Deep Learning (BEV033DLE)
Lecture 11 Variational Autoencoders

Czech Technical University in Prague

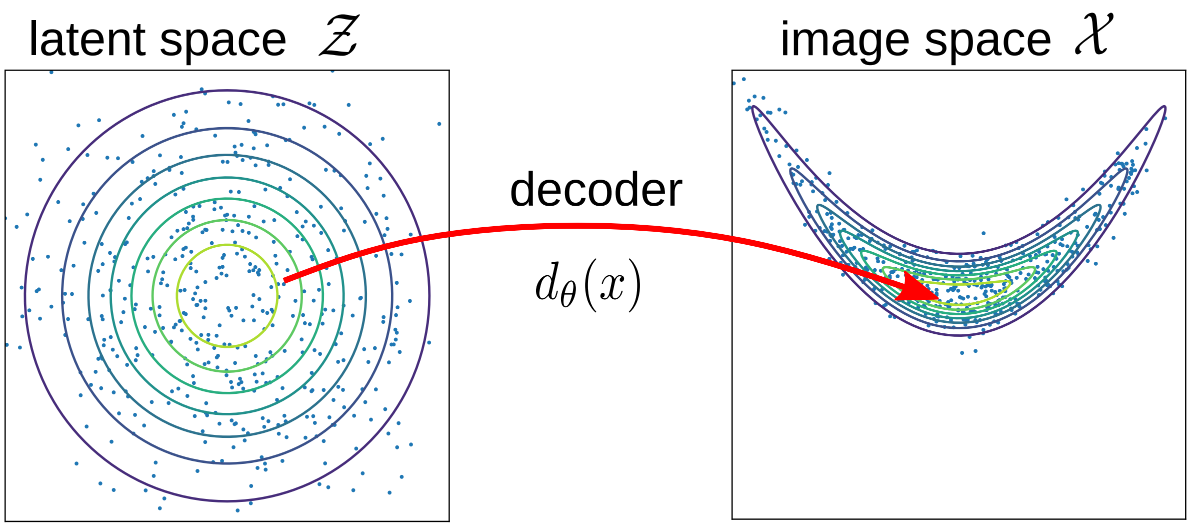
- Generative models in machine learning
- Variational autoencoders (VAE)
- Alternative approaches

Generative models

Generative models: Given training data $\mathcal{T} = \{x_j \mid j = 1, \ldots, \ell\}$ drawn i.i.d. from an unknown distribution $p_d(x)$, the goal is to learn a DNN model that allows to generate random instances of x similar to $x \sim p_d(x)$.

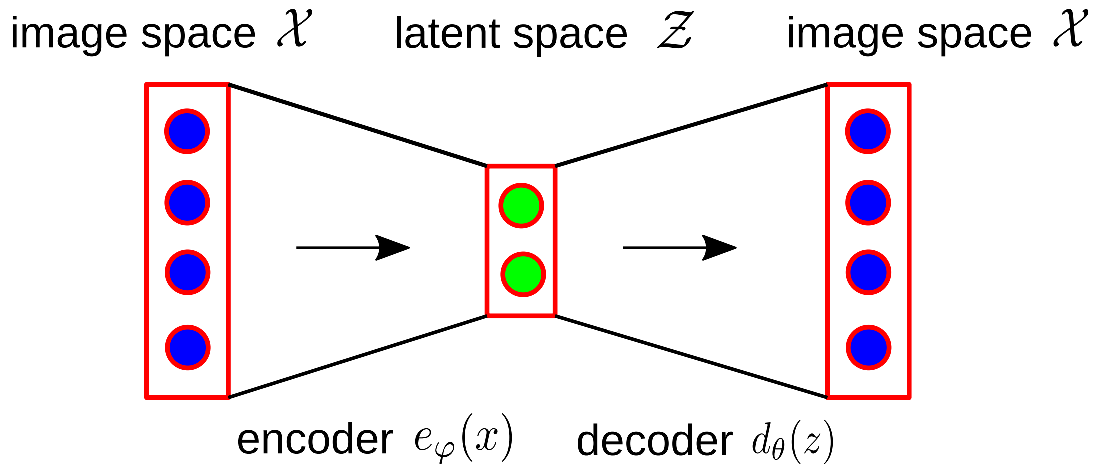
Approach this task by using *latent variable models:*

- fix a latent noise space $\mathcal{Z}$ and a distribution $p(z)$ on it,
- design a neural network d_θ that maps $\mathcal{Z}$ to the feature space $\mathcal{X}$,
- learn its parameters θ so that the resulting distribution $p_\theta(x)$ "reproduces" the data distribution.



Generative models

Classical autoencoder networks



e.g. with learning criterion $\mathbb{E}_{\mathcal{T}} \|x - d_\theta \circ e_\varphi(x)\|^2$. However,

- the distribution in the latent space is beyond our control,
- the model can not be used for sampling/generating x instances.

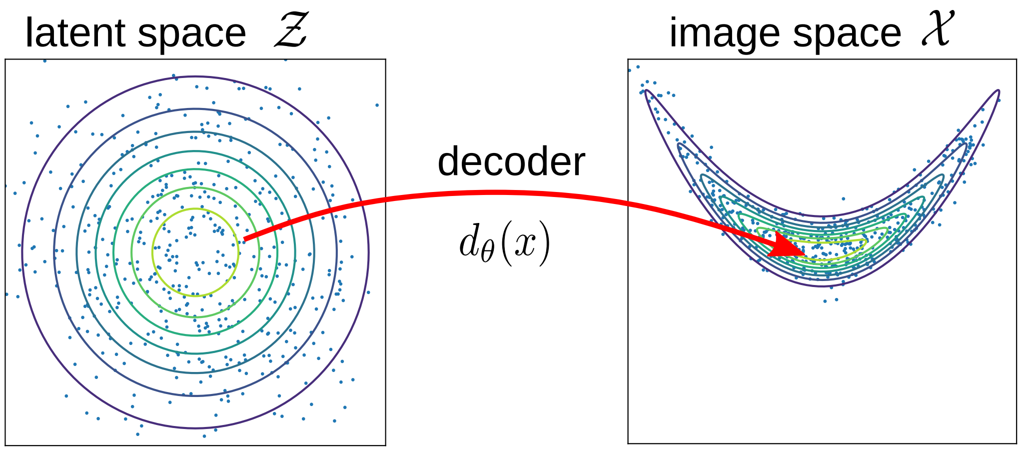
(Gaussian) Variational Autoencoders

- latent space $\mathcal{Z} = \mathbb{R}^m$, prior distribution $p(z)$: $\mathcal{N}(0, \mathbb{I})$
- image space $\mathcal{X} = \mathbb{R}^n$, conditional distribution $p_\theta(x \,|\, z)$: $\mathcal{N}(\mu_\theta(z), \sigma^2 \mathbb{I})$

  The mapping $\mathcal{Z} \ni z \mapsto \mu_\theta \in \mathcal{X}$ is modelled in terms of a (deep, convolutional) *decoder network* $d_\theta \colon \mathcal{Z} \to \mathcal{X}$.
- Learning goal: maximise data log-likelihood

$$L(\theta; \mathcal{T}) = \mathbb{E}_{\mathcal{T}} \log p_\theta(x) = \mathbb{E}_{\mathcal{T}} \log \int_{\mathcal{Z}} dz \, p_\theta(x \,|\, z) p(z)$$

Computing $L(\theta)$ or $\nabla_\theta L(\theta)$ is not tractable! It would require to integrate the decoder mapping $d_\theta(z)$ over the latent space $\mathcal{Z}$:



(Gaussian) Variational Autoencoders

Use ELBO, i.e. a lower bound of the data log-likelihood

$$L(\theta) \geqslant L_B(\theta) = \mathbb{E}_{\mathcal{T}} \mathbb{E}_{q(z\,|\,x)} \Big[\log p_\theta(x\,|\,z) - \log \frac{q(z\,|\,x)}{p(z)} \Big]$$

May be we can apply the *EM algorithm* directly?

E-step fix θ_t, set $q_t(z\,|\,x) = p_{\theta_t}(z\,|\,x)$

M-step fix $q_t(z\,|\,x)$, maximise $\mathbb{E}_{\mathcal{T}} \mathbb{E}_{q_t(z\,|\,x)} \log p_\theta(x\,|\,z) \to \max_\theta$

No, computing $p_{\theta_t}(z\,|\,x)$ would require to "invert" the decoder network.

Way out: choose a class of *amortized inference* models $q_\varphi(z\,|\,x) \colon \mathcal{N}\big(\mu_\varphi(x), \operatorname{diag}(\sigma^2_\varphi(x)\big)$.

The mapping $x \mapsto \mu_\varphi(x), \sigma_\varphi(x)$ is modelled in terms of a (deep, convolutional) *encoder network* $e_\varphi \colon \mathcal{X} \to (\mathcal{Z}, \mathcal{Z})$.

The ELBO criterion reads now

$$L_B(\theta, \varphi) = \mathbb{E}_{\mathcal{T}} \Big[\mathbb{E}_{q_\varphi(z\,|\,x)} \log p_\theta(x\,|\,z) - D_{KL}(q_\varphi(z\,|\,x) \,\|\, p(z)) \Big]$$

Can we maximise it by gradient ascent?

(Gaussian) Variational Autoencoders

$$L_B(\theta,\varphi) = \mathbb{E}_{\mathcal{T}}\Big[\mathbb{E}_{q_\varphi(z\,|\,x)} \log p_\theta(x\,|\,z) - D_{KL}(q_\varphi(z\,|\,x) \parallel p(z))\Big]$$

- $\mathbb{E}_{\mathcal{T}}$: SGD with mini-batches ✓
- $D_{KL}(q_\varphi(z\,|\,x) \parallel p(z))$: both Gaussians factorise and the KL-divergence decomposes into a sum over components $\sum_{i=1}^m D_{KL}(q_\varphi(z_i\,|\,x) \parallel p(z_i))$. The KL-divergence of univariate Gaussian distributions can be computed in closed form! ✓
- $\nabla_\theta \mathbb{E}_{q_\varphi(z\,|\,x)} \log p_\theta(x\,|\,z)$: use SGD by sampling $z \sim q_\varphi(z\,|\,x)$. ✓
- $\nabla_\varphi \mathbb{E}_{q_\varphi(z\,|\,x)} \log p_\theta(x\,|\,z)$: this gradient is *critical*. We can not simply replace $\mathbb{E}_{q_\varphi(z\,|\,x)}$ by a sample $z \sim q_\varphi(z\,|\,x)$, because it will depend on φ!

  Consider $\nabla_\varphi \mathbb{E}_{q_\varphi(z)} f(z)$: if we replace $\mathbb{E}_{q_\varphi(z)}$ by a finite sample $\mathcal{S}$ with elements $z \sim q_\varphi(z)$, then $\nabla_\varphi \sum_{z\in\mathcal{S}} f(z) = ?$

  Re-parametrisation trick: Simple solution for Gaussians:

$$\mathbb{E}_{z\sim\mathcal{N}(\mu,\sigma^2)}[f(z)] = \mathbb{E}_{z\sim\mathcal{N}(0,1)}\big[f(\sigma z + \mu)\big]$$

  Now, if μ and σ depend on φ:

$$\nabla_\varphi \mathbb{E}_{z\sim\mathcal{N}(\mu_\varphi,\sigma_\varphi^2)}[f(z)] = \mathbb{E}_{z\sim\mathcal{N}(0,1)}\big[\nabla_\varphi f(\sigma_\varphi z + \mu_\varphi)\big]$$

(Gaussian) Variational Autoencoders

Overall, the learning step for a (Gaussian) VAE is pretty simple:

Fetch a mini-batch x from training data

1. apply the encoder network $e_\varphi(x) \mapsto \mu_\varphi(x), \sigma_\varphi(x)$ and compute $q_\varphi(z \,|\, x)$
2. compute the KL-divergence $D_{KL}(q_\varphi(z \,|\, x) \,\|\, p(z))$
3. sample a batch $z \sim q_\varphi(z \,|\, x)$ with reparametrisation
4. apply the decoder network $d_\theta(z) \mapsto \mu_\theta(z)$ and compute $\log p_\theta(x \,|\, z)$
5. combine the ELBO terms and let PyTorch compute the derivatives and make an SGD step.

Strengths and weaknesses of VAEs

- concise model, simple objective (ELBO), can be optimised by SGD ✓
- local optima, *posterior collapse*: some latent components collapse to $q_\varphi(z_i \,|\, x) = p(z_i)$, i.e. they carry no information. ✗
- amortized inference model $q_\varphi(z \,|\, x)$ may have not enough expressive power to close the gap between $L(\theta)$ and $L_B(\theta, \varphi)$. ✗

(Gaussian) Variational Autoencoders

Advanced VAEs with strong encoders can generate very good images. A. Vahdat et al., NeurIPS 2020: A Deep Hierarchical VAE trained on CelebA data.

Alternative Approaches

Keep latent space and $p(z)$, consider *deterministic* decoders $D_\theta(z)$, which map $z \in \mathcal{Z} \mapsto x \in \mathcal{X}$. This mapping induces a probability distribution $p_\theta(x)$ on $\mathcal{X}$

Design a quantitative "measure" for the difference between the distributions $p_d(x)$ and $p_\theta(x)$ and try to minimise it.

Popular examples: Generative Adversarial Networks (GAN)

- GAN: uses a binary classifier network and trains it to distinguish natural images (training data) from generated ones.
- WGAN: uses Wasserstein distance to measure the difference between $p_d(x)$ and $p_\theta(x)$.