

Likelihood-based deep generative models

beyond VAEs

Pierre-Alexandre Mattei



MSc Data Science, Université Côte d'Azur

Recap from 2020...

- We saw together **how to use deep learning as a building block within generative modelling**
- In particular, we studied **deep latent variable models (DLVMs)** and **variational autoencoders (VAEs)** in depth
- You studied with Pr. Precioso many deep architectures, including **convolutional neural nets (CNNs)** and **recurrent neural nets (RNNs)**
- Today, **we will see how these architectures can be used to create deep generative models**

What's a generative model?

- We have some data $x_1, \dots, x_n \in \mathcal{X}$ from an unknown distribution p_{true}
- We want to **learn a probability distribution** p_θ such that $p_\theta \approx p_{\text{true}}$
- Why is this interesting to have access to $p_\theta(x)$?

What's a generative model?

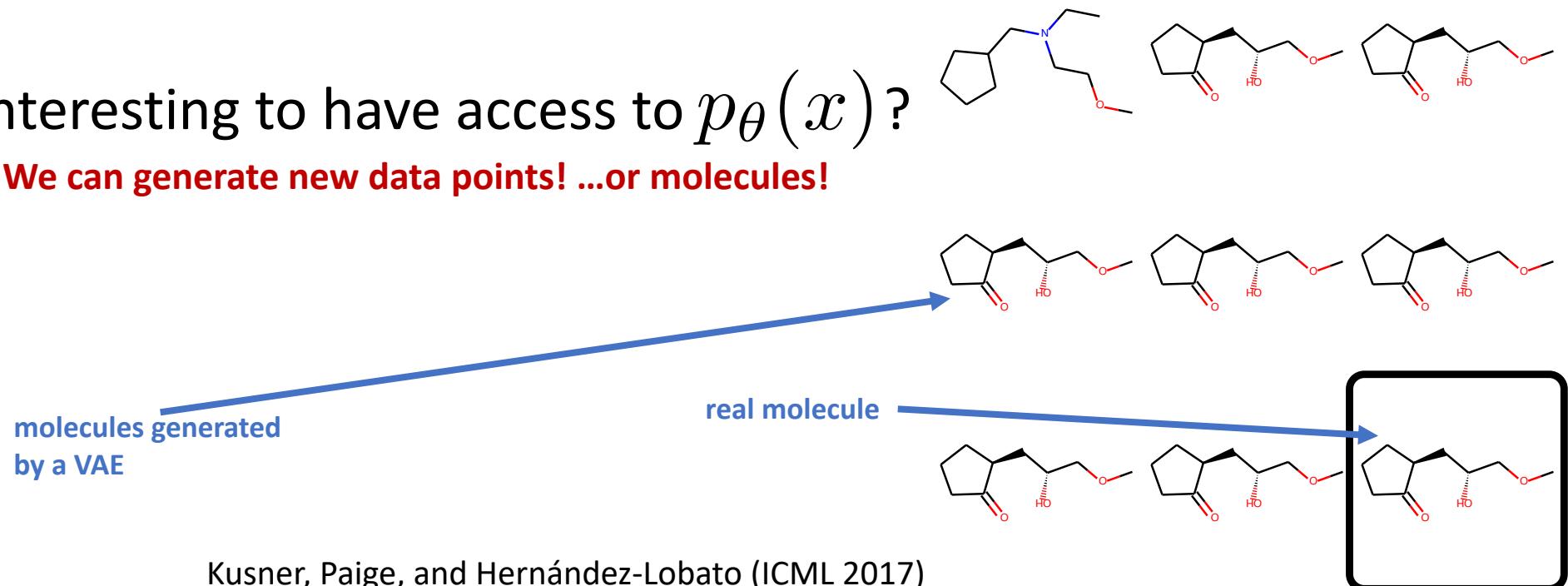
- We have some data $x_1, \dots, x_n \in \mathcal{X}$ from an unknown distribution p_{true}
- We want to **learn a probability distribution** p_θ such that $p_\theta \approx p_{\text{true}}$
- Why is this interesting to have access to $p_\theta(x)$?
 - **We can generate new data points!** ...like images...



Images generated by a VAE
Vahdat and Kautz (NeurIPS 2020)

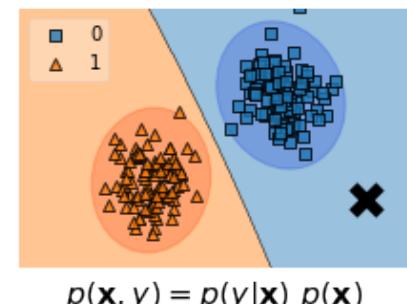
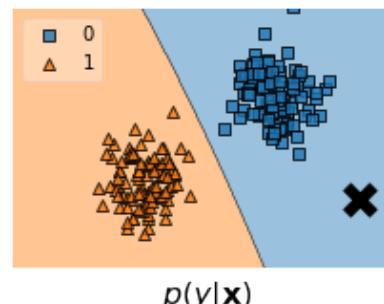
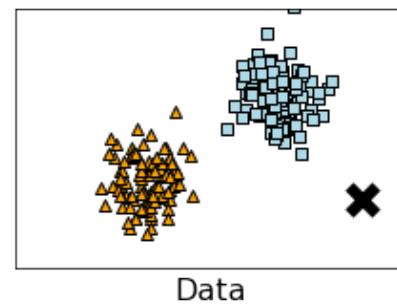
What's a generative model?

- We have some data $x_1, \dots, x_n \in \mathcal{X}$ from an unknown distribution p_{true}
- We want to **learn a probability distribution** p_{θ} such that $p_{\theta} \approx p_{\text{true}}$
- Why is this interesting to have access to $p_{\theta}(x)$?
 - **We can generate new data points! ...or molecules!**



What's a generative model?

- We have some data $x_1, \dots, x_n \in \mathcal{X}$ from an unknown distribution p_{true}
- We want to **learn a probability distribution** p_θ such that $p_\theta \approx p_{\text{true}}$
- Why is this interesting to have access to $p_\theta(x)$?
 - It can allow to **assess uncertainty!**



$p(\text{blue}|\mathbf{x})$ is high
= certain decision!

$p(\text{blue}|\mathbf{x})$ is high
and $p(\mathbf{x})$ is low
= uncertain decision!

What's a generative model?

- We have some data $x_1, \dots, x_n \in \mathcal{X}$ from an unknown distribution p_{true}
 - We want to **learn a probability distribution** p_θ such that $p_\theta \approx p_{\text{true}}$
 - Why is this interesting to have access to $p_\theta(x)$?
 - **It can be used to handle missing data!**

Single imputation with a VAE



Multiple imputation with a VAE



How to train generative models?

- We have a family of probability distributions $(p_\theta(x))_{\theta \in \Theta}$
- We want to find the **right** θ in order to have $p_\theta \approx p_{\text{true}}$
- What techniques do you know to do this?

How to train generative models?

- We have a family of probability distributions $(p_\theta(x))_{\theta \in \Theta}$
- We want to find the **right** θ in order to have $p_\theta \approx p_{\text{true}}$

- What techniques do you know to do this?
- **Maximum likelihood** chooses

$$\hat{\theta} = \operatorname{argmax}_\theta \log p_\theta(x_1, \dots, x_n)$$

- We'll focus today on maximum likelihood techniques, but there are alternatives, for example **adversarial training (cf GANs)** or **score matching**

Recap: Information-theoretic foundations of maximum likelihood

- Why is maximum likelihood a decent idea?
- We want to have $p_\theta \approx p_{\text{true}}$
- We can measure the distance between these two distributions using the **Kullback-Leibler divergence**.

$$\mathbf{KL}(p||q) = \mathbb{E}_{x \sim p} \left[\log \frac{p(x)}{q(x)} \right]$$

Recap: Information-theoretic foundations of maximum likelihood

- In the case of iid data, maximum likelihood becomes

$$\hat{\theta} = \operatorname{argmax}_{\theta} \sum_{i=1}^n \log p_{\theta}(x_i)$$

- We saw in a previous lecture that, when n goes to infinity

$$\operatorname{argmax}_{\theta} \sum_{i=1}^n \log p_{\theta}(x_i) \approx \operatorname{argmin}_{\theta} \text{KL}(p_{\text{true}} || p_{\theta})$$

- This means that **max. likelihood will asymptotically find the closest model to the truth** (in terms of KL divergence).

Recap: maximum likelihood in practice

- In the case of iid data, maximum likelihood becomes

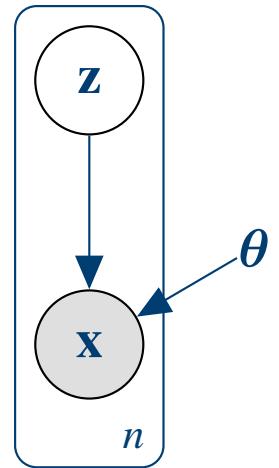
$$\hat{\theta} = \operatorname{argmax}_{\theta} \sum_{i=1}^n \log p_{\theta}(x_i)$$

- Maximising a sum of many differentiable functions can be done efficiently via **stochastic gradient techniques** (e.g. Adam).
- We therefore mainly need to be able to compute efficiently $\nabla_{\theta} \log p_{\theta}(x_i)$
- Problem: when the model is complex, computing this is difficult!

Recap: deep latent variable models and VAEs

Assume that $(\mathbf{x}_i, \mathbf{z}_i)_{i \leq n}$ are i.i.d. random variables driven by the model:

$$\begin{cases} \mathbf{z} \sim p(\mathbf{z}) & \text{(prior)} \\ \mathbf{x} \sim p_{\theta}(\mathbf{x} \mid \mathbf{z}) = \Phi(\mathbf{x} \mid f_{\theta}(\mathbf{z})) & \text{(observation model)} \end{cases}$$



where

- $\mathbf{z} \in \mathbb{R}^d$ is the **latent** variable,
- $\mathbf{x} \in \mathcal{X}$ is the **observed** variable,
- the function $f_{\theta} : \mathbb{R}^d \rightarrow H$ is a **(deep) neural network** called the **decoder**
- $(\Phi(\cdot \mid \eta))_{\eta \in H}$ is a parametric family called the **observation model**, usually **very simple**: unimodal and fully factorised (e.g. multivariate Gaussians or products of multinomials)

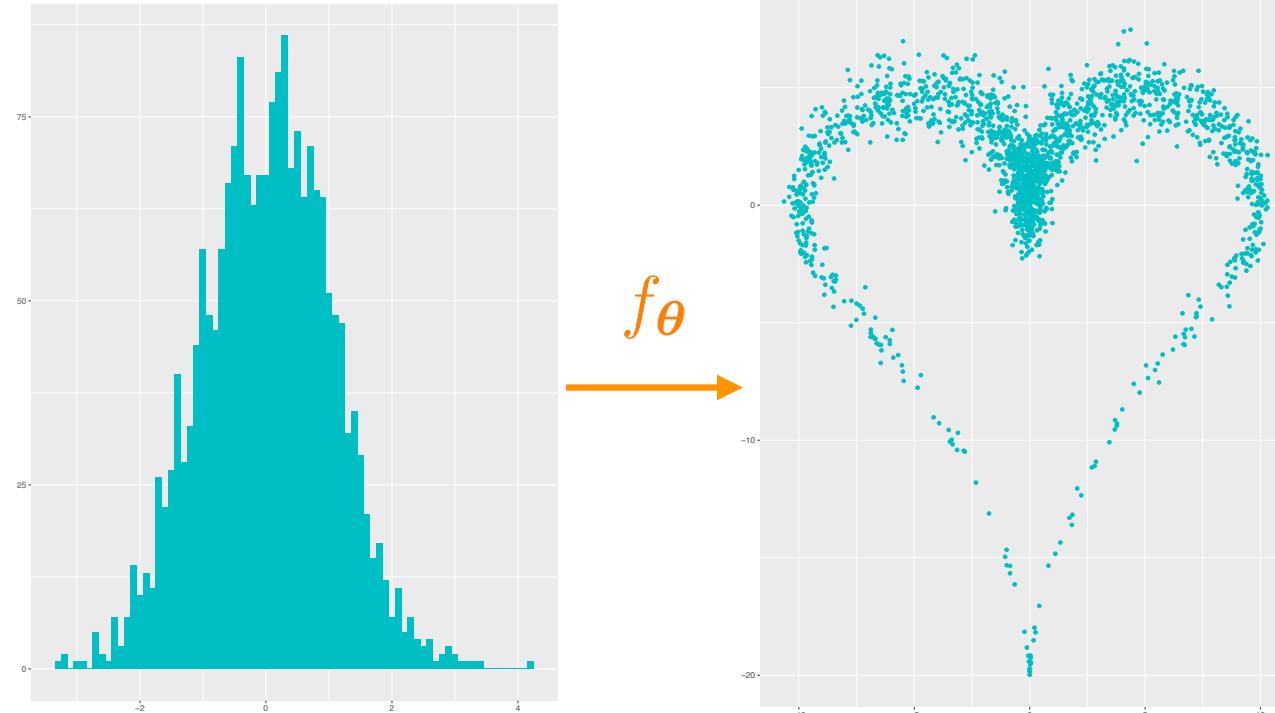
Recap: deep latent variable models and VAEs

The role of the **decoder** $f_{\theta} : \mathbb{R}^d \rightarrow H$ is:

- to transform \mathbf{z} (**the code**) into parameters $\eta = f_{\theta}(\mathbf{z})$ of the observation model $\Phi(\cdot | \eta)$.
- The weights θ of the **decoder** are learned.

Simple non-linear decoder ($d = 1, p = 2$): $f_{\theta}(z) = \mu_{\theta}(\mathbf{z}), \Sigma_{\theta}(\mathbf{z})$ with, for all $z \in \mathbb{R}$,

$$\mu_{\theta}(z) = (10 \sin(z)^3, 10 \cos(z) - 10 \cos(z)^4), \quad \Sigma_{\theta}(\mathbf{z}) = \text{Diag} \left(\left(\frac{\sin(z)}{3z} \right)^2, \left(\frac{\sin(z)}{z} \right)^2 \right).$$



Recap: deep latent variable models and VAEs

Training data $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ binary MNIST



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0, 1\}^{28 \times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

Recap: deep latent variable models and VAEs

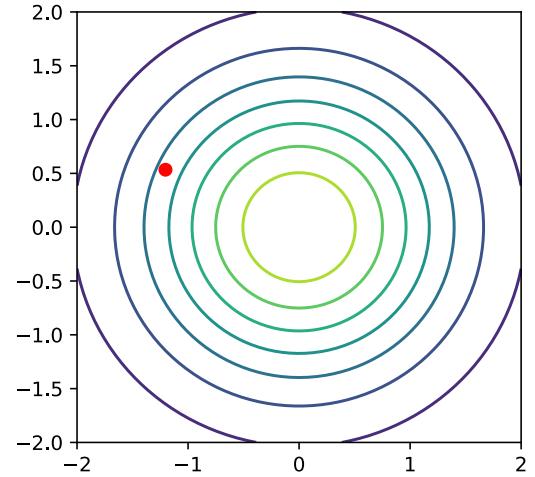
Training data $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ binary MNIST



Generation

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$

$$\mathbf{z} = (-1.2033, 0.5340)$$



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0, 1\}^{28 \times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$

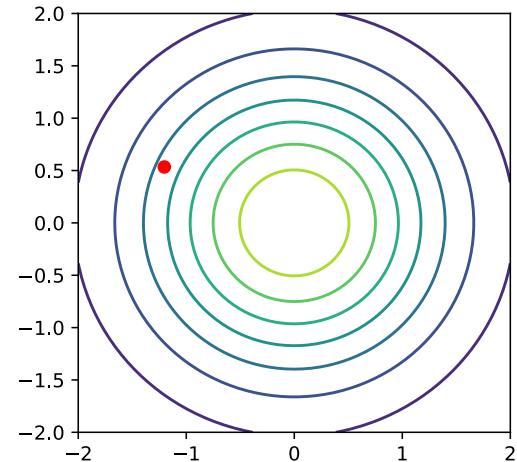
Recap: deep latent variable models and VAEs

Training data $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ binary MNIST



Generation

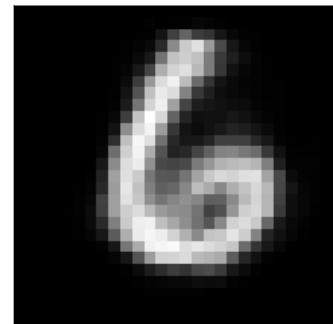
$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$
$$\mathbf{z} = (-1.2033, 0.5340)$$



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0, 1\}^{28 \times 28}$

$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

$$f(\mathbf{z})$$



Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \beta)$$

Recap: deep latent variable models and VAEs

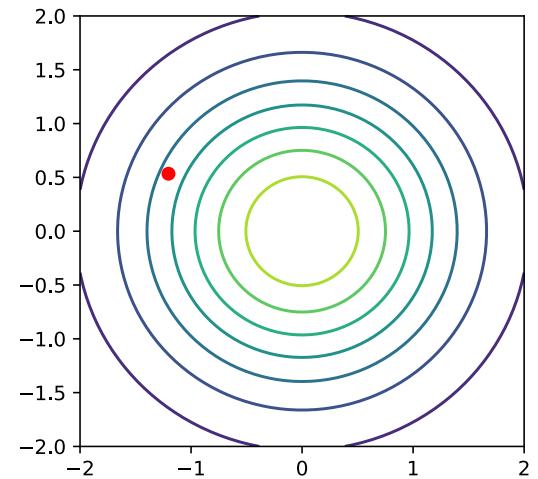
Training data $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ binary MNIST



Generation

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$

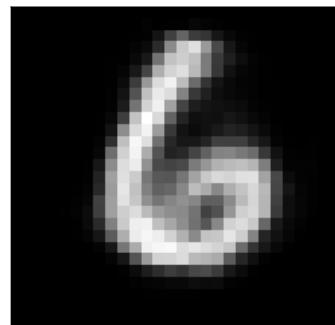
$$\mathbf{z} = (-1.2033, 0.5340)$$



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0, 1\}^{28 \times 28}$

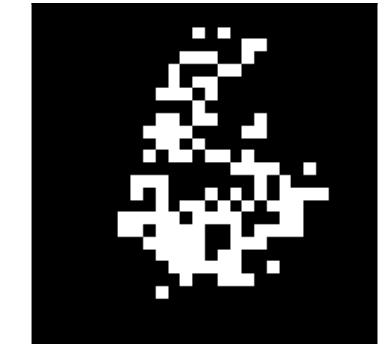
$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

$$f(\mathbf{z})$$



Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$



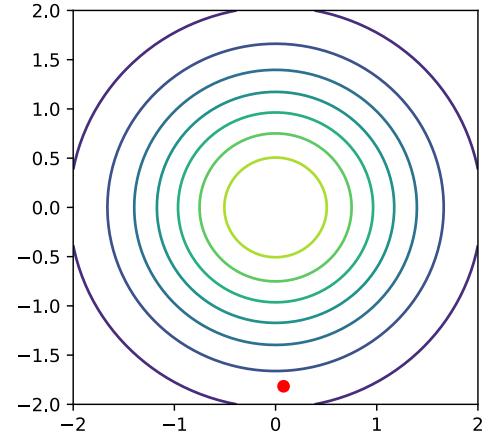
Recap: deep latent variable models and VAEs

Training data $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ binary MNIST



Generation

$$\mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$
$$\mathbf{z} = (0.0791, -1.8165)$$



Generative model for $\mathbf{z} \in \mathbb{R}^2$ and $\mathbf{x} \in \{0, 1\}^{28 \times 28}$

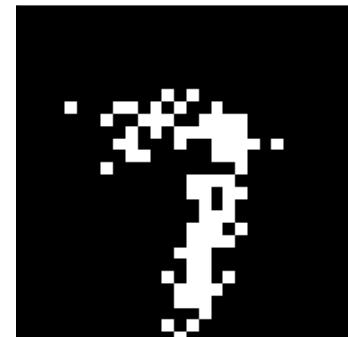
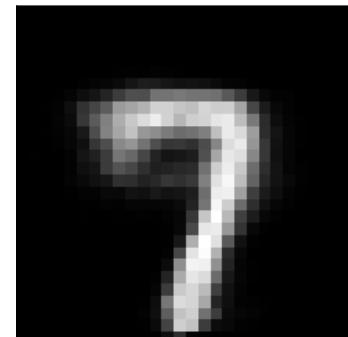
$$\begin{cases} \mathbf{z} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\mathbf{z})) \end{cases}$$

$$f(z)$$

$$x^{j,k} \sim \text{Bern}(f^{j,k}(z))$$

Decoder network

$$f(\mathbf{z}) = \text{Sigmoid}(\mathbf{V} \tanh(\mathbf{W}\mathbf{z} + \mathbf{b}) + \boldsymbol{\beta})$$



Recap: VAE training

- The likelihood is equal to $\ell(\theta) = \log p_{\theta}(\mathbf{X}) = \sum_{i=1}^n \log p_{\theta}(\mathbf{x}_i)$,

which is full of the nasty intractable integrals

$$p_{\theta}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\theta}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}.$$

- How did we manage to train the model?

Recap: VAE training

- The likelihood is equal to $\ell(\theta) = \log p_{\theta}(\mathbf{X}) = \sum_{i=1}^n \log p_{\theta}(\mathbf{x}_i)$,

which is full of the nasty intractable integrals

$$p_{\theta}(\mathbf{x}_i) = \int_{\mathbb{R}^d} p_{\theta}(\mathbf{x}_i \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}.$$

- **How did we manage to train the model?**

Recap: IWAE training of DLVMs

- We maximised an **approximation of the likelihood**, computed using an **encoder**

$$\ell(\boldsymbol{\theta}) \approx \sum_{i=1}^n \mathbb{E}_{\mathbf{z}_{i1}, \dots, \mathbf{z}_{iK} \sim q_{\boldsymbol{\gamma}}(\mathbf{z}|\mathbf{x}_i)} \left[\log \frac{1}{K} \sum_{k=1}^K \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_i | \mathbf{z}_{ik}) p(\mathbf{z}_{ik})}{q_{\boldsymbol{\gamma}}(\mathbf{z}_{ik} | \mathbf{x}_i)} \right] = \mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma})$$

- This approximation also happens to be a **lower bound** of the likelihood that becomes tighter and tighter

$$\mathcal{L}_1(\boldsymbol{\theta}, \boldsymbol{\gamma}) \leq \mathcal{L}_2(\boldsymbol{\theta}, \boldsymbol{\gamma}) \leq \dots \leq \mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma}) \xrightarrow[K \rightarrow \infty]{} \ell(\boldsymbol{\theta}).$$

- The case $K=1$ corresponds to the original VAE bound.

Going beyond VAEs

- One issue with VAEs is that we can't compute the likelihood exactly, but have to rely on approximations
- We will see other families of models where **we can compute the likelihood exactly**
- The two families we'll focus on will be **autoregressive models** and **normalising flows**

Autoregressive models

- We want to find a way to design a flexible $p(\mathbf{x}) = p(x_1, \dots, x_d)$ that we can still compute exactly.
- The main idea is to use probability chain rule

$$p(\mathbf{x}) = p(x_1, \dots, x_d) = p(x_1)p(x_2|x_1)\dots p(x_d|x_1, \dots, x_{d-1})$$

which may be rewritten

$$p(\mathbf{x}) = \prod_{j=1}^d p(x_j|x_1, \dots, x_{j-1})$$

Autoregressive models

- We will use neural nets to model the predictive distribution

$$p_{\theta}(x_j | x_1, \dots, x_{j-1})$$

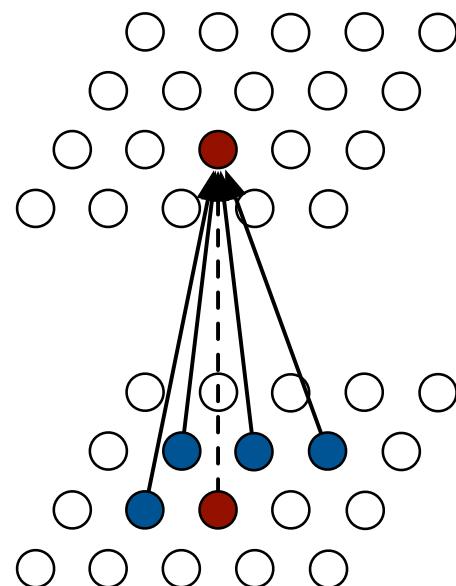
- For example, this could be modelled using a recurrent network (like a LSTM) or even a CNN with some masking
- At the end, **we can compute the likelihood of a single x exactly**

$$p_{\theta}(\mathbf{x}) = \prod_{j=1}^d p_{\theta}(x_j | x_1, \dots, x_{j-1})$$

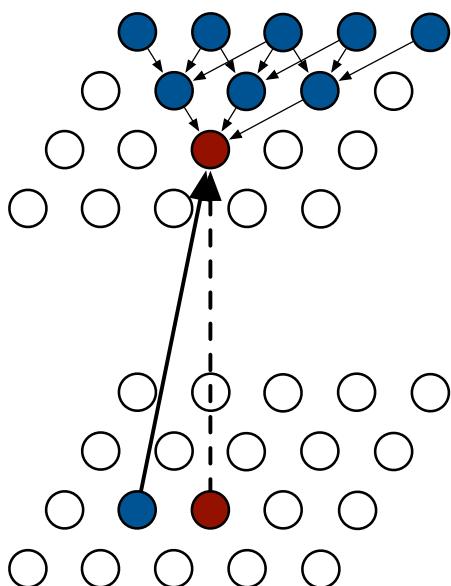
Autoregressive models: examples

- We have a lot of freedom to choose the neural net that parametrises

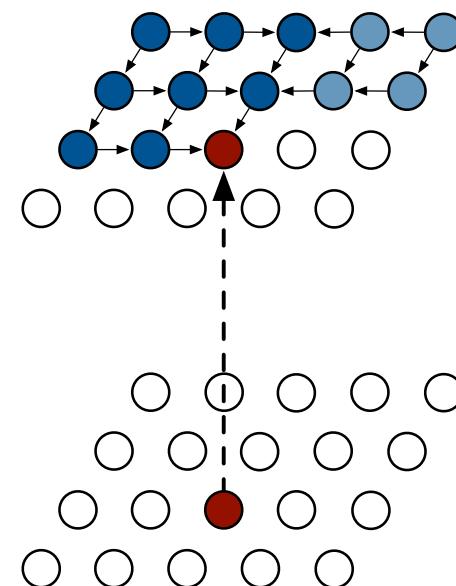
$$p_{\theta}(x_j | x_1, \dots, x_{j-1})$$



PixelCNN



Row LSTM



Diagonal BiLSTM

(van den Oord et al.,
ICML 2016)

Autoregressive models: examples

- We have a lot of freedom to choose the neural net that parametrises

$$p_{\theta}(x_j | x_1, \dots, x_{j-1})$$



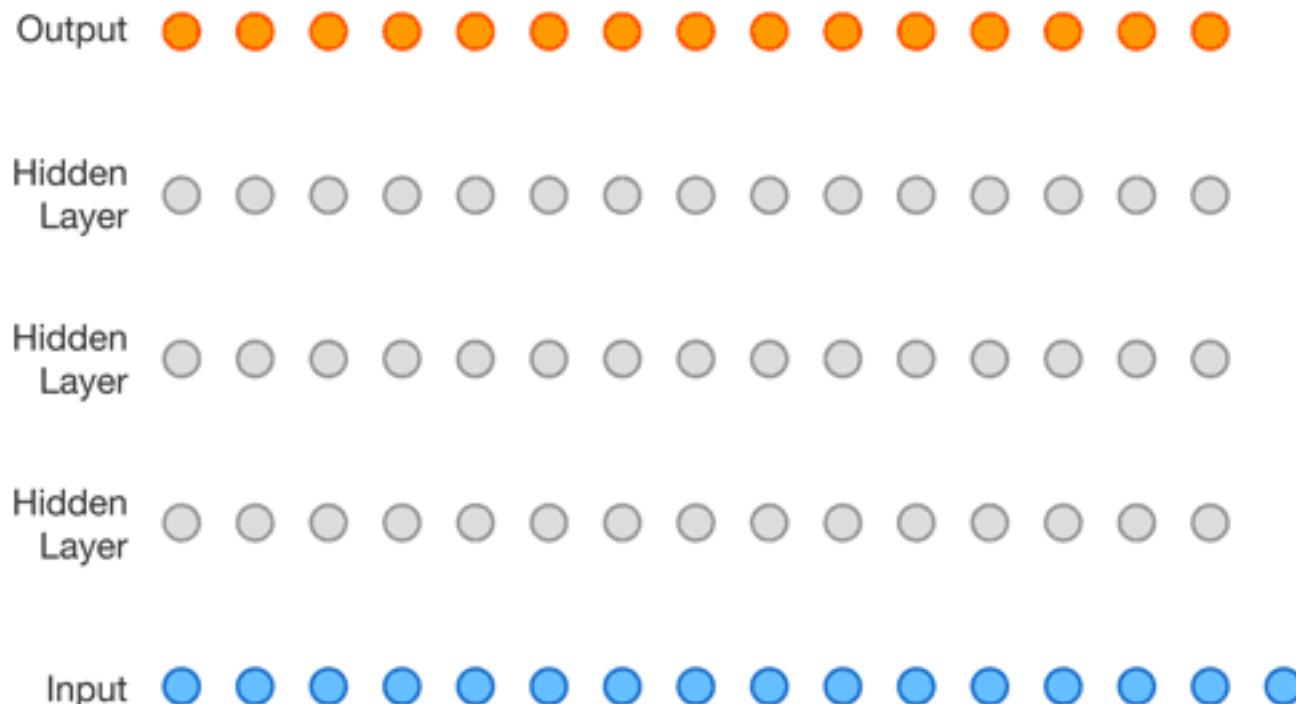
Figure 1. Image completions sampled from a PixelRNN.

(van den Oord et al.,
ICML 2016)

Autoregressive models: examples

- We have a lot of freedom to choose the neural net that parametrises

$$p_{\theta}(x_j | x_1, \dots, x_{j-1})$$



Normalising flows

- Key idea: sample from a simple distribution, and then pass the samples through an **invertible neural net** $f_\theta(z)$

$$z \sim p(z), \quad x = f_\theta(z)$$

- It looks a lot like VAEs except that
 - The “code” has same dimension as the data (to ensure invertibility)
 - The likelihood can be computed exactly using the change of variable formula

$$p_X(\mathbf{x}) = p_Z(z) \left| \det \left(\frac{\partial \mathbf{x}}{\partial z} \right) \right|^{-1}$$

Normalising flows

- The fact that the network is invertible allows us to use the change of variable formula

$$p_X(\mathbf{x}) = p_Z(z) \left| \det \left(\frac{\partial \mathbf{x}}{\partial z} \right) \right|^{-1}$$

- We need to design a decoder $f_\theta(z)$ such that all terms are easy to compute.
- There are many architectures for that purpose, we will see one called **RealNVP** (Dinh et al., ICLR 2017)

Normalising flows: from NICE to RealNVP

Partition the variables \mathbf{z} into two disjoint subsets, say $\mathbf{z}_{1:d}$ and $\mathbf{z}_{d+1:n}$ for any $1 \leq d < n$

- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:
 - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
 - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} + m_\theta(\mathbf{z}_{1:d})$ ($m_\theta(\cdot)$ is a neural network with parameters θ , d input units, and $n - d$ output units)
- Inverse mapping $\mathbf{x} \mapsto \mathbf{z}$:
 - $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$ (identity transformation)
 - $\mathbf{z}_{d+1:n} = \mathbf{x}_{d+1:n} - m_\theta(\mathbf{x}_{1:d})$
- Jacobian of forward mapping:

$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & I_{n-d} \end{pmatrix}$$

$$\det(J) = 1$$

- **Volume preserving transformation** since determinant is 1.

Normalising flows: from NICE to RealNVP

- Forward mapping $\mathbf{z} \mapsto \mathbf{x}$:
 - $\mathbf{x}_{1:d} = \mathbf{z}_{1:d}$ (identity transformation)
 - $\mathbf{x}_{d+1:n} = \mathbf{z}_{d+1:n} \odot \exp(\alpha_\theta(\mathbf{z}_{1:d})) + \mu_\theta(\mathbf{z}_{1:d})$
 - $\mu_\theta(\cdot)$ and $\alpha_\theta(\cdot)$ are both neural networks with parameters θ , d input units, and $n - d$ output units [\odot : elementwise product]
- Inverse mapping $\mathbf{x} \mapsto \mathbf{z}$:
 - $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$ (identity transformation)
 - $\mathbf{z}_{d+1:n} = (\mathbf{x}_{d+1:n} - \mu_\theta(\mathbf{x}_{1:d})) \odot (\exp(-\alpha_\theta(\mathbf{x}_{1:d})))$
- Jacobian of forward mapping:

$$J = \frac{\partial \mathbf{x}}{\partial \mathbf{z}} = \begin{pmatrix} I_d & 0 \\ \frac{\partial \mathbf{x}_{d+1:n}}{\partial \mathbf{z}_{1:d}} & \text{diag}(\exp(\alpha_\theta(\mathbf{z}_{1:d}))) \end{pmatrix}$$

$$\det(J) = \prod_{i=d+1}^n \exp(\alpha_\theta(\mathbf{z}_{1:d})_i) = \exp \left(\sum_{i=d+1}^n \alpha_\theta(\mathbf{z}_{1:d})_i \right)$$

- **Non-volume preserving transformation** in general since determinant can be less than or greater than 1