Addendum: Towards Diffusion Models and VAE

Volker Tresp
Winter 2023-2024

Towards Diffusion Models

Where is the Low-dimensional Space?

- In the VAE and GAN, we sampled randomly from a low-dimensional latent space (M_{hidd}-dimensional) and then mapped the sample to an image on the manifold (M-dimensional)
- In the diffusion model, we map from high-dimensional (M-dimensional) noise to a high-dimensional manifold (M-dimensional)
- How can this work? Where is the M_{hidd}-dimensional space?

Denoising Autoencoder

- Let's assume a training image $\mathbf{x}_i$
- The denoising autoencoder (DAE) was trained for $\mathbf{x}_i^k = \mathbf{x}_i + \epsilon_i^k$; $k = 1, \ldots$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and $\sigma^2 << 1$ small; then we obtained

$$\hat{\mathbf{x}}_i = \mathsf{DAE}(\mathbf{x}_i^k)$$

- Cost function

$$\sum_i \sum_k \|\mathbf{x}_i - \mathsf{DAE}(\mathbf{x}_i^k)\|^2$$

Update

- For a test image. $\mathbf{x}$, the noise is predicted to be

$$\epsilon \approx \mathbf{x} - \mathsf{DAE}(\mathbf{x})$$

- Interesting: Whereas $\mathsf{DAE}(\mathbf{x})$ predicts a noise free image on the manifold, we also have obtained a noise predictor!

- Let's just take a small step

$$\mathbf{x}' = \mathbf{x} - \eta(\mathbf{x} - \mathsf{DAE}(\mathbf{x})) = \mathbf{x}(1-\eta) + \eta\mathsf{DAE}(\mathbf{x})$$

- Where is the M_{hidd}-dimensional space? It is within the DAE (resp. U-net)!

- This will restore the closest image

Attractor on the Manifold?

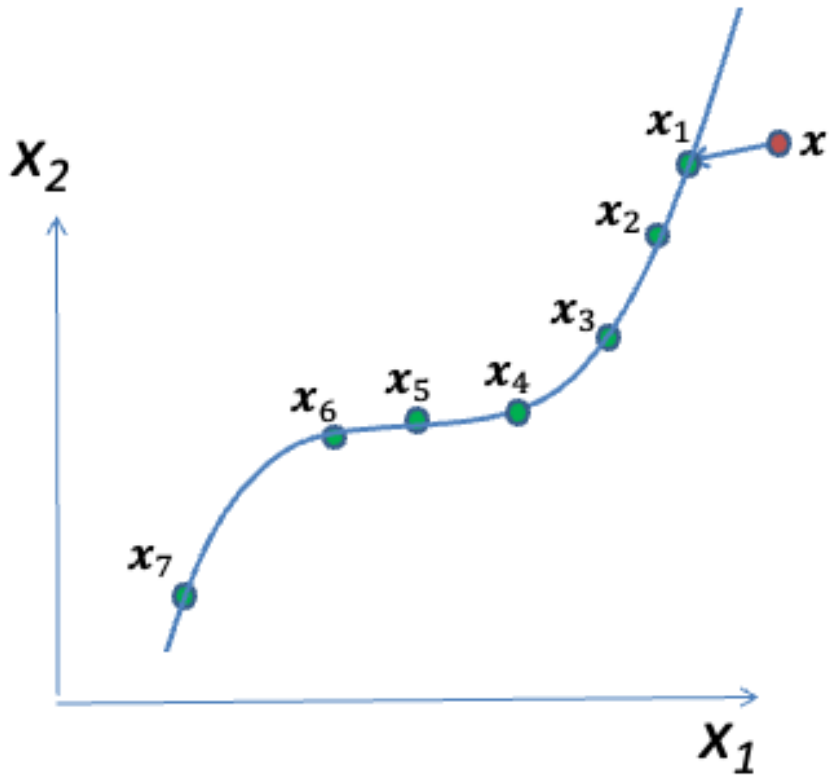
- If $\sigma^2 << 1$ $\mathbf{x}$ is close to a training image $\mathbf{x}_i$, and $\mathbf{x}_i$ is an attractor (for all $\mathbf{x}$ in the neighbourhood of x_i, x_i is the target for the DAE)
- For any σ^2,

$$\text{DAE}(\mathbf{x}) = \sum_i P(\mathbf{x}_i|\mathbf{x})\mathbf{x}_i$$

  With $P(\mathbf{x}|\mathbf{x}_i) \sim \mathcal{N}(\mathbf{x}; \mathbf{x}_i, \sigma^2)$ we have

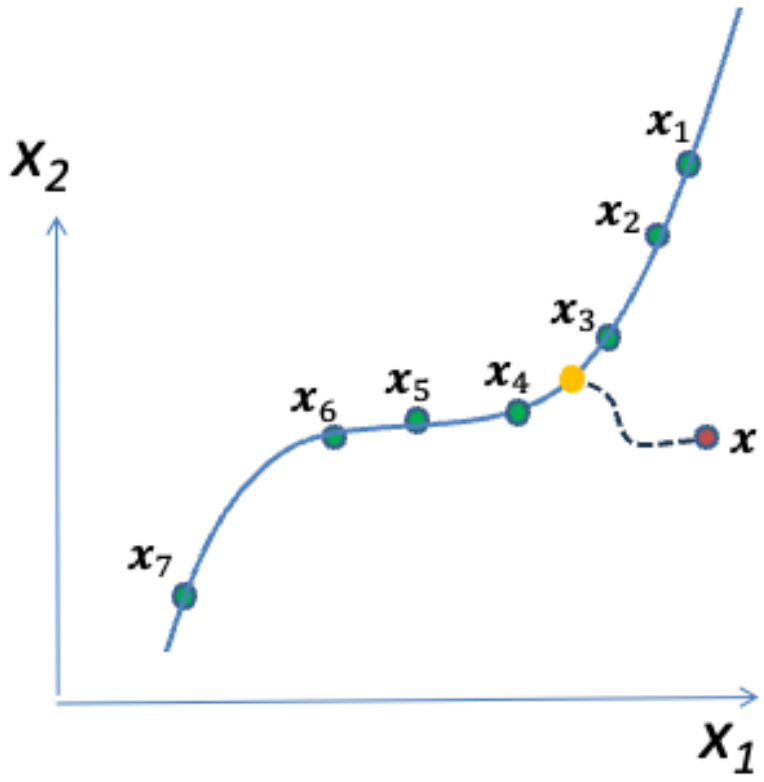
$$P(\mathbf{x}_i|\mathbf{x}) = \frac{\mathcal{N}(\mathbf{x}; \mathbf{x}_i, \sigma^2)}{\sum_i \mathcal{N}(\mathbf{x}; \mathbf{x}_i, \sigma^2)}$$

- If σ^2 is small, $\mathbf{x} \approx \mathbf{x}_i$, there is a clear winner, and $\mathbf{x}$ will be attracted by $\mathbf{x}_i$: Most images in the neighbourhood of $\mathbf{x}_i$ will have been generated by $\mathbf{x}_i$
- For large σ^2, all $\mathbf{x}_i$ are equally likely to be source for $\mathbf{x}$ and the update moves $\mathbf{x}$ towards the mean training image
- Eventually it will be attrackted by the manifold



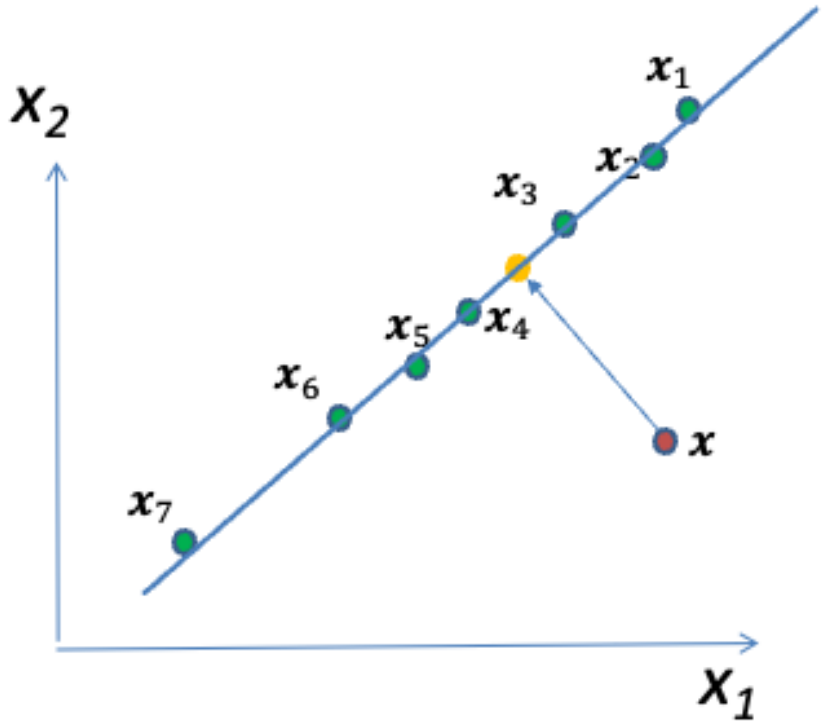
DAE

- The DNA restores the image closest to x
- Here: x_1



„Diffusion“

- “Diffusion” first generates an image on the manifold (yellow dot)



„SVD – Diffusion“

- If the images are on a linear subspace, DAE($\boldsymbol{x}$)= $VV^T \boldsymbol{x}$ is on the manifold and is not identical to a stored image
- Stored images are not attractors
- To get attractor properties, DAE($\boldsymbol{x}$) needs to be nonlinear

Variational Autoencoder

Learning a Generator

- If I knew for each data point $\mathbf{h}$ and $\mathbf{x}$, I could learn a generator $g^{gen}(\cdot)$ simply by supervised training
- Unfortunately, $\mathbf{h}$ is unknown
- How about I assume that $\mathbf{h}$ comes from a Gaussian distribution with unit variance $P(\mathbf{h}) = \mathcal{N}(\mathbf{h}; 0, I)$ (each dimension is independently Gaussian distributed, with unit variance)
- Goal: **any** sample $\mathbf{h}$ from this Gaussian distribution should generate a valid **x** (this was not enforced in the normal autoencoder)
- This is the idea behind the Variational Autoencoder (VAE)

Generating Data from the VAE

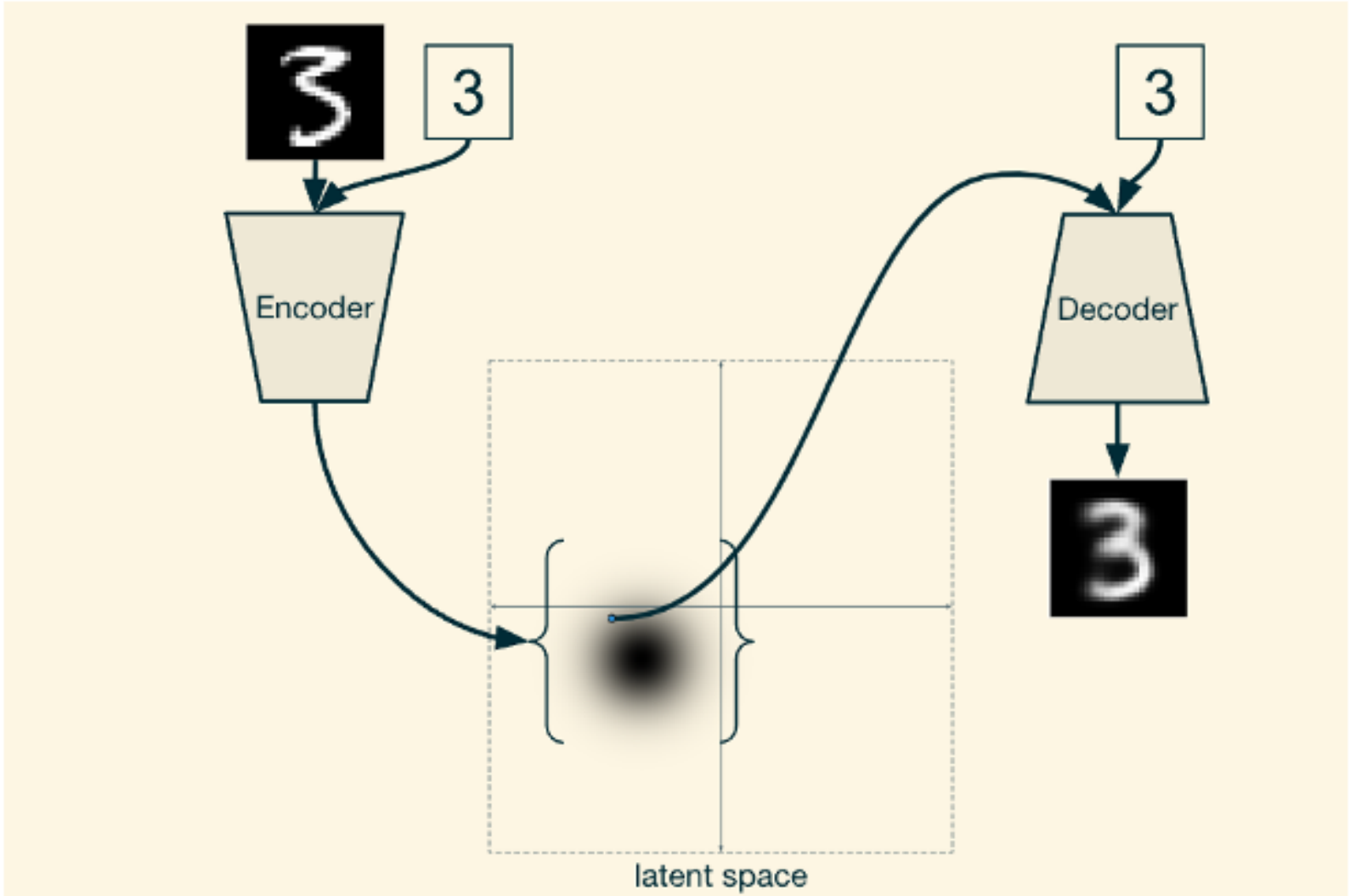
- After training, we generate a new $\mathbf{x}$ by first generating a sample $\mathbf{h}_s$ from $\mathcal{N}(0, I)$; then

$$\mathbf{x} = g^{gen}(\mathbf{h}_s, \mathbf{v})$$

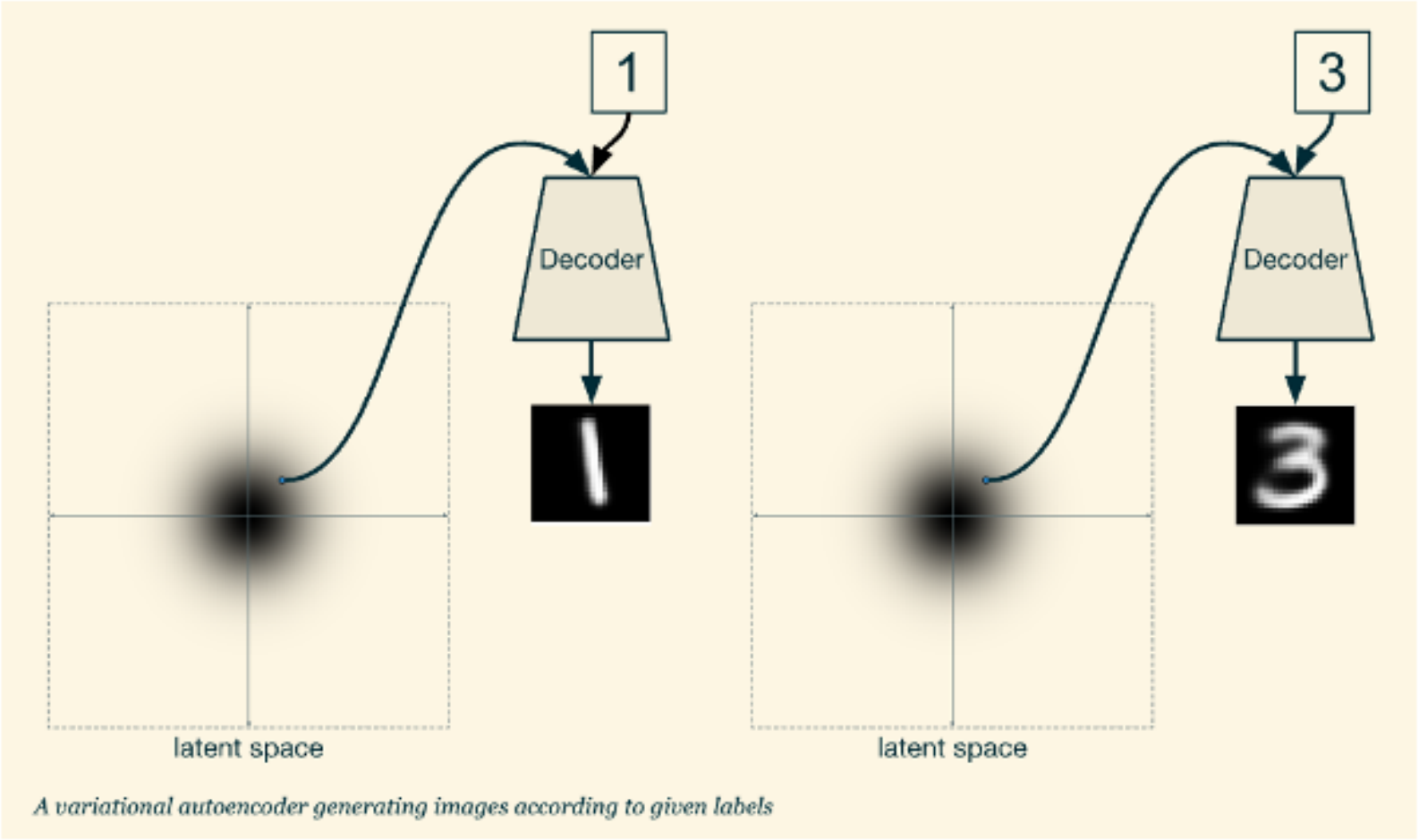
  Here, $\mathbf{v}$ are parameters of the generator $g^{gen}(\cdot)$

- Thus generating a new $\mathbf{h}$ is trivial, but how do I learn the $g^{gen}(\cdot)$?

- This is a bit involved and is described in the Appendix



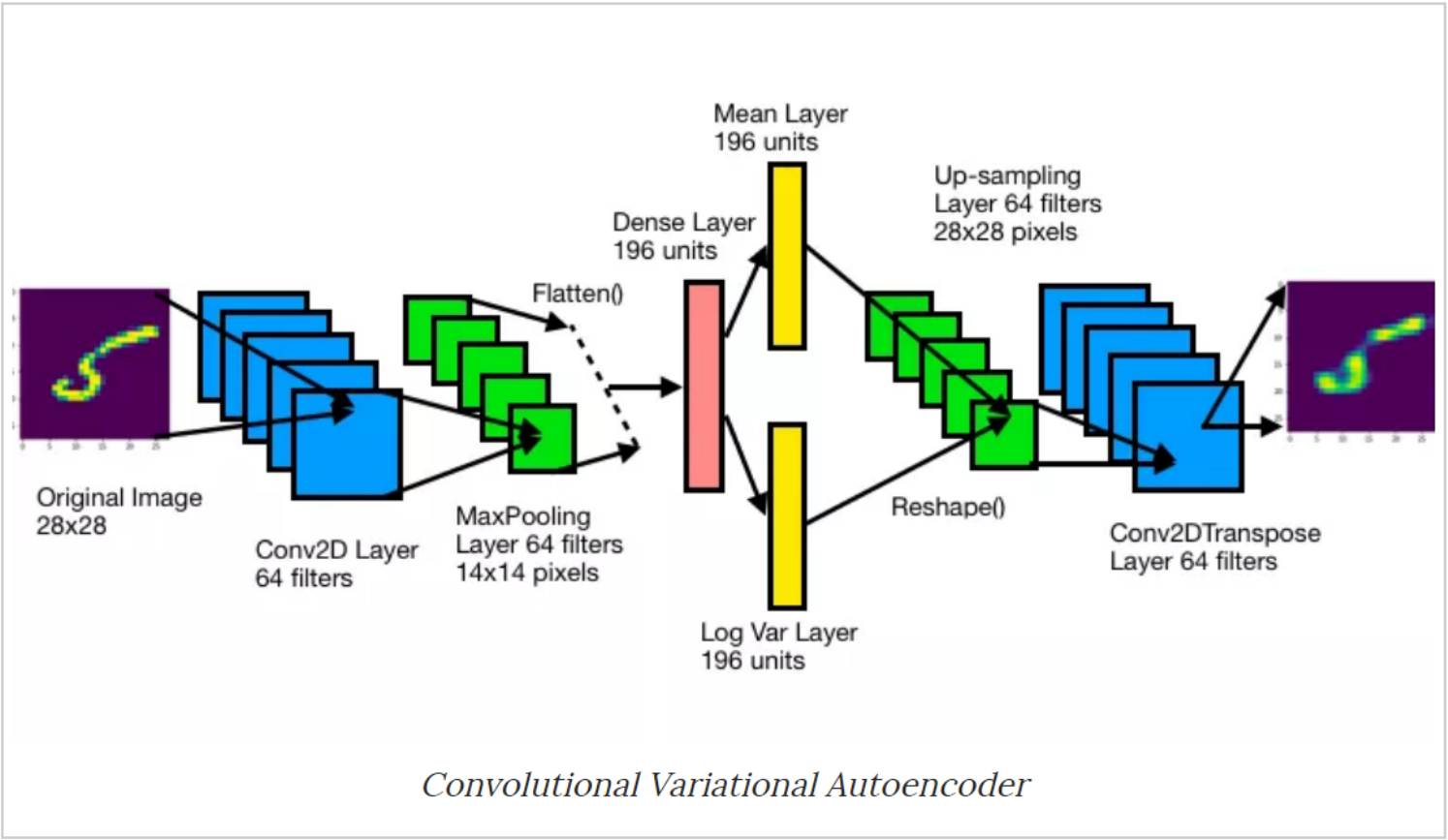
A conditional variational autoencoder



A variational autoencoder generating images according to given labels

Convolutional Variational Autoencoders

- The convolutional variational autoencoder uses convolutional layers



Convolutional Variational Autoencoder

Training the VAE

Training the VAE

- Training with a maximum likelihood approach is infeasible; one uses a mean field approximation, a special case of a variational approximation; details can be found in the Appendix
- For each data point $\mathbf{x}$, one learns the variational parameters, mean $\mu_{\mathbf{u}}(\mathbf{x})$ and covariance matrix $\Sigma_{\mathbf{u}}(\mathbf{x})$ as a function of $\mathbf{x}$ (e.g., with a deep neural network with weights $\mathbf{u}$); this is the **encoder**; when $\Sigma_{\mathbf{u}}(\mathbf{x})$ is diagonal, the encoder has $2M_{hidd}$ outputs
- A sample $\mathbf{h}_s$ is generated from $\mathcal{N}(\mu_{\mathbf{u}}(\mathbf{x}), \Sigma_{\mathbf{u}}(\mathbf{x}))$ (which is an approximation to $P(\mathbf{h}|\mathbf{x})$); thus instead of transmitting the prediction of the encoder $\mu_{\mathbf{u}}(\mathbf{x})$ to the decoder, we transmit a noisy version of $\mu_{\mathbf{u}}(\mathbf{x})$
- Based on this sample all parameters ($\mathbf{v}$, $\mathbf{u}$) are adapted with backpropagation
- Then one proceeds with the next data point

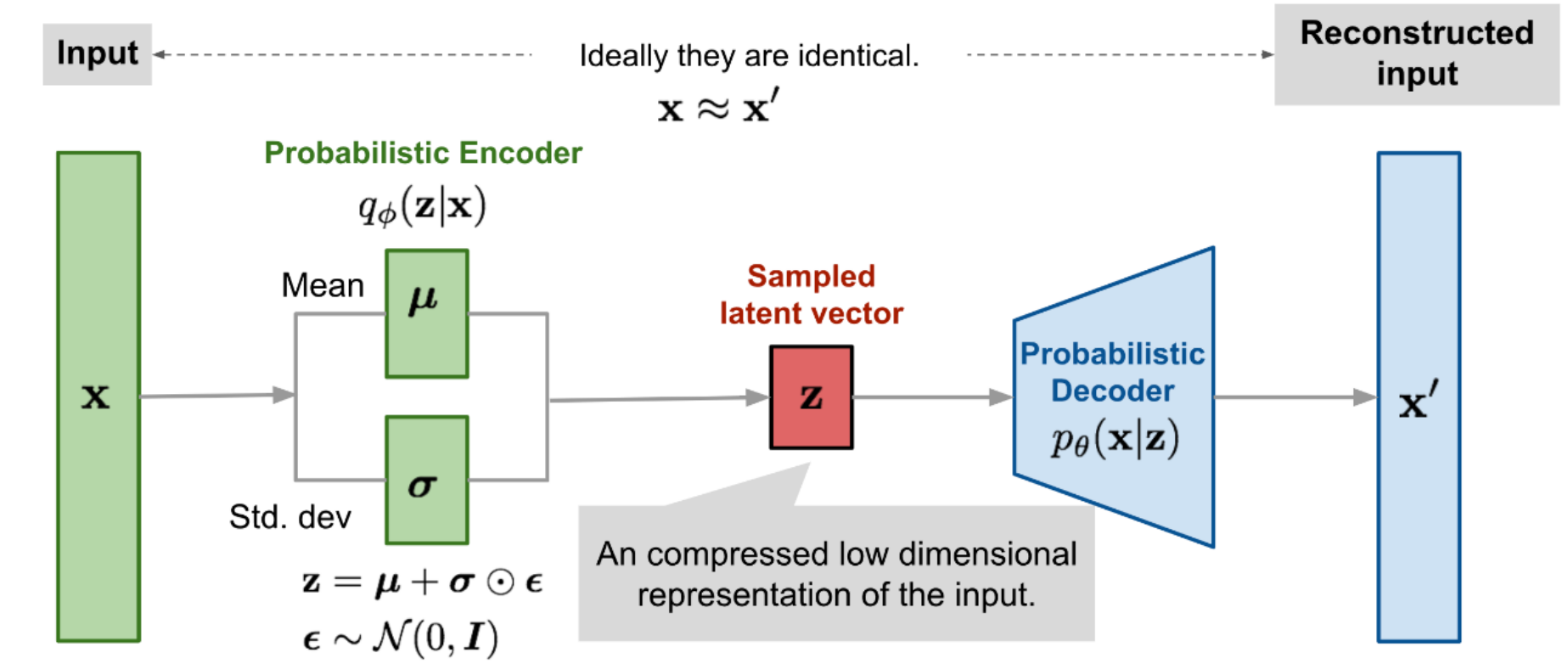
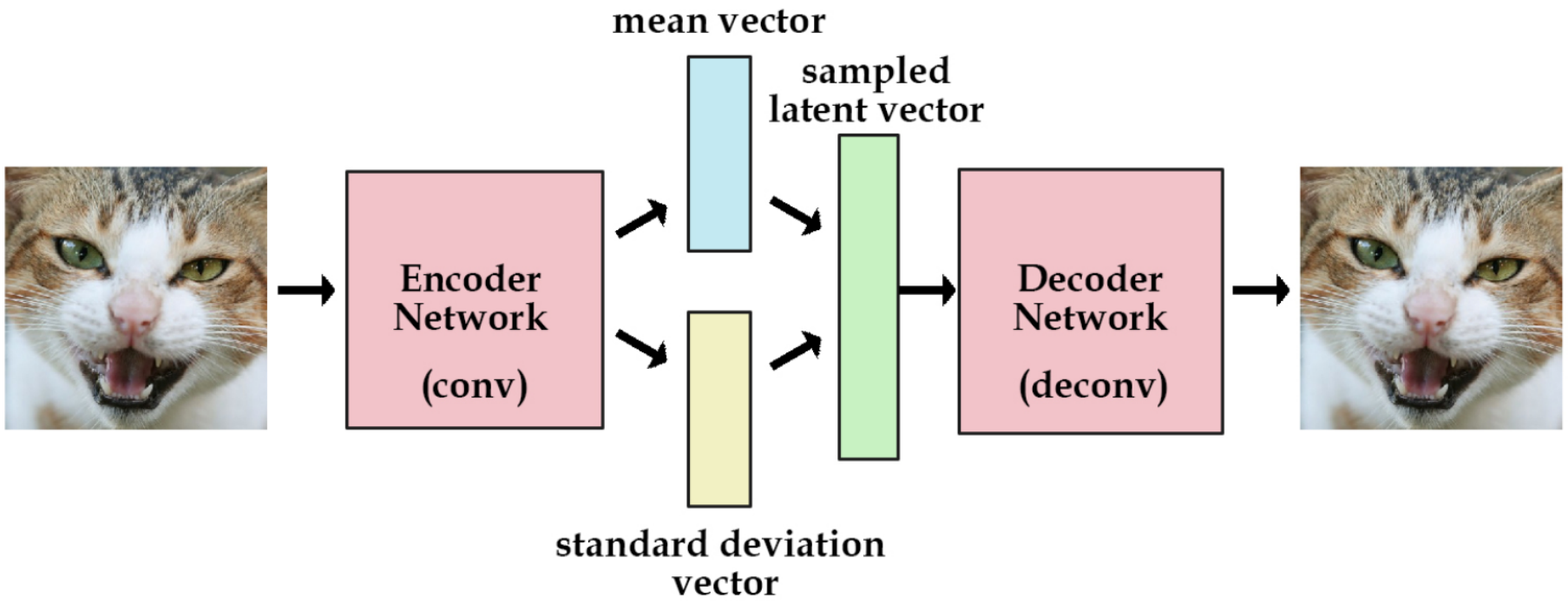
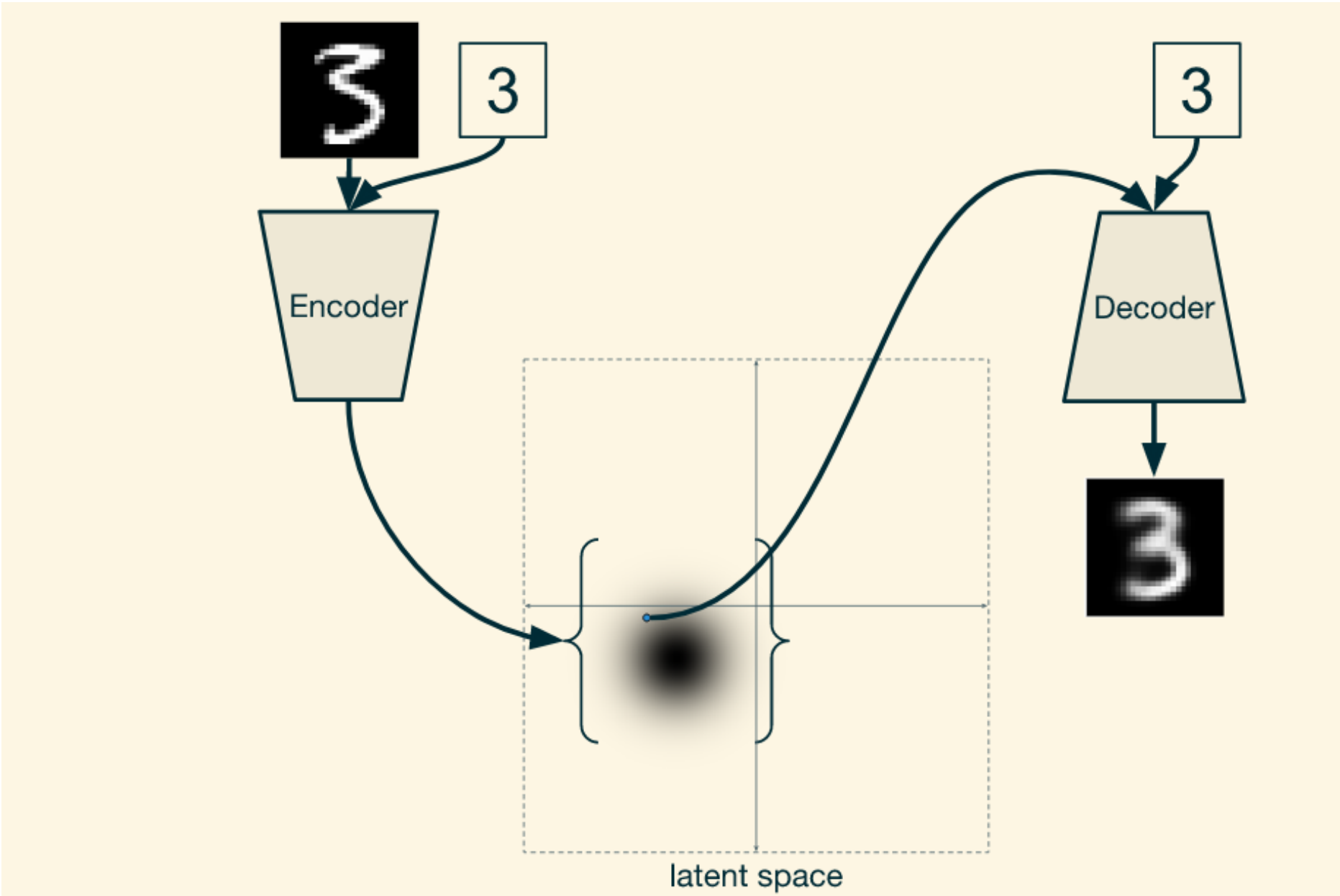


Fig. 9. Illustration of variational autoencoder model with the multivariate Gaussian assumption.


mean vector
sampled
latent vector
Encoder
Network
(conv)
Decoder
Network
(deconv)
standard deviation
vector



A conditional variational autoencoder

Distribution*

- Both the VAE decoder and a GAN generator produce samples from probability distributions

$$P(\mathbf{x}) = \int \mathcal{N}(\mathbf{x}; g(\mathbf{h}), \epsilon^2 I)\mathcal{N}(\mathbf{h}; 0, I))d\mathbf{h}$$

  where latent features are generated from $\mathcal{N}(\mathbf{h}; 0, I))$ and where we added a tiny noise with variance ϵ^2 to the generator

- With $\epsilon^2 \to 0$, points outside the manifold will get zero probability density

- Obtaining the probability value of $P(\mathbf{x})$ by the last formula is not straight forward

Variational Autoencoder*

- We apply mean field theory, which is a special case of a variational method
- We consider one data point with measured $\mathbf{x}$
- The contribution to the log likelihood of that data point is

$$l(\mathbf{x}) = \log P(\mathbf{x}) = \log \int P(\mathbf{h})P(\mathbf{x}|\mathbf{h})d\mathbf{h}$$

  The VAE assumes that $P(\mathbf{h}) = \mathcal{N}(\mathbf{h}; 0, I)$

- With approximating density $Q(\mathbf{h})$, we write

$$l(\mathbf{x}) = \log \int Q(\mathbf{h}) \frac{P(\mathbf{h})P(\mathbf{x}|\mathbf{h})}{Q(\mathbf{h})} d\mathbf{h}$$

  Using Jensen's inequality

$$l(\mathbf{x}) \geq \int Q(\mathbf{h})[\log P(\mathbf{h}) + \log P(\mathbf{x}|\mathbf{h}) - \log Q(\mathbf{h})]d\mathbf{h}$$

Variational Autoencoder (cont'd)

- The last expression can be written as

$$\mathbb{E}_Q[\log(P(\mathbf{h})P(\mathbf{x}|\mathbf{h}))] - \mathbb{E}_Q[\log Q(\mathbf{h})]$$

  or as

$$\mathbb{E}_Q[\log P(\mathbf{x}|\mathbf{h})] - \mathbb{E}_Q[\log Q(\mathbf{h}) - \log P(\mathbf{h})]$$

  The second term in the last equation is the same as (Kullback-Leibler divergence) $D_{KL}(Q(\mathbf{h})\|P(\mathbf{h}))$, so we maximize

$$\mathbb{E}_Q[\log P(\mathbf{x}|\mathbf{h})] - D_{KL}(Q(\mathbf{h})\|P(\mathbf{h}))$$

- With a least squares cost function, the first term becomes, with $\mathbf{h}_s$ being a sample from $Q(\mathbf{h})$,

$$\int Q(\mathbf{h}) \log P(\mathbf{x}|\mathbf{h}) d\mathbf{h} \approx const - \frac{\|\mathbf{x} - g_{\mathbf{v}}^{gen}(\mathbf{h}_s)\|^2}{2\sigma_\epsilon^2}$$

Variational Autoencoder (cont'd)

- The VAE assumes for the approximating distribution, $Q(\mathbf{h}) = \mathcal{N}(\mathbf{h}; \mu_{\mathbf{u}}(\mathbf{x}), \Sigma_{\mathbf{u}}(\mathbf{x}))$. Then, (see Wikipedia page on Kullback-Leibler divergence),

$$D_{KL}(Q(\mathbf{h})\|P(\mathbf{h})) = \frac{1}{2}\left(\text{tr}\,(\Sigma_{\mathbf{u}}(\mathbf{x})) + \mu_{\mathbf{u}}(\mathbf{x})^{\mathsf{T}}\mu_{\mathbf{u}}(\mathbf{x}) - M_{hidd} - \ln\det\Sigma_{\mathbf{u}}(\mathbf{x})\right)$$

- When $\Sigma_{\mathbf{u}}(\mathbf{x})$ is diagonal, the encoder has $2M_{hidd}$ outputs

Variational Autoencoder: Reparameterization trick

- Reparameterization trick: since we cannot do backpropagation through the sampling, we sample ϵ_s from $\mathcal{N}(0, I)$ and multiply the sample by $\sqrt{\Sigma_{\mathbf{u}}(\mathbf{x})}$ and add $\mu_{\mathbf{u}}$.

Variational Autoencoder (cont'd)

- The overall cross entropy cost function with a diagonal $\Sigma_{\mathbf{u}}(\mathbf{x})$, and which we use to minimize w.r.t $\mathbf{w}$ and $\mathbf{u}$, is for data point $\mathbf{x}$ and random sample $\epsilon_s \sim \mathcal{N}(0, I)$

$$cost(\mathbf{x}, \epsilon_s) = \frac{1}{2\sigma_\epsilon^2} \|\mathbf{x} - g_{\mathbf{v}}^{gen}(\mu_{\mathbf{u}}(\mathbf{x}) + \sqrt{\Sigma_{\mathbf{u}}(\mathbf{x})} \circ \epsilon_s)\|^2$$

$$+\frac{1}{2}\left(\mu(\mathbf{x})^{\mathsf{T}}\mu(\mathbf{x}) + \sum_j \left[\Sigma_{\mathbf{u},j,j}(\mathbf{x}) - \log \Sigma_{\mathbf{u},j,j}(\mathbf{x})\right]\right)$$

- The first term in the large bracket encourages $\mu_{\mathbf{u}}(\mathbf{x}) \rightarrow 0$; the second term is minimum if $\Sigma_{\mathbf{u},j,j} = 1, \forall j$

Variational Autoencoder (cont'd)

- Encoder: $\mu_{\mathbf{u}}(\mathbf{x})$ and $\Sigma_{\mathbf{u}}(\mathbf{x})$ generate representations also for test inputs (this is a major advance: in previous variational approximations, this was an iterative step))
- Decoder/Generator for a new $\mathbf{x}$: To generate a new data point we sample $\mathbf{h}_s$ from $\mathcal{N}(0, I)$ and calculate $g_{\mathbf{v}}^{gen}(\mathbf{h}_s)$.
- See: Tutorial on Variational Autoencoders, Carl Doersch

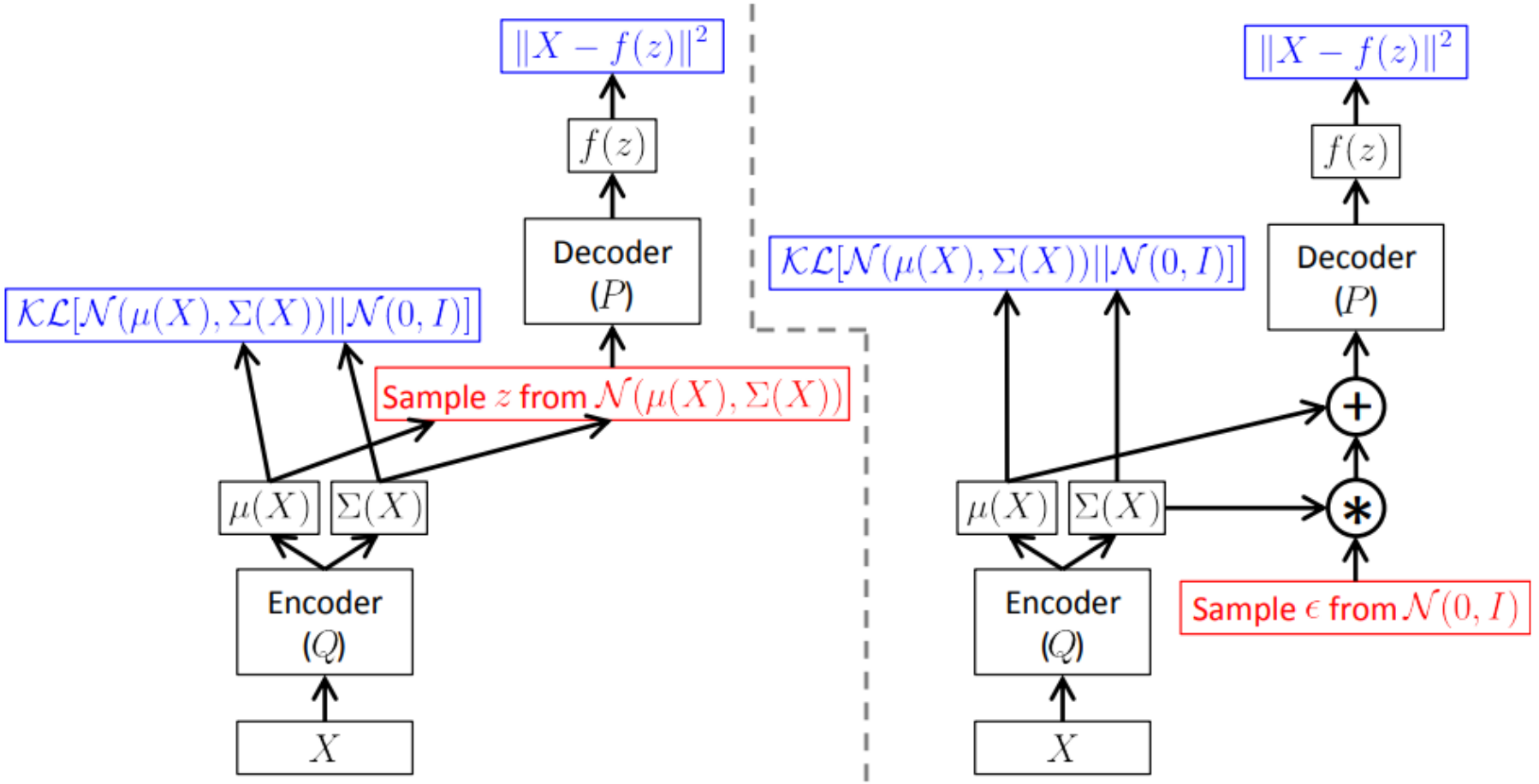


Figure 4: A training-time variational autoencoder implemented as a feedforward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward

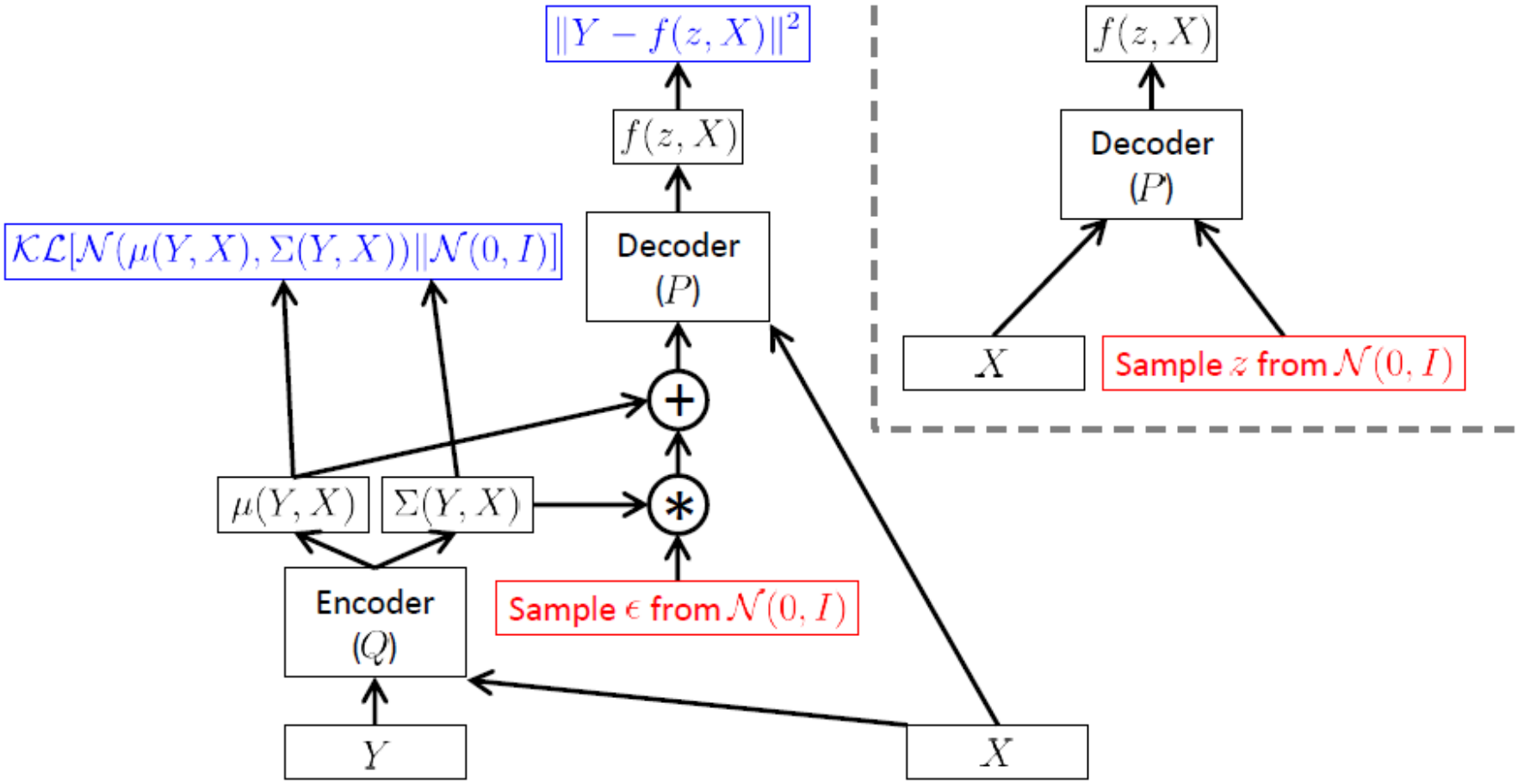


Figure 6: Left: a training-time conditional variational autoencoder implemented as a feedforward neural network, following the same notation as Figure 4. Right: the same model at test time, when we want to sample from $P(Y|X)$.