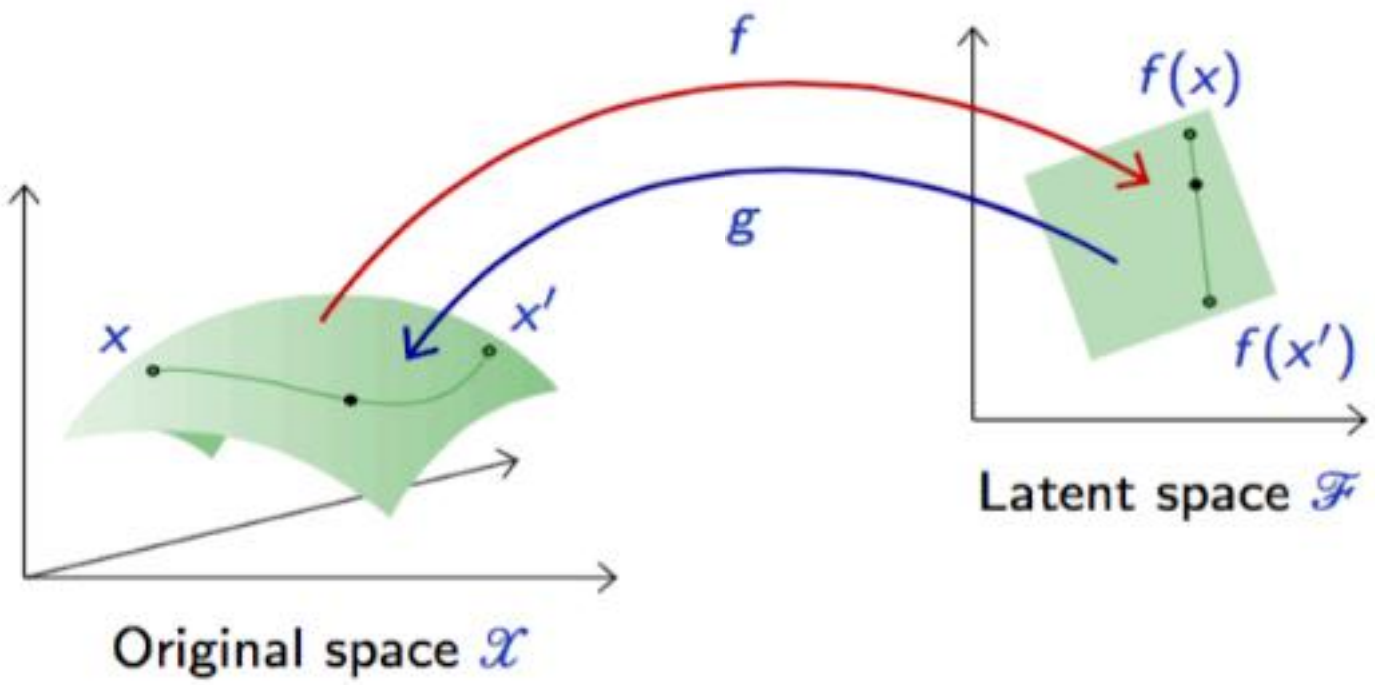


Figure credit: F. Fleuret, EE-559 Deep learning

Autoencoders

Interpolate samples in the latent space



Use a simple interpolation scheme: $\xi(x, x', \alpha) = g((1-\alpha)f(x) + \alpha f(x')); \alpha \in [0, 1]$

Figure credit: F. Fleuret, EE-559 Deep learning

Autoencoders

Interpolate samples in the latent space

Figure credit: F. Fleuret, EE-559 Deep learning

Autoencoders

Interpolate samples in the latent space

Figure credit: F. Fleuret, EE-559 Deep learning

Denoising Autoencoder

Adding a sparsity constraint on activations:

$$||encoder(x)||_1 \sim \rho, \rho = 0.05$$

Learns sparse features, easily interpretable

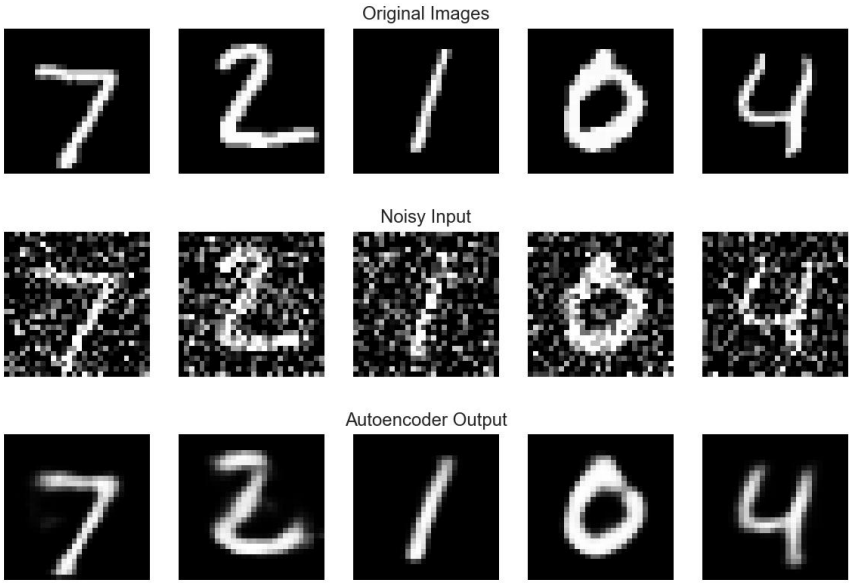
Denoising Autoencoder

Adding a sparsity constraint on activations:

$$||encoder(x)||_1 \sim \rho, \rho = 0.05$$

Learns sparse features, easily interpretable

Denoising Autoencoder: train features for robustness to noise.



Autoencoders

Uses

- After pre-training use the latent code as input to a classifier instead of x
- Semi-supervised learning simultaneous learning of the latent code (on a large, unlabeled dataset) and the classifier (on a smaller, labeled dataset)
- Use decoder $D(x)$ as a Generative model: generate samples from random noise

Autoencoders

Limitations

- Direct autoencoder fails to capture good representations for complex data such as images
- The generative model is usually of very poor quality (very blurry for images for instance)
- Don't have a specific distribution in the latent space
- Sampling becomes difficult

Autoencoders

Reality check

For image features, ImageNet pretraining is still much better than unsupervised models

Pretraining Method	Supervision	Pretraining time	Classification	Detection	Segmentation
ImageNet [26]	1000 class labels	3 days	**78.2%**	**56.8%**	**48.0%**
Random Gaussian	initialization	< 1 minute	53.3%	43.4%	19.8%
Autoencoder	-	14 hours	53.8%	41.9%	25.2%
Agrawal *et al.* [1]	egomotion	10 hours	52.9%	41.8%	-
Doersh *et al.* [7]	context	4 weeks	55.3%	**46.6%**	-
Wang *et al.* [39]	motion	1 week	**58.4%**	44.0%	-
Ours	context	14 hours	56.5%	44.5%	**29.7%**

- Results shown after fine-tuned the network on Pascal VOC dataset
- The "ours" method is feature representation based on context inpainting

Pathak, Deepak, et al. Context encoders: Feature learning by inpainting. CVPR 2016.

Variational Autoencoders (VAEs)

VAE: Preliminaries and notation

Bayes' Theorem

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$$

- $p(\mathbf{z}|\mathbf{x})$ = the posterior probability: given the image, what is the probability that this is of a cat?
- $p(\mathbf{x}|\mathbf{z})$ = the likelihood: given a value of $\mathbf{z}$ this computes how "probable" this image $\mathbf{x}$ is under that category ({"is-a-cat" / "is-not-a-cat"})
- $p(\mathbf{z})$ = the prior probability: captures any prior information we know about $\mathbf{z}$

VAE

Instead of training an autoencoder and modeling the distribution of $\mathbf{z}$, we can try an alternative approach:

Impose a distribution for $\mathbf{z}$ and then train a decoder g so that $g(\mathbf{z})$ matches the training data.

In other words: put some restrictions in z such that our data points x are distributed in a latent space following a probability density function $P(Z)$, e.g $N(0, I)$

This allows for sampling $\mathbf{z}$ to generate new data points $\mathbf{x}$

Variational Autoencoders (VAE)

Assume the data samples $\mathbf{x}^{(i)}$ are generated by the model:

$$p_{\theta^*}(\mathbf{x}, \mathbf{z}) = p_{\theta^*}(\mathbf{z}) \cdot p_{\theta^*}(\mathbf{x}|\mathbf{z})$$

Variational Autoencoders (VAE)

Assume the data samples $\mathbf{x}^{(i)}$ are generated by the model:

$$p_{\theta^*}(\mathbf{x}, \mathbf{z}) = p_{\theta^*}(\mathbf{z}) \cdot p_{\theta^*}(\mathbf{x}|\mathbf{z})$$

- $\mathbf{x}$ is an observed r.v. with values in $\mathbb{R}^n$;
- $\mathbf{z}$ is a latent r.v. with values in $\mathbb{R}^d$;
- True continuous parameters θ^* are unknown;
- Estimate parameters θ from data $\mathbf{x}^{(i)}$ by maximizing the marginal likelihood (MLE):

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z}) \cdot p_\theta(\mathbf{x}|\mathbf{z}) \, \mathrm{d}\mathbf{z}$$

Variational Autoencoders (VAE)

Assume the data samples $\mathbf{x}^{(i)}$ are generated by the model:

$$p_{\theta^*}(\mathbf{x}, \mathbf{z}) = p_{\theta^*}(\mathbf{z}) \cdot p_{\theta^*}(\mathbf{x}|\mathbf{z})$$

- $\mathbf{x}$ is an observed r.v. with values in $\mathbb{R}^n$;
- $\mathbf{z}$ is a latent r.v. with values in $\mathbb{R}^d$;
- True continuous parameters θ^* are unknown;
- Estimate parameters θ from data $\mathbf{x}^{(i)}$ by maximizing the marginal likelihood (MLE):

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z}) \cdot p_\theta(\mathbf{x}|\mathbf{z}) \, \mathrm{d}\mathbf{z}$$

But this high dimensional integral cannot be estimated efficently.

Variational Autoencoders (VAE)

Assume the data samples $\mathbf{x}^{(i)}$ are generated by the model:

$$p_{\theta^*}(\mathbf{x}, \mathbf{z}) = p_{\theta^*}(\mathbf{z}) \cdot p_{\theta^*}(\mathbf{x}|\mathbf{z})$$

- $\mathbf{x}$ is an observed r.v. with values in $\mathbb{R}^n$;
- $\mathbf{z}$ is a latent r.v. with values in $\mathbb{R}^d$;
- True continuous parameters θ^* are unknown;
- Estimate parameters θ from data $\mathbf{x}^{(i)}$ by maximizing the marginal likelihood (MLE):

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z}) \cdot p_\theta(\mathbf{x}|\mathbf{z}) \, \mathrm{d}\mathbf{z}$$

But this high dimensional integral cannot be estimated efficently.

Kingma and Welling, Auto-Encoding Variational Bayes, ICLR 2014

Variational Autoencoders (VAE)

Introduce approximate (variational) posterior on the latent variable: $q_\phi(\mathbf{z}|\mathbf{x})$ with continuous parameters ϕ.

Variational Autoencoders (VAE)

Introduce approximate (variational) posterior on the latent variable: $q_\phi(\mathbf{z}|\mathbf{x})$ with continuous parameters ϕ.

We can rewrite the log-likelihood as:

$$log\, p_\theta(\mathbf{x}^{(i)}) = \mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\theta, \phi; \mathbf{x}^{(i)})$$

with:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = -\mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})} log\, p_\theta(\mathbf{x}^{(i)}|\mathbf{z})$$

Variational Autoencoders (VAE)

Introduce approximate (variational) posterior on the latent variable: $q_\phi(\mathbf{z}|\mathbf{x})$ with continuous parameters ϕ.

We can rewrite the log-likelihood as:

$$log\, p_\theta(\mathbf{x}^{(i)}) = \mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\theta, \phi; \mathbf{x}^{(i)})$$

with:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = -\mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})} log\, p_\theta(\mathbf{x}^{(i)}|\mathbf{z})$$

- $\mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z}|\mathbf{x}^{(i)}))$ is positive but unknown. It depends on the quality of our approximation.

Variational Autoencoders (VAE)

Introduce approximate (variational) posterior on the latent variable: $q_\phi(\mathbf{z}|\mathbf{x})$ with continuous parameters ϕ.

We can rewrite the log-likelihood as:

$$log\, p_\theta(\mathbf{x}^{(i)}) = \mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\theta, \phi; \mathbf{x}^{(i)})$$

with:

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = -\mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})} log\, p_\theta(\mathbf{x}^{(i)}|\mathbf{z})$$

- $\mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z}|\mathbf{x}^{(i)}))$ is positive but unknown. It depends on the quality of our approximation.
- $\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)})$ is a lower bound to maximize w.r.t. θ and ϕ.

Variational Autoencoders (VAE)

- Assume prior distribution for latent variable $\mathbf{z}$:

$$p_\theta(\mathbf{z}) = \mathcal{N}(0, 1)$$

Variational Autoencoders (VAE)

- Assume prior distribution for latent variable $\mathbf{z}$:

$$p_\theta(\mathbf{z}) = \mathcal{N}(0, 1)$$

- Parametrize $p_\theta(\mathbf{x}|\mathbf{z})$ by a neural network f_θ (decoder):

$$p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) = \mathcal{N}(f_\theta(\mathbf{z}), 1)$$

Variational Autoencoders (VAE)

- Assume prior distribution for latent variable $\mathbf{z}$:

$$p_\theta(\mathbf{z}) = \mathcal{N}(0, 1)$$

- Parametrize $p_\theta(\mathbf{x}|\mathbf{z})$ by a neural network f_θ (decoder):

$$p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) = \mathcal{N}(f_\theta(\mathbf{z}), 1)$$

- Parametrize $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ by a neural network with two heads μ_ϕ and σ_ϕ (encoder):

$$q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mu_\phi(\mathbf{x}^{(i)}), \sigma_\phi(\mathbf{x}^{(i)}))$$

Variational Autoencoders (VAE)

- Assume prior distribution for latent variable $\mathbf{z}$:

$$p_\theta(\mathbf{z}) = \mathcal{N}(0, 1)$$

- Parametrize $p_\theta(\mathbf{x}|\mathbf{z})$ by a neural network f_θ (decoder):

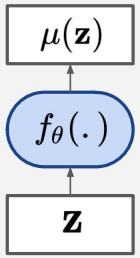
$$p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) = \mathcal{N}(f_\theta(\mathbf{z}), 1)$$

- Parametrize $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ by a neural network with two heads μ_ϕ and σ_ϕ (encoder):

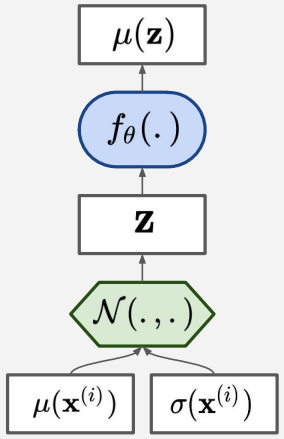
$$q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \mathcal{N}(\mu_\phi(\mathbf{x}^{(i)}), \sigma_\phi(\mathbf{x}^{(i)}))$$

- Reparametrization trick:

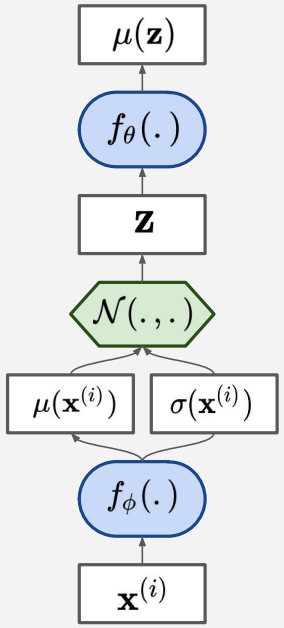
$$\mathbf{z} = \mu_\phi(\mathbf{x}^{(i)}) + \sigma_\phi(\mathbf{x}^{(i)}) \cdot \epsilon \quad \text{with} \quad \epsilon \sim \mathcal{N}(0, 1)$$



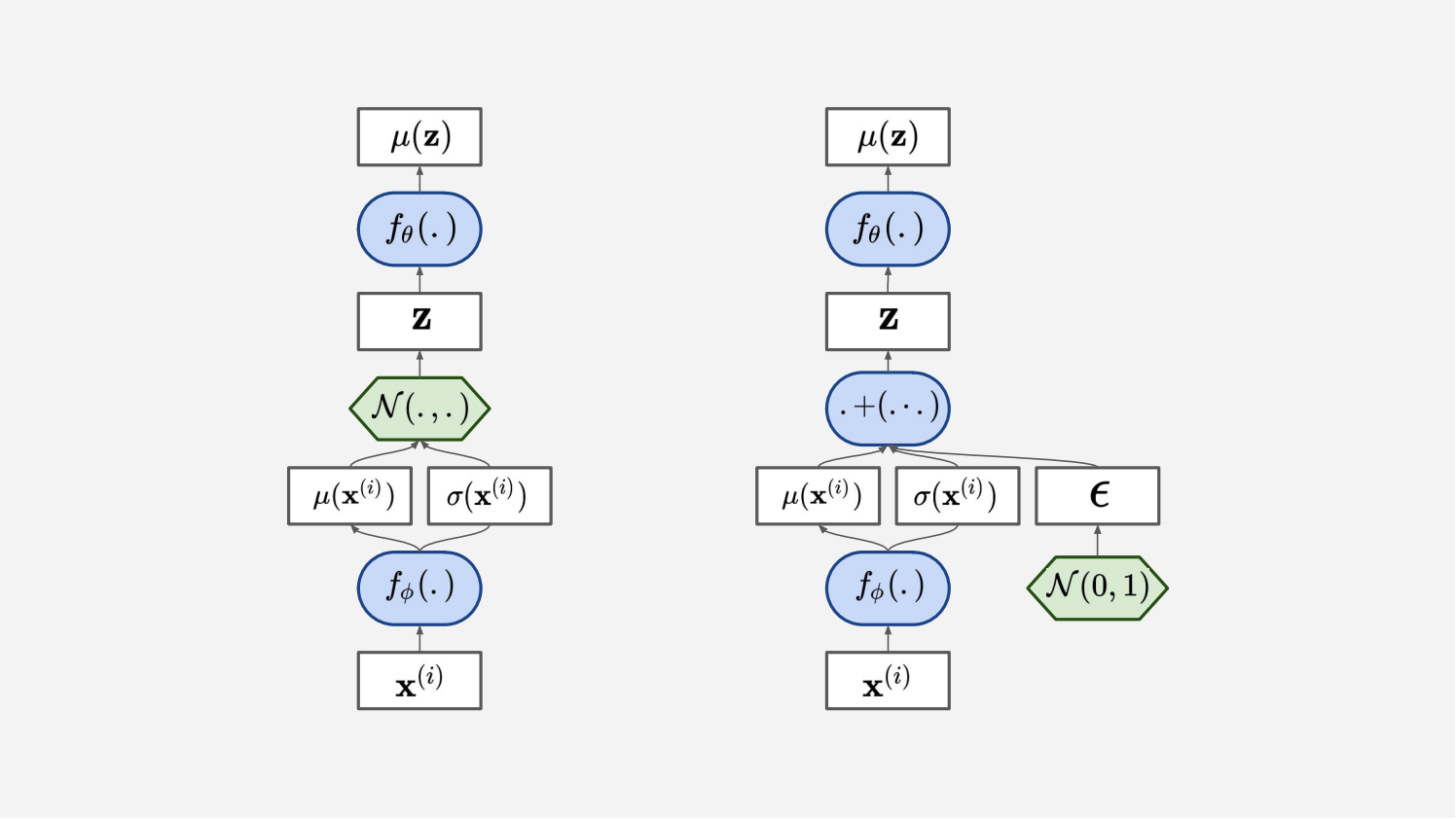
The decoder f_θ defines the likelihood of data.



The latent variable $\mathbf{z}$ is stochastic.



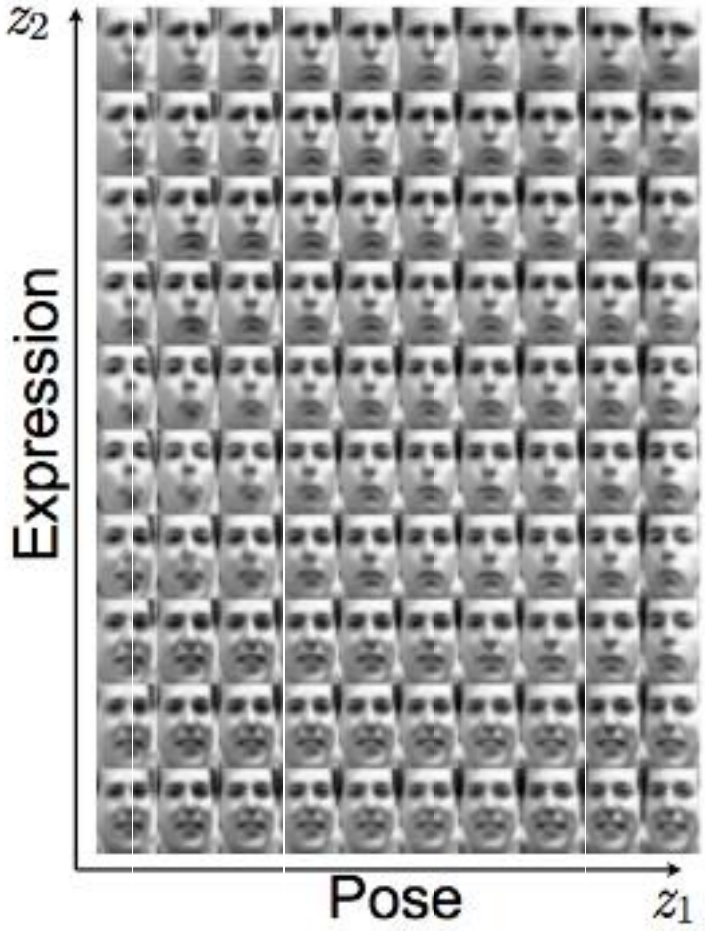
The encoder f_ϕ defines an approximate posterior on $\mathbf{z}$.

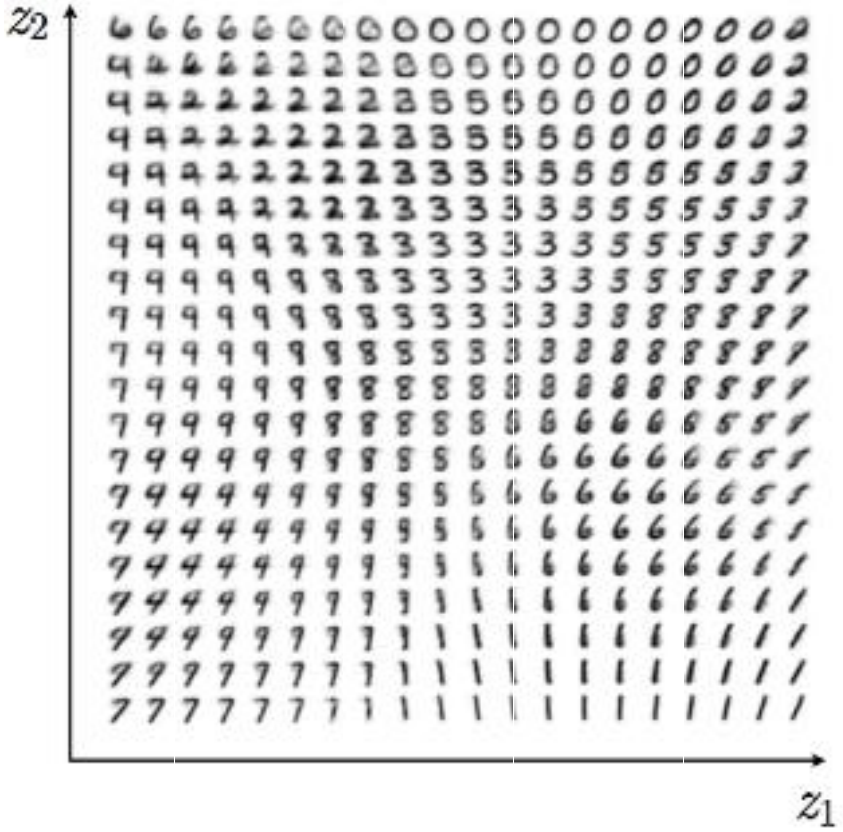


The reparametrization makes the objective differentiable wrt. θ & ϕ.

VAE

Generating data

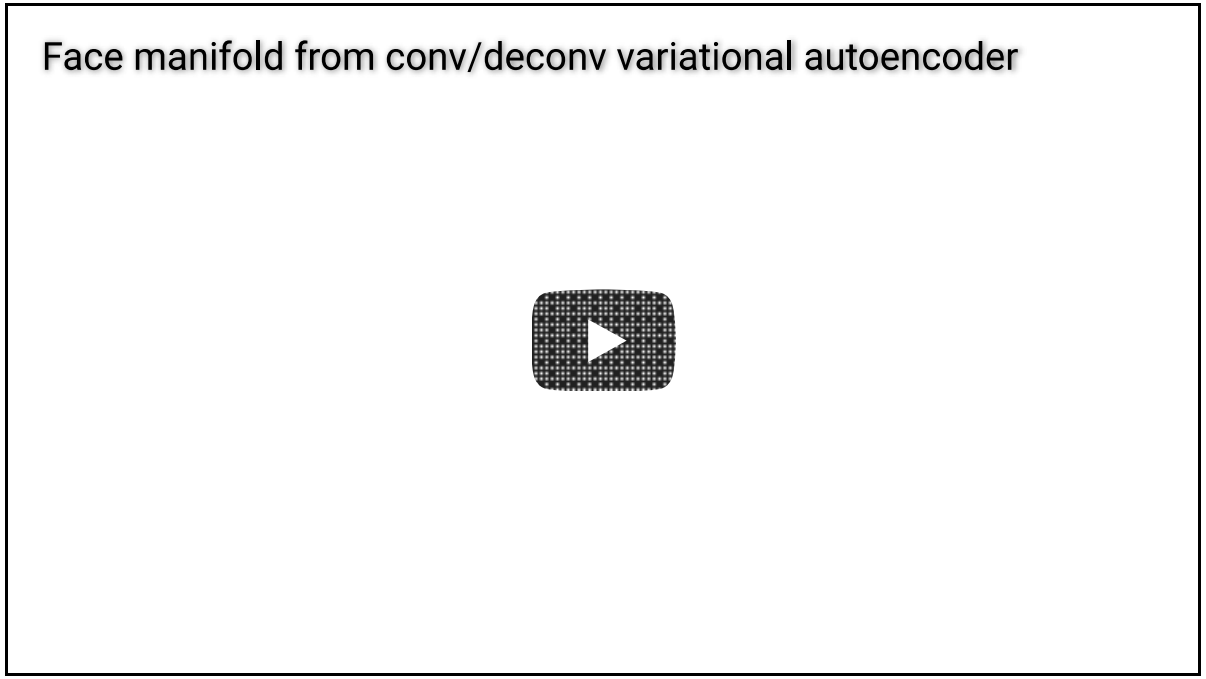




VAE

Generating data from Labeled Faces in the Wild (LFW) and ImageNet (small)

VAE



- conv/deconv VAE trained by Alec Radford in 2015 on Labeled Faces in the Wild (LFW) dataset, 2h on single GTX 980]

VAE

```
class VariationalAutoEncoder:
  def __init__(self, dim_x, dim_z, dim_hidden):
    self.dim_x = dim_x
    self.dim_z = dim_z
    self.dim_hidden = dim_hidden

    self.encoder_f = nn.Sequential(
        nn.Linear(self.dim_x, self.dim_hidden),
        nn.ReLU(),
        nn.Linear(self.dim_hidden , self.dim_z * 2)
    )

    self.decoder_g = nn.Sequential(
        nn.Linear(self.dim_z, self.dim_hidden),
        nn.ReLU(),
        nn.Linear(self.dim_hidden , self.dim_x * 2)
)
```

VAE

```
def log_g_x_given_z(self, z, x):
  mu, logvar = self.decoder_g(z).split(self.dim_x, 1)
  u = - (x - mu).pow(2) / (2 * logvar.exp()) - logvar.mul(0.5)
  log_p = u.sum(1) - 0.5 * math.log(2 * math.pi)
  return log_p

def kl_f_given_x_to_g(self, x):
  mu, logvar = self.encoder_f(x).split(self.dim_z, 1)
  u = - 0.5 * (1 + logvar - mu.pow(2) - logvar.exp())
  kl = u.sum(1)
  return kl

def sample_from_f_given_x(self, x):
  mu, logvar = self.encoder_f(x).split(self.dim_z, 1)
  std = logvar.mul(0.5).exp()
  u = Variable(mu.data.new(mu.size()).normal_())
  z = u * std + mu
  return z
```

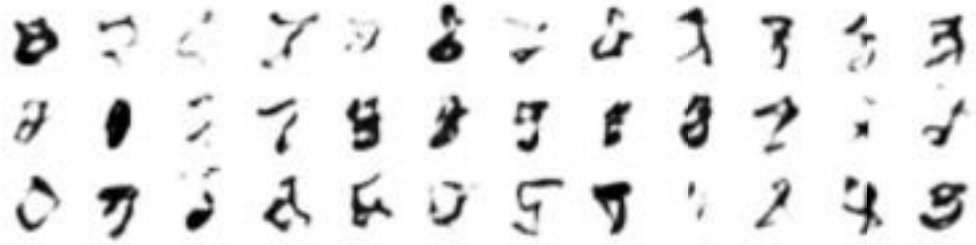
VAE

```
def train(self, x, nb_epochs, bs, lr):
  optimizer = optim.Adam(chain(self.encoder_f.parameters(),
                               self.decoder_g.parameters()),
                         lr = lr)

  for k in range(0, nb_epochs):

    for xb in x.split(bs):
      loss_kl = self.kl_f_given_x_to_g(xb).mean()

      zb = self.sample_from_f_given_x(xb)
      loss_E = - self.log_g_x_given_z(zb, xb).mean()
      loss = loss_kl + loss_E
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
```

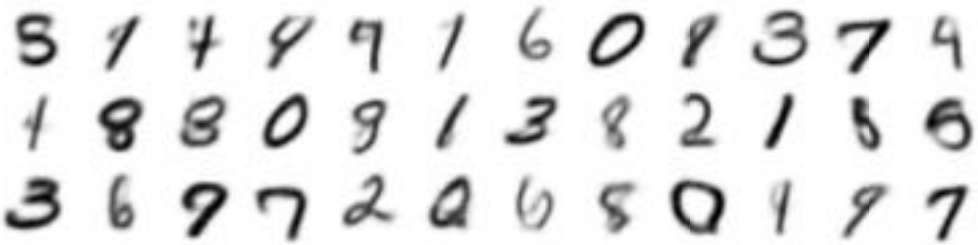
VAE

VAE

Autoencoder sampling ($d = 32$)



Variational Autoencoder sampling ($d = 32$)



VAE

Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models: well-defined probabilistic model of the generative process
- Allows inference of $q(\mathbf{z}|\mathbf{x})$, which can be useful feature representation for other tasks
- Quite easy to train in practice

VAE

Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models: well-defined probabilistic model of the generative process
- Allows inference of $q(\mathbf{z}|\mathbf{x})$, which can be useful feature representation for other tasks
- Quite easy to train in practice

Cons:

- Not asymptotically consistent unless q is perfect
- Samples are blurrier (Gaussian parameterization of the decoder output) and lower quality compared to state-of-the-art (GANs)
- Is the continuous parametrization of posterior latent distribution too restrictive?

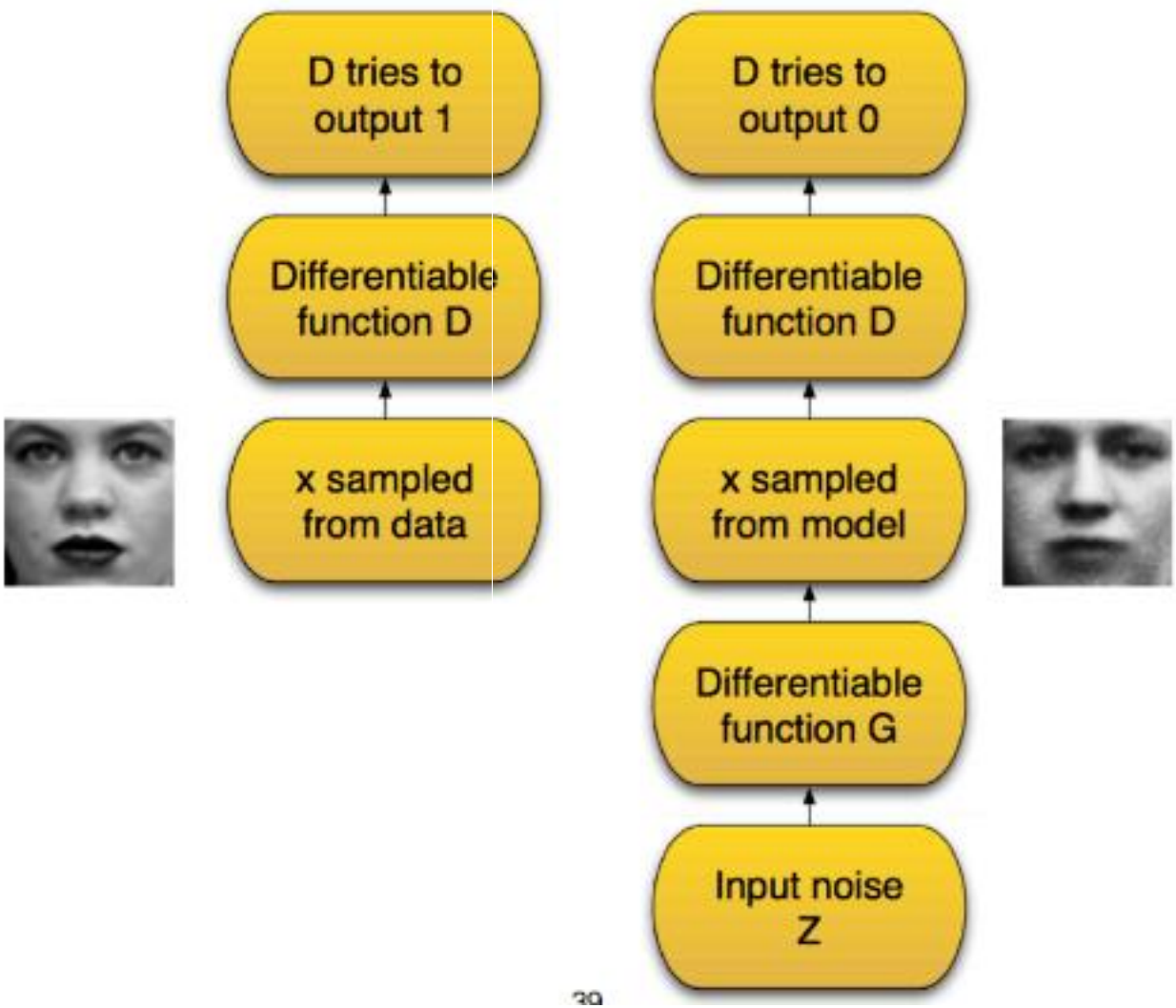
Generative Adversarial Networks (GANs)

GANs

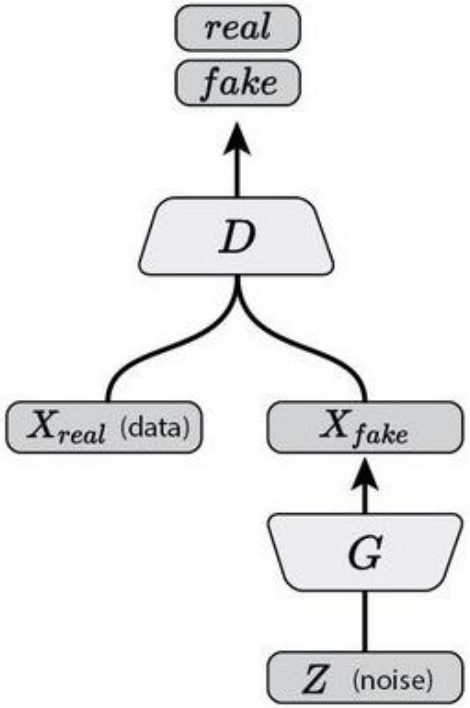
What if we give up on explicitly modeling density, and just want ability to sample?

GANs don't work with any explicit density function. Instead, they take game-theoretic approach: learn to generate from training distribution through 2-player game

GANs

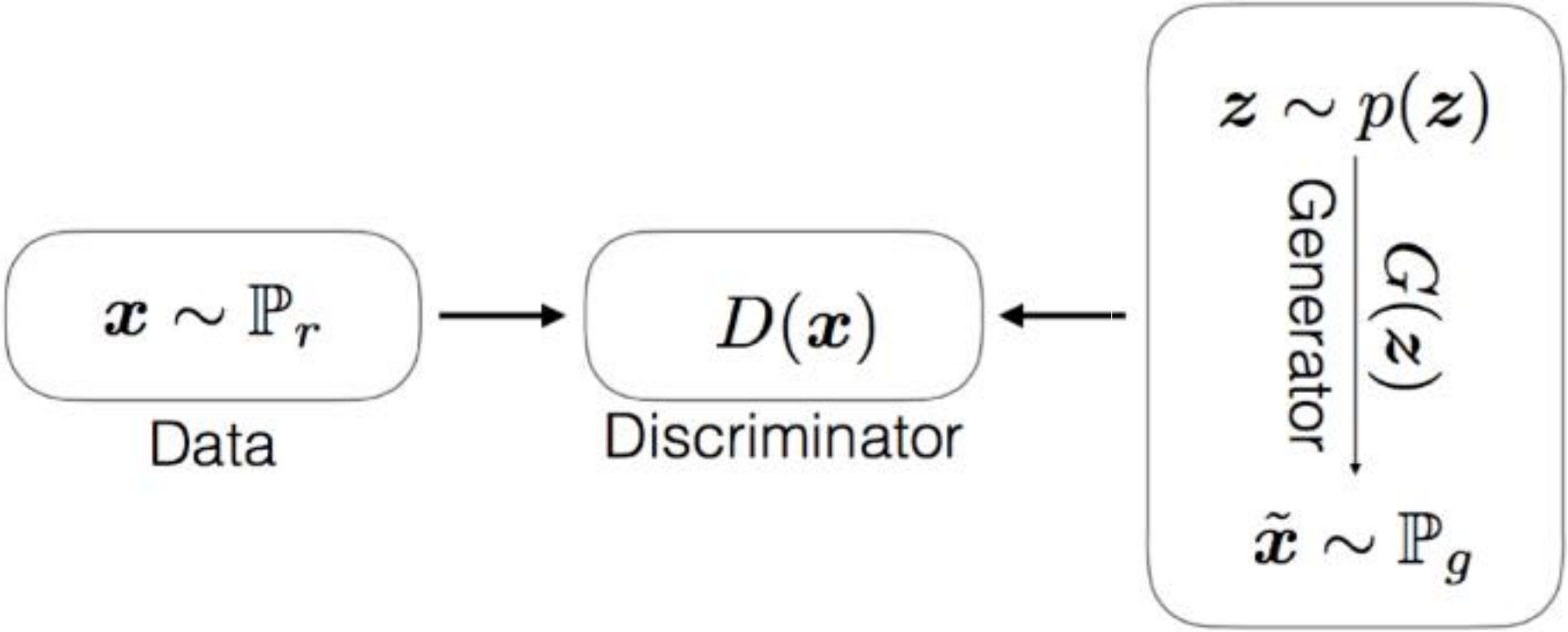

D tries to output 1
Differentiable function D
x sampled from data
D tries to output 0
Differentiable function D
x sampled from model
Differentiable function G
Input noise Z
39

GANs



Alternate training of a generative network G and a discrimininative network D

GANs


$\boldsymbol{x} \sim \mathbb{P}_r$
Data
$D(\boldsymbol{x})$
Discriminator
$\boldsymbol{z} \sim p(\boldsymbol{z})$
Generator
$G(\boldsymbol{z})$
$\tilde{\boldsymbol{x}} \sim \mathbb{P}_g$

GANs

- D tries to find out which example are generated or real
- G tries to fool D into thinking its generated examples are real

GANs

- D tries to find out which example are generated or real
- G tries to fool D into thinking its generated examples are real

Sample real data $x \sim p_{data}$

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

$$D(x) \in [0,1] \rightarrow 1$$

GANs

- D tries to find out which example are generated or real
- G tries to fool D into thinking its generated examples are real

Sample real data $x \sim p_{data}$

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$
$$D(x) \in [0,1] \rightarrow 1$$

Sample $\mathbf{z}$ and generate fake data $G(\mathbf{z})$

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$
$$D(G(z)) \in [0,1] \rightarrow 0$$

GANs

- D tries to find out which example are generated or real
- G tries to fool D into thinking its generated examples are real

Sample real data $x \sim p_{data}$

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$
$$D(x) \in [0,1] \rightarrow 1$$

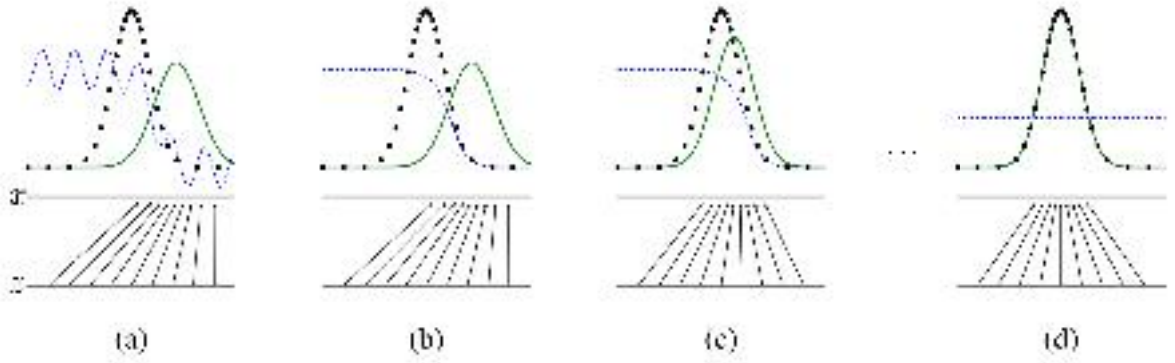
Sample $\mathbf{z}$ and generate fake data $G(\mathbf{z})$

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$
$$D(G(z)) \in [0,1] \rightarrow 0$$

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$
$$D(G(z)) \in [0,1] \rightarrow 1$$

GANs

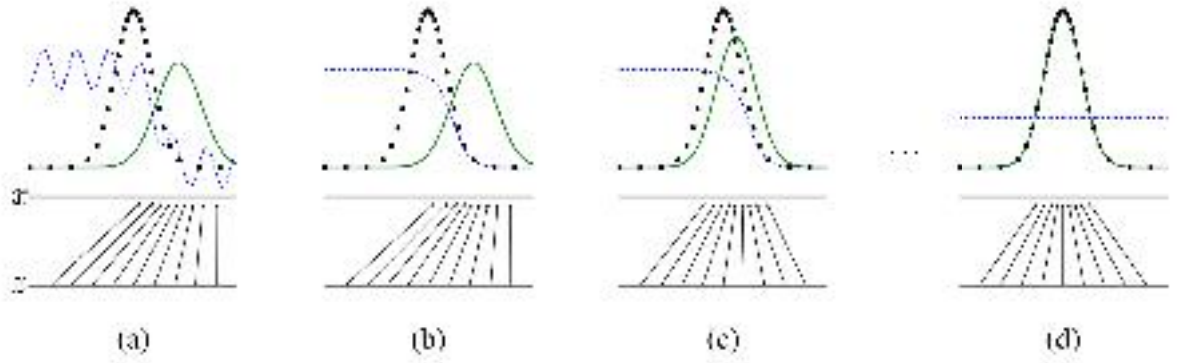
1D-example



- optimal: $D = \frac{1}{2}$, $G(\mathbf{z}) = p_{data}$

GANs

1D-example



- optimal: $D = \frac{1}{2}$, $G(\mathbf{z}) = p_{data}$
- G never "sees" training data, it is solely updated from gradients coming from D

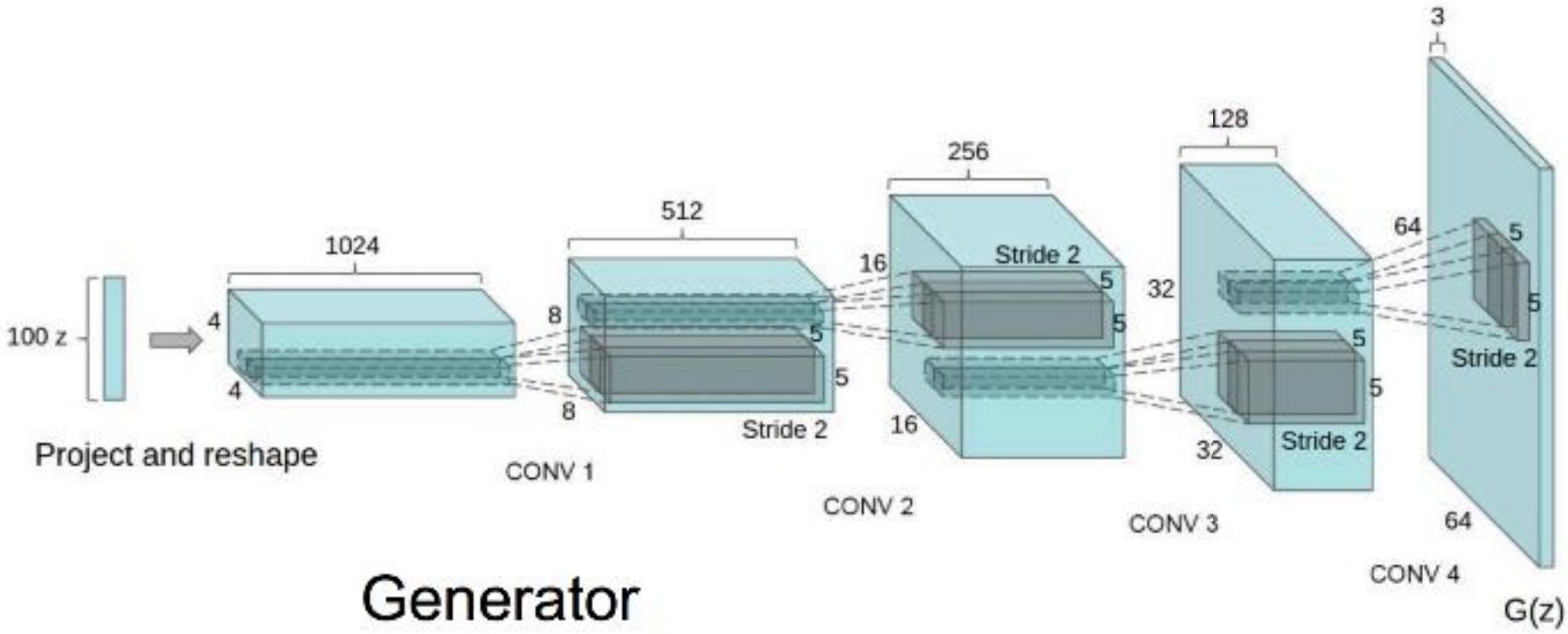
GANs

There is some theory ...

- Equilibrium is a saddle point of the discriminator loss
- Resembles Jensen-Shannon divergence
- Generator minimizes the log-probability of the discriminator being correct

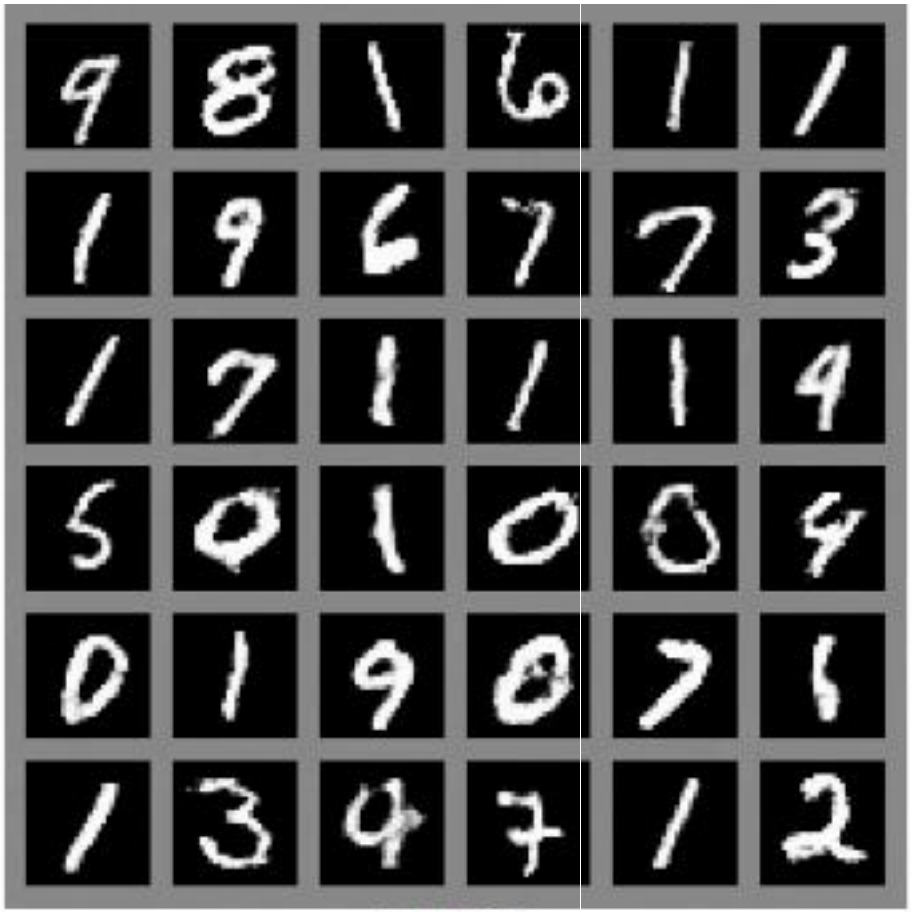
GANs

Convolutional architectures



GANs

Generated samples



MNIST

CIFAR-10

GANs

Generated samples

128x128 LSUN bedroom scenes

GANs

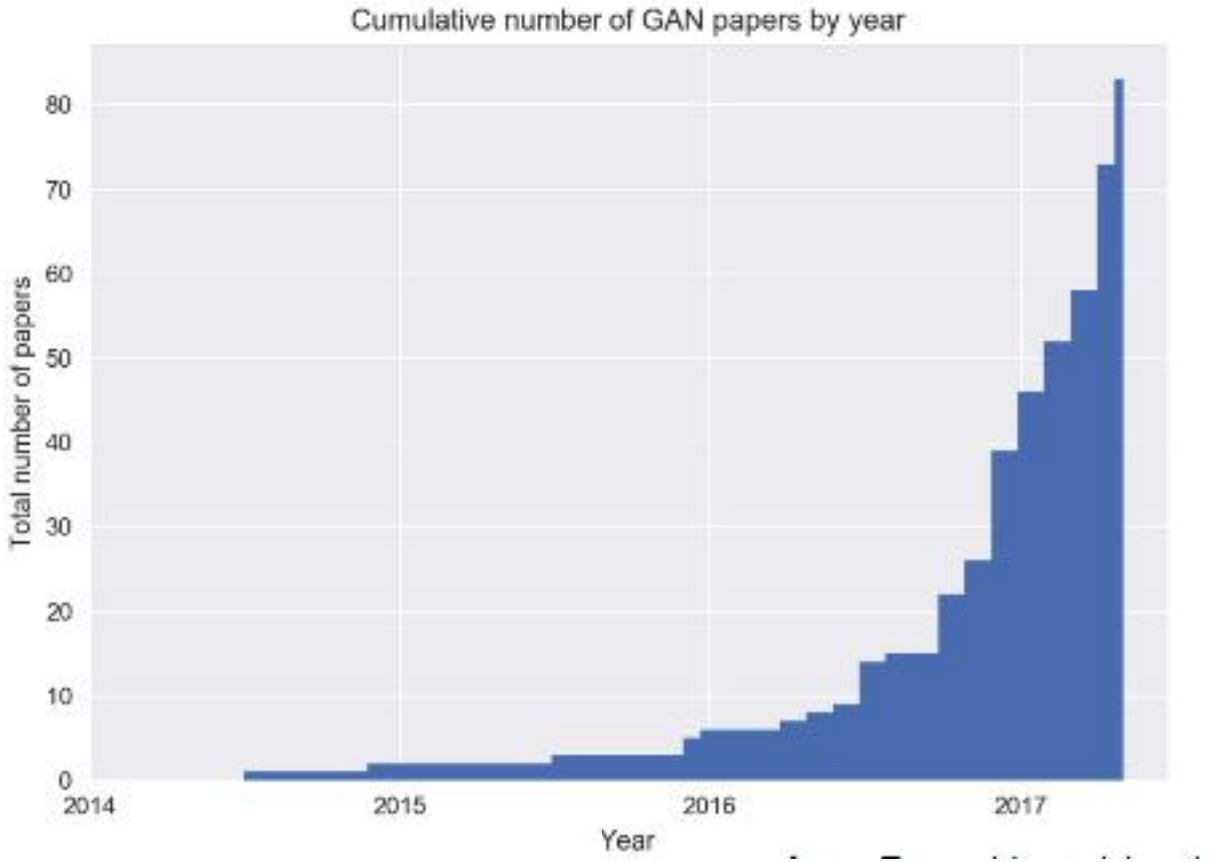
Results have become significantly impressive during 2017

GAN-Zoo

GAN — Generative Adversarial Networks
3D-GAN — Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
acGAN — Face Aging With Conditional Generative Adversarial Networks
AC-GAN — Conditional Image Synthesis With Auxiliary Classifier GANs
AdaGAN — AdaGAN: Boosting Generative Models
AEGAN — Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
AffGAN — Amortised MAP Inference for Image Super-resolution
AL-CGAN — Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
ALI — Adversarially Learned Inference
AMGAN — Generative Adversarial Nets with Labeled Data by Activation Maximization
AnoGAN — Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
ArtGAN — ArtGAN: Artwork Synthesis with Conditional Categorial GANs
b-GAN — b-GAN: Unified Framework of Generative Adversarial Networks
Bayesian GAN — Deep and Hierarchical Implicit Models
BEGAN — BEGAN: Boundary Equilibrium Generative Adversarial Networks
BGAN — Adversarial Feature Learning
BS-GAN — Boundary-Seeking Generative Adversarial Networks
CGAN — Conditional Generative Adversarial Nets
CCGAN — Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
CatGAN — Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
CoGAN — Coupled Generative Adversarial Networks
Context-RNN-GAN — Contextual RNN-GANs for Abstract Reasoning Diagram Generation
C-RNN-GAN — C-RNN-GAN: Continuous recurrent neural networks with adversarial training
CS-GAN — Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
CVAE-GAN — CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
CycleGAN — Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
DTN — Unsupervised Cross Domain Image Generation
DCGAN — Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
DiscoGAN — Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
DR-GAN — Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
DualGAN — DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
EBGAN — Energy-based Generative Adversarial Network
f-GAN — f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
GAWWN — Learning What and Where to Draw
GeoGAN — Geometric GAN: Generative Adversarial Networks with Maximum Margin Ranking
GP-GAN — GP-GAN: Towards Realistic High Resolution Image Blending
IAN — Neural Photo Editing with Introspective Adversarial Networks
IGAN — Generative Visual Manipulation on the Natural Image Manifold
IcGAN — Invertible Conditional GANs for image editing
ID-CGAN — Image De-raining Using a Conditional Generative Adversarial Network
Improved GAN — Improved Techniques for Training GANs
InfoGAN — InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
LAGAN — Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
LAPGAN — Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks
LR-GAN — LR-GAN: Layered Recursive Generative Adversarial Networks for Image Generation
LSGAN — Least Squares Generative Adversarial Networks
LS-GAN — Loss-Sensitive Generative Adversarial Networks on Lipschitz Densities
MGAN — Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks
MAGAN — MAGAN: Margin Adaptation for Generative Adversarial Networks
MAD-GAN — Multi-Agent Diverse Generative Adversarial Networks
MalGAN — Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN
MaliGAN — Maximum-Likelihood Augmented Discrete Generative Adversarial Networks
MAPTA-GAN — Deep Unsupervised Representation Learning for Remote Sensing Images
McGAN — McGan: Mean and Covariance Feature Matching GAN
MDGAN — Mode Regularized Generative Adversarial Networks
MedGAN — Generating Multi-label Discrete Electronic Health Records using Generative Adversarial Networks
MIX+GAN — Generalization and Equilibrium in Generative Adversarial Nets (GANs)
MPM-GAN — Message Passing Multi-Agent GANs
MV-BiGAN — Multi-view Generative Adversarial Networks
pix2pix — Image-to-Image Translation with Conditional Adversarial Networks
PPGN — Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space
PrGAN — 3D Shape Induction from 2D Views of Multiple Objects
RenderGAN — RenderGAN: Generating Realistic Labeled Data
RTT-GAN — Recurrent Topic-Transition GAN for Visual Paragraph Generation
SGAN — Stacked Generative Adversarial Networks
SGAN — Texture Synthesis with Spatial Generative Adversarial Networks
SAD-GAN — SAD-GAN: Synthetic Autonomous Driving using Generative Adversarial Networks
SalGAN — SalGAN: Visual Saliency Prediction with Generative Adversarial Networks
SE-GAN — SE-GAN: Speech Enhancement Generative Adversarial Network
SeGAN — SeGAN: Segmenting and Generating the Invisible
SeqGAN — SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient
SimGAN — Learning from Simulated and Unsupervised Images through Adversarial Training
SketchGAN — Adversarial Training For Sketch Retrieval
S-L GAN — Semi-Latent GAN: Learning to generate and modify facial images from attributes
Softmax-GAN — Softmax GAN
SRGAN — Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network
S²GAN — Generative Image Modeling using Style and Structure Adversarial Networks
SS-GAN — Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
StackGAN — StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks
TGAN — Temporal Generative Adversarial Nets
TAC-GAN — TAC-GAN — Text Conditioned Auxiliary Classifier Generative Adversarial Network
TP-GAN — Beyond Face Rotation: Global and Local Perception GAN for Photorealistic and Identity Preserving Frontal View Synthesis
Triple-GAN — Triple Generative Adversarial Nets
Unrolled GAN — Unrolled Generative Adversarial Networks
VGAN — Generating Videos with Scene Dynamics
VGAN — Generative Adversarial Networks as Variational Training of Energy Based Models
VAE-GAN — Autoencoding beyond pixels using a learned similarity metric
VariGAN — Multi-View Image Generation from a Single-View
ViGAN — Image Generation and Editing with Variational Info Generative AdversarialNetworks
WGAN — Wasserstein GAN
WGAN-GP — Improved Training of Wasserstein GANs
WaterGAN — WaterGAN: Unsupervised Generative Network to Enable Real-time Color Correction of Monocular Underwater Images

GANs

Results have become significantly impressive during 2017

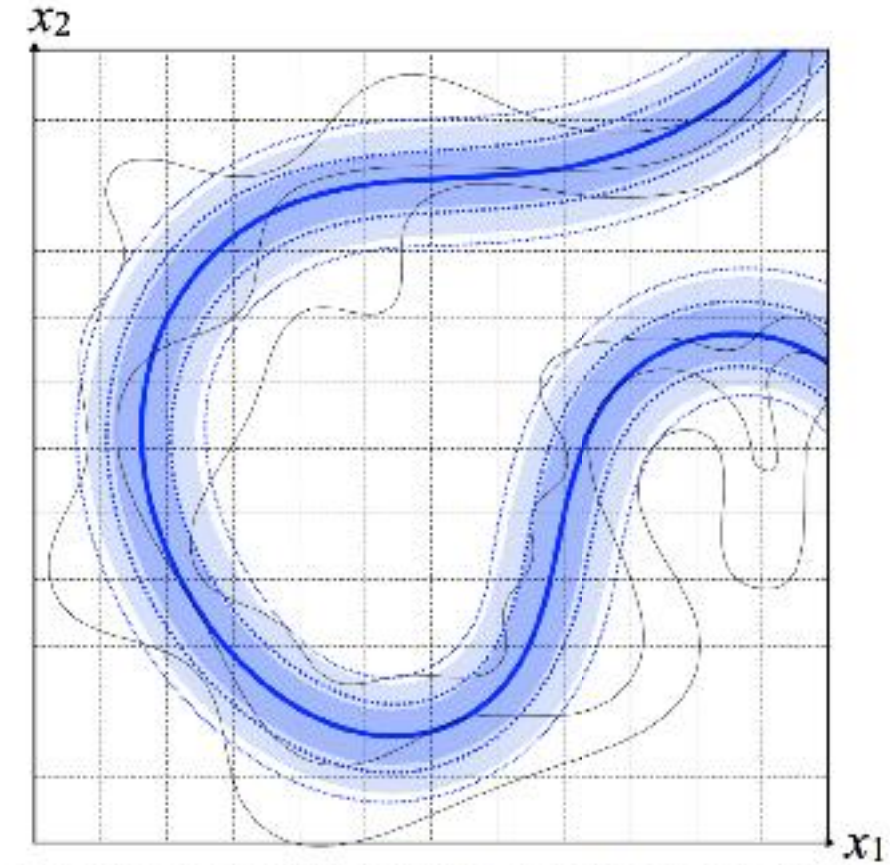


GANs

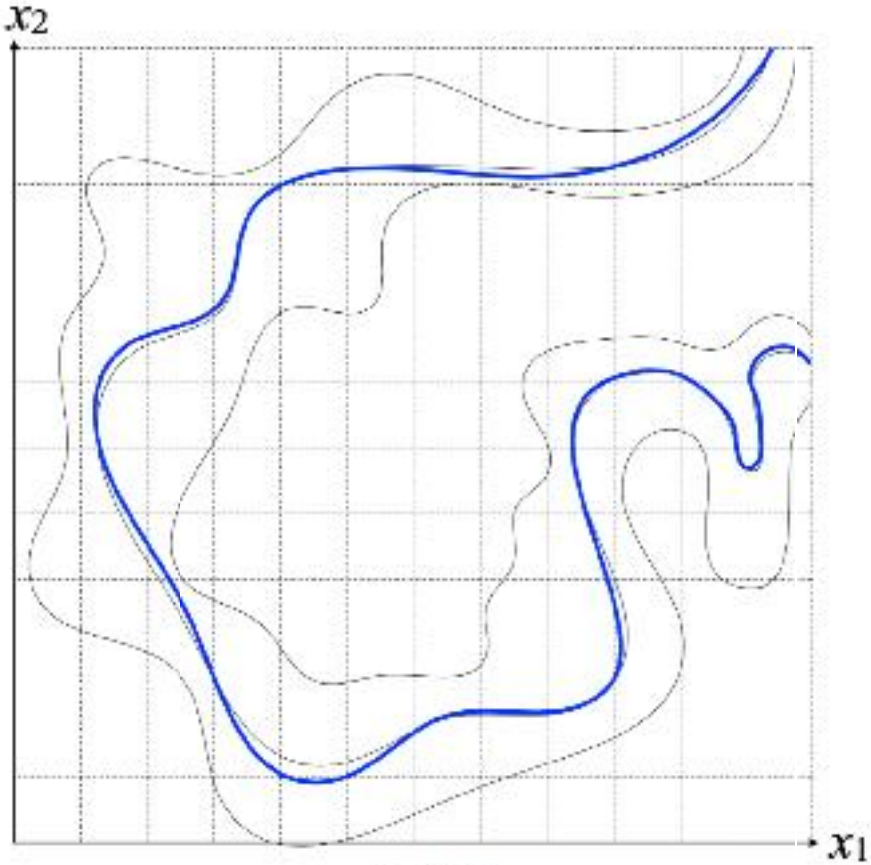
- Although is still more of less a beauty contest, significant progress has been made

GANs

What makes GANs so special?



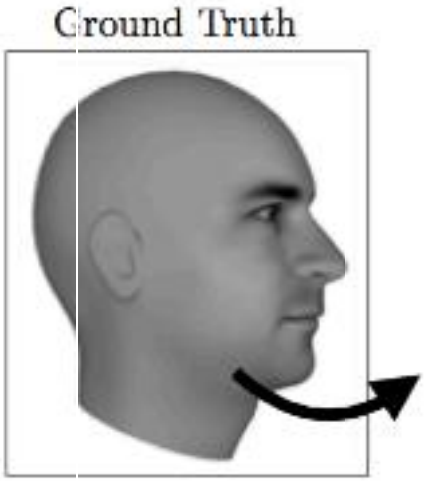
more traditional max-likelihood approach



GAN

GANs in action

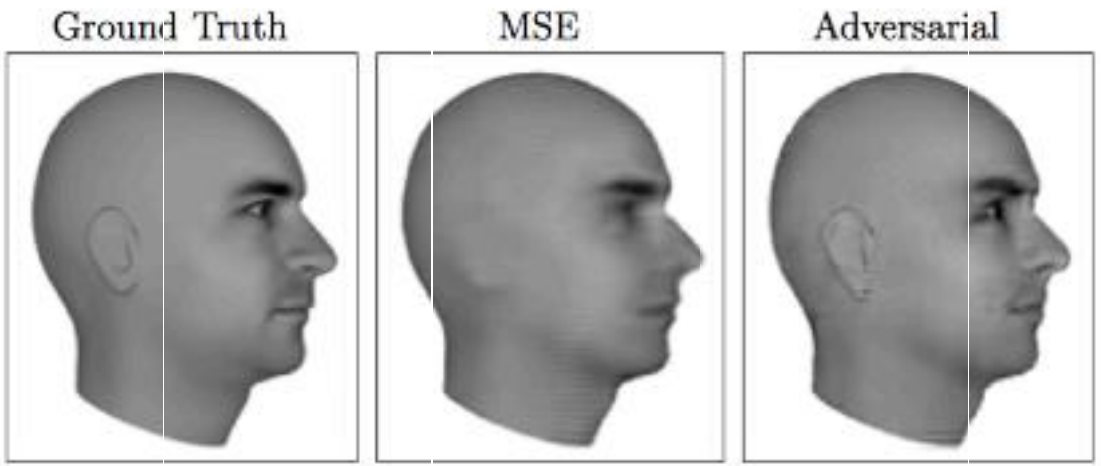
Next video frame prediction



Lotter et al., 2016

GANs in action

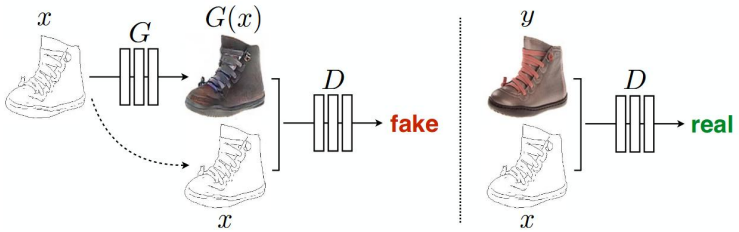
Next video frame prediction



Lotter et al., 2016

GANs in action

Image-to-image translation: conditional GANs



The generation no longer makes use of **z**, rather is conditionned by an input **x**

Isola et al., Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

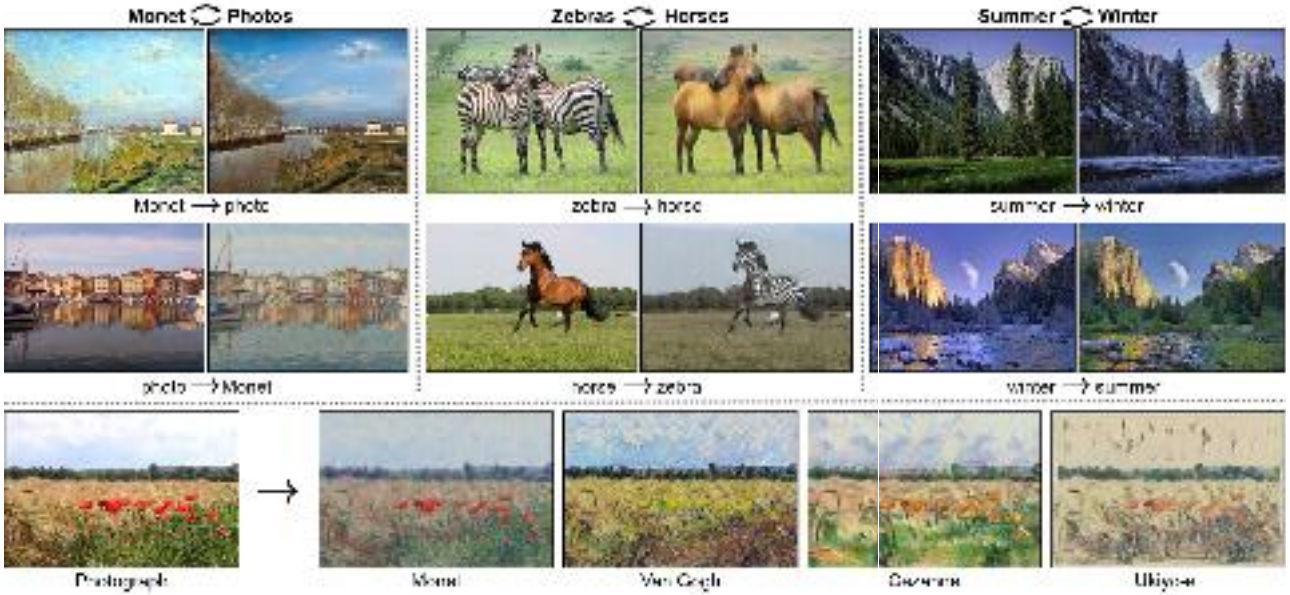
GANs in action

Unsupervised image-to-image translation

Liu et al., 2017

GANs in action

[CycleGAN]

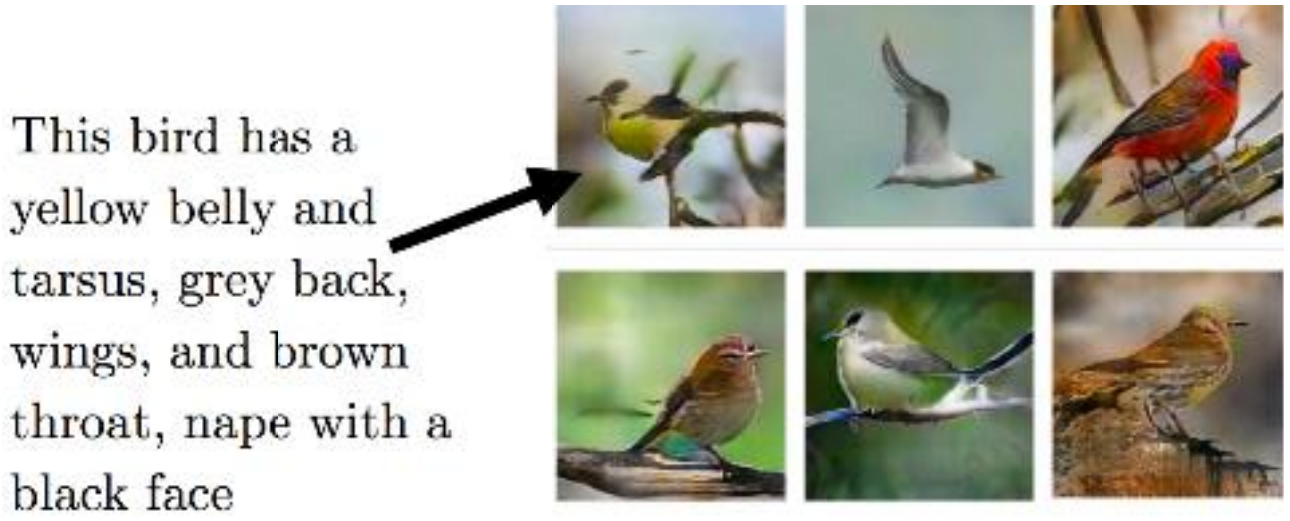


- No alignment between pairs needed, simply two different sets of images

Zhu et al., Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV 2017

GANs in action

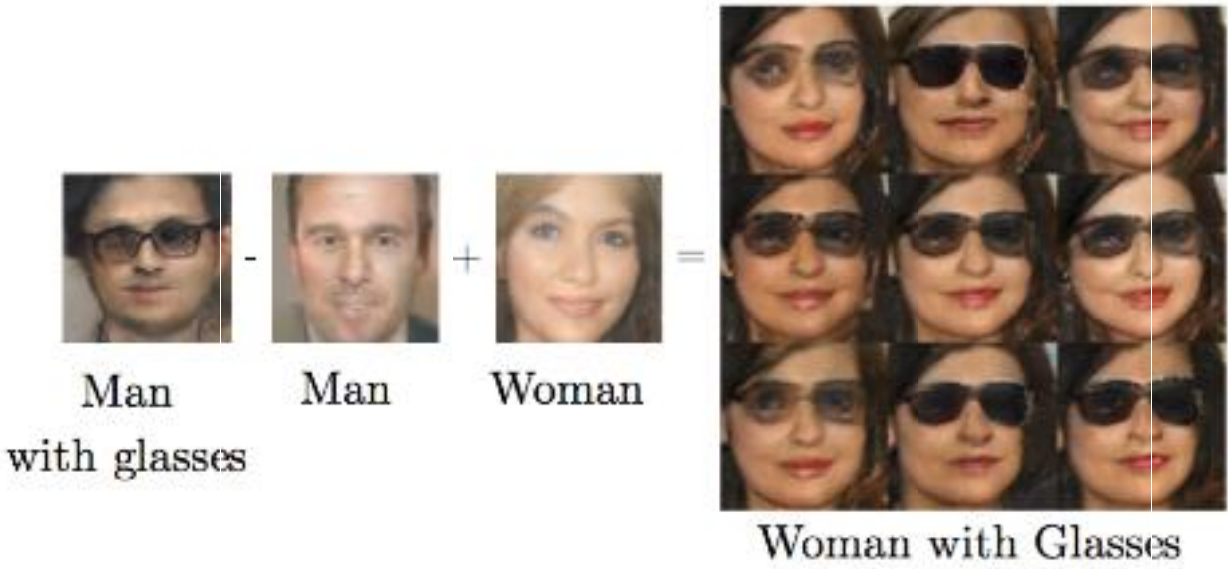
Text-to-image synthesis



Zhang et al., 2016

GANs in action

Face arithmetics



- latent space has some local linea properties (vector arithmetic is like with word2vec)

Radford et al., Unsupervised representation learning with deep convolutional generative adversarial networks. 2015.

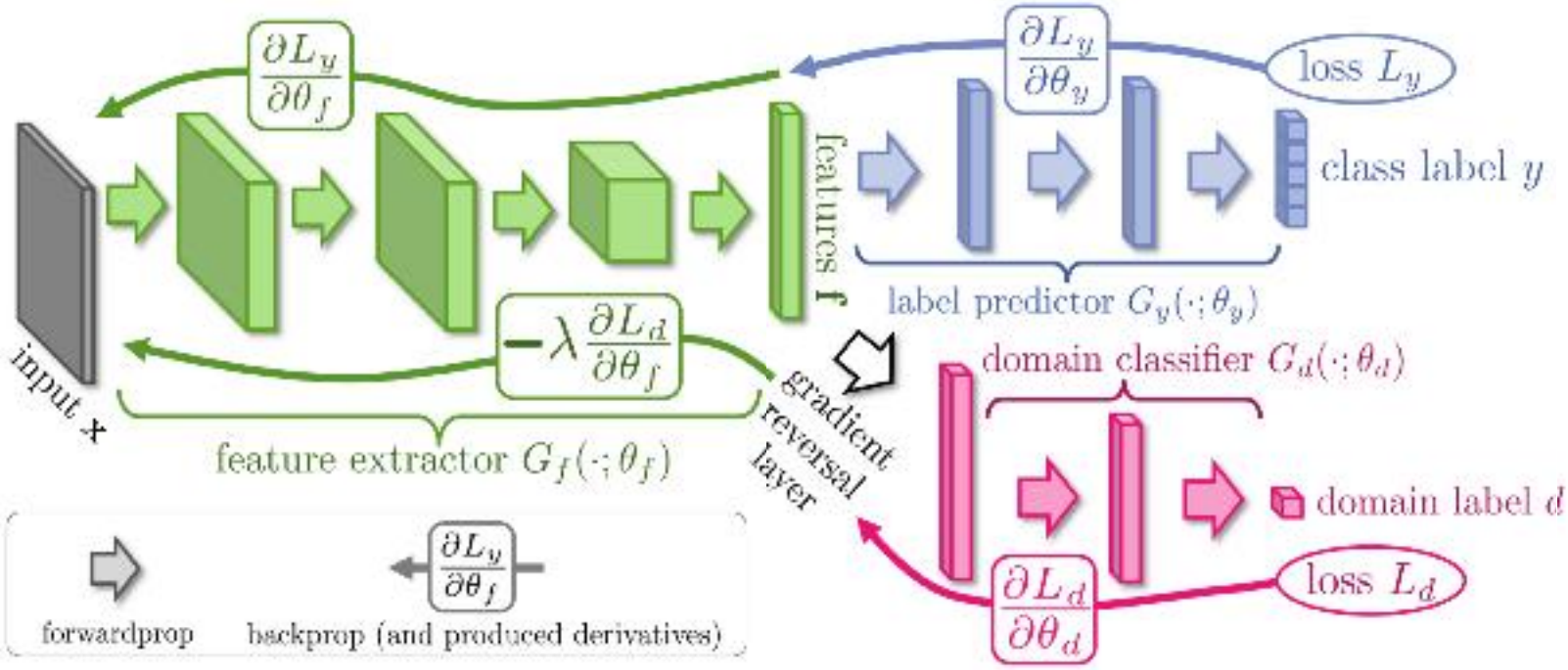
GANs in action

Progressive growing of GANs

All images are generated by walking through the latent space

Karras, Tero, et al. Progressive growing of gans for improved quality, stability, and variation. 2017.

Adversarial training



- Forces the features (green) not to be specialised in discriminating between domains

Ganin et al., Domain-adversarial training of neural networks, JMLR 2016

GANs

- Don't work with an explicit density function
- Take game-theoretic approach: learn to generate from training distribution through 2-player game

Pros:

- beautiful and high-quality samples

Cons:

- unstable training
- can't (directly) solve inference queries $p(z|x)$

Takeaways

(Reconstruction) Autoencoders

- have no direct probabilistic interpretation;
- are not designed to generate useful samples;
- encoder defines a useful latent representation.

Takeaways

(Reconstruction) Autoencoders

- have no direct probabilistic interpretation;
- are not designed to generate useful samples;
- encoder defines a useful latent representation.

VAEs

- model explicitly (a lower bound of) the likelihood;
- high quality samples from high dimensional distributions;
- encoder defines a useful latent representation;
- optimization problem is often well-behaved.

Takeaways

GANs

- likelihood-free generative models;
- high quality samples from high dimensional distributions;
- discriminator not meant be used as encoder;
- optimization problem is trickier than for VAEs (open research).

Takeaways

GANs

- likelihood-free generative models;
- high quality samples from high dimensional distributions;
- discriminator not meant be used as encoder;
- optimization problem is trickier than for VAEs (open research).

There exists other kinds of generative models, e.g. auto-regressive models:

- examples: PixelCNN, WaveNet, RNN language models...
- can be used as prior and decoder for VAEs, generators for GANs.

Takeaways

Adversarial training is useful beyond generative models:

- domain adaptation;
- learning representations blind to sensitive attributes;
- defend against malicious inputs (adversarial examples);
- regularization by training on adversarial examples.

Takeaways

Adversarial training is useful beyond generative models:

- domain adaptation;
- learning representations blind to sensitive attributes;
- defend against malicious inputs (adversarial examples);
- regularization by training on adversarial examples.

Quality of samples from VAE and GAN depends a lot on the architectures of sub-networks.

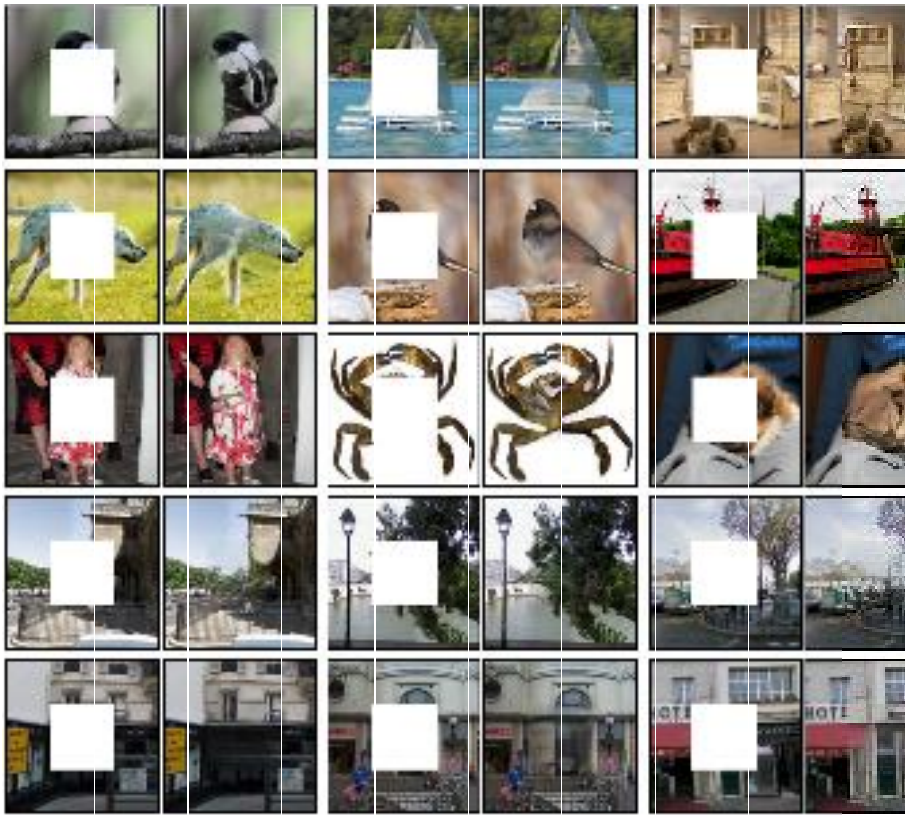
Self-supervised learning

- supervised loss
- no labeling needed

Self-supervised learning

Context Encoders

- remove patch from image
- train autoencoder to "hallucinate" missing content using context information
- ground truth easily available from the removed patch
- leveraging an adversarial loss for diversity and sharpness
- learned features are effective for CNN pre-training on classification, detection, and segmentation tasks

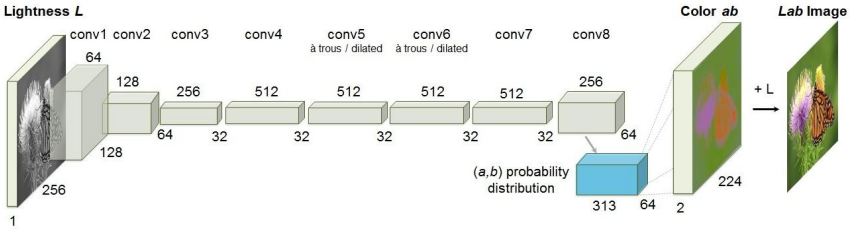


D. Pathak et al., Context Encoders: Feature Learning by Inpainting, CVPR 2016

Self-supervised learning

Image colorization

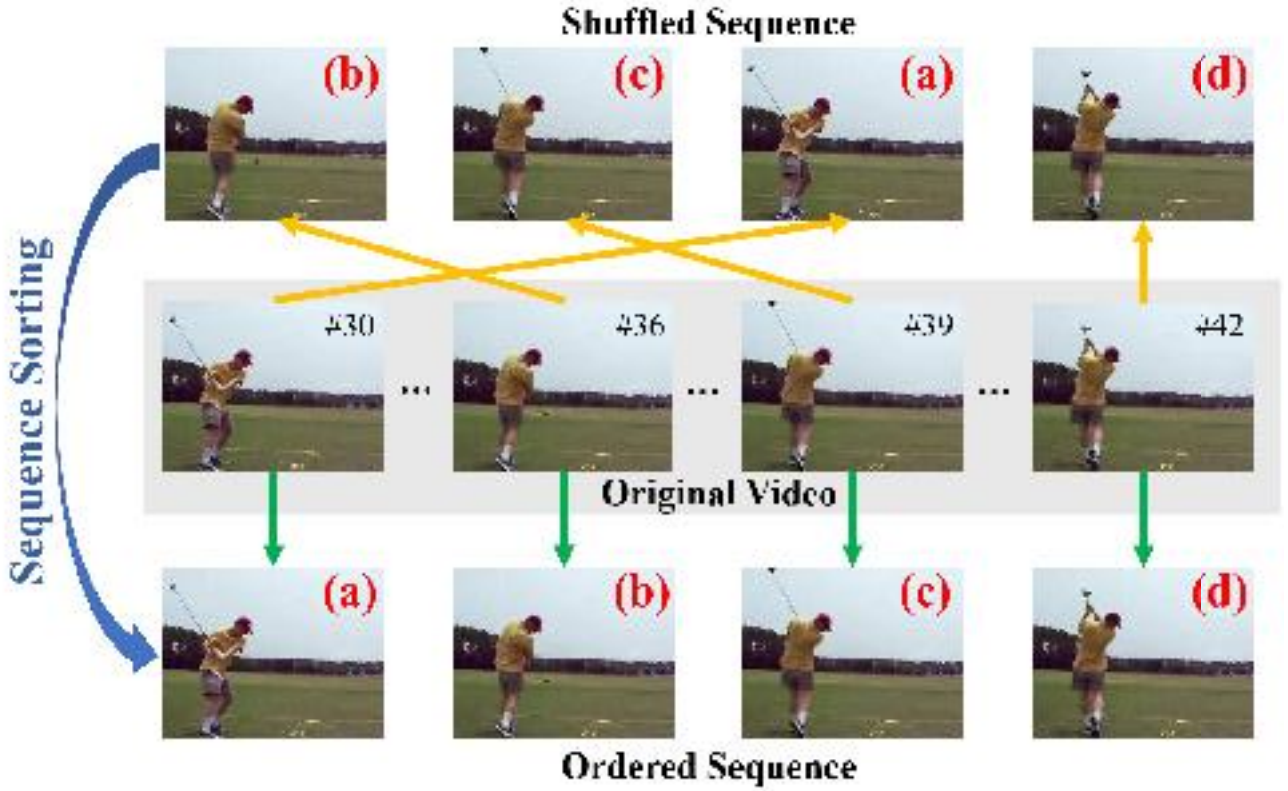
- given grayscale photograph, train model to hallucinate plausible color version of the photo
- for training convert color images to grayscale and use color variant as training objective



R. Zhang et al., Colorful Image Colorization, ECCV 2016

Self-supervised learning

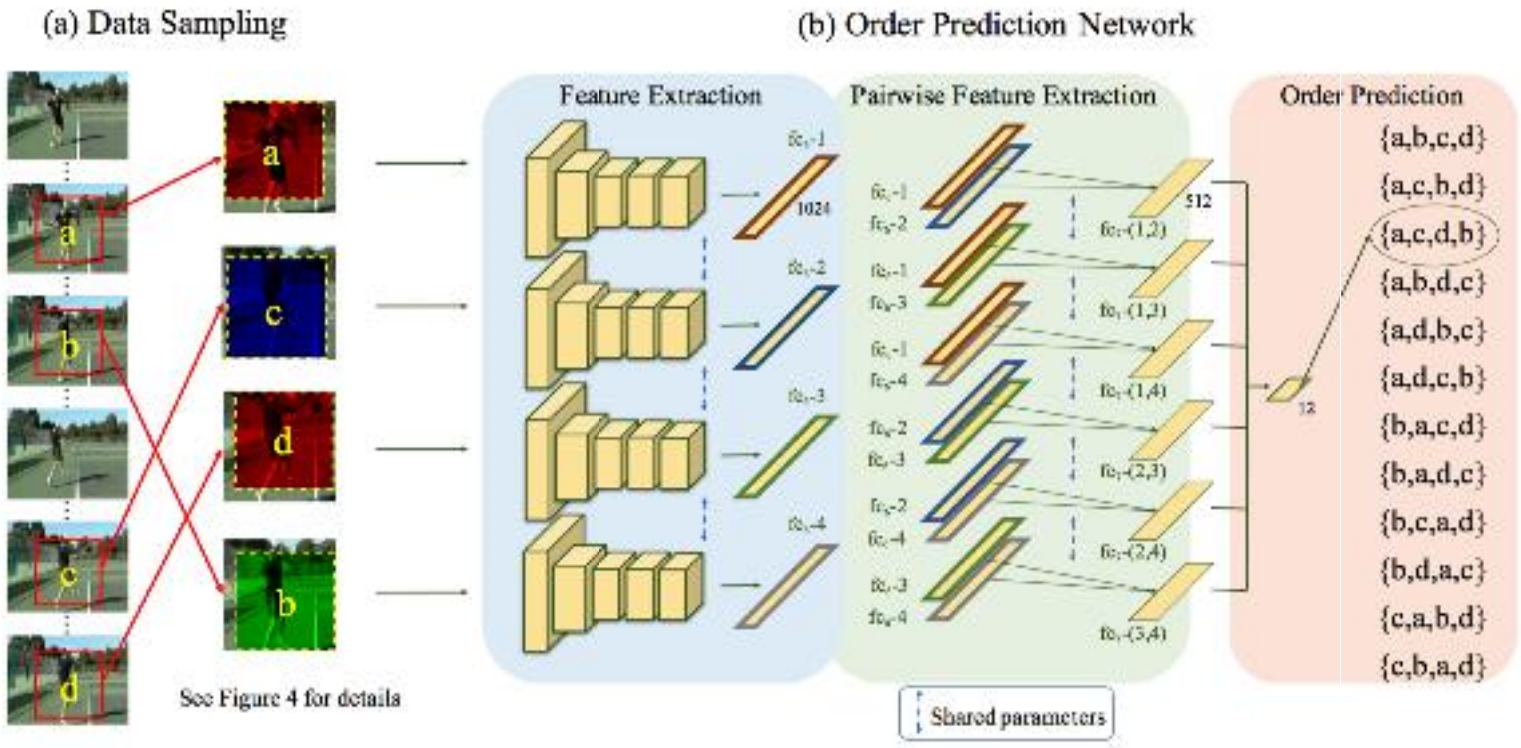
Sorting video sequences



H.Y. Lee et al., Unsupervised Representation Learning by Sorting Sequences, ICCV 2017

Self-supervised learning

Sorting video sequences



H.Y. Lee et al., Unsupervised Representation Learning by Sorting Sequences, ICCV 2017

Self-supervised learning

Connecting visual appearance of objects with their sounds

- take video and separate its visual content from sound
- train a network to decide whether a given image and sound come from the same video
- after training, the obtained features get close to SoTA results

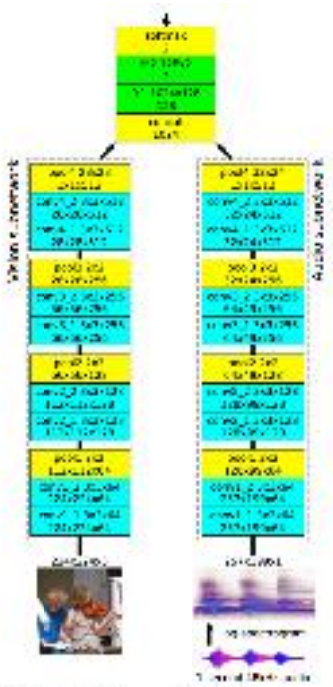
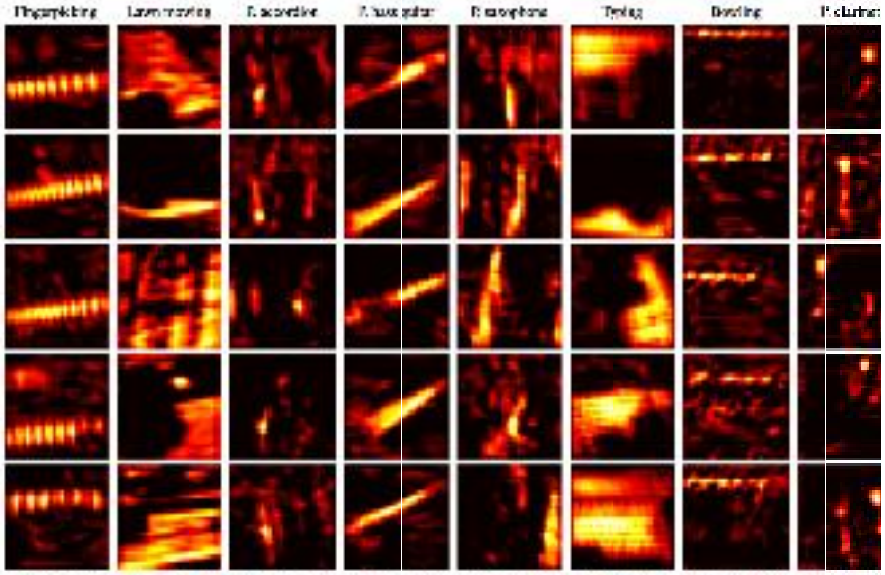


Figure 2. L^3-Net architecture. Each blocks represents a single layer with text providing more information – first row: layer name and parameters, second row: output feature map size. Layers with a name prefix conv, pool, fc, concat, softmax are convolutional, max-pooling, fully connected, concatenation and softmax layers, respectively. The listed parameters are: conv – kernel size and number of channels, pooling – kernel size, fc – size of the weight matrix. The stride of pool layers is equal to the kernel size and there is no padding. Each convolutional layer is followed by batch normalization [2] and a ReLU nonlinearity, and the first fully connected layer (fc1) is followed by ReLU.

Arandjelovic & Zisserman, Look, Listen and Learn, ICCV 2017

Self-supervised learning

Connecting visual appearance of objects with their sounds



Arandjelovic & Zisserman, Look, Listen and Learn, ICCV 2017

Open topics

- Domain adaptation from synthetic to real
- Adversarial robustness
- Few shot learning
- Modelisation of uncertainty

What we did not cover

Lots of things

- Recurrent neural networks
- Reinforcement Learning
- Uncertainty estimation
- Many computer vision tasks and architectures

References

- EE-559 Deep learning (Francois Fleuret, EPFL)
- CS-231n Convolutional Neural Networks for Visual Recognition (Fei-Fei Li et al, Stanford)
- Deep Learning book (Ian Goodfellow, Yoshua Bengio, Aaron Courville)
- Neural Networks and Deep Learning (Michael Nielsen)
- M2DS Deep Learning (Charles Ollion and Olivier Grisel)
- SIF Deep Learning for Vision (Yannis Avrithis, Inria)
- INFO8004-1 Advanced Machine Learning (Pierre Geurts, Gilles Louppe, Louis Wehenkel, Université de Liège)

Thank you!