Deep generative models
VAEs and beyond

Pierre-Alexandre Mattei

Menu for this lecture

1. Recap on VAEs + implementation
2. On d-separation and VAEs
3. Other deep generative models

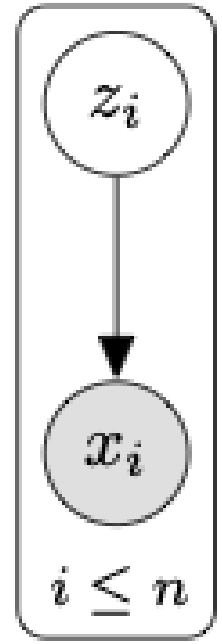
1

Recap on VAEs/IWAEs

Deep latent variable models

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

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



where

- $\mathbf{z} \in \mathbb{R}^d$ is the **latent** variable,
- $\mathbf{x} \in \mathcal{X}$ is the **observed** variable.

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

Maximum likelihood for DLVM

The log-likelihood function of our dataset $(x_1, \ldots, x_n)$ is

$$\ell(\theta) = \sum_{i=1}^{n} \log p(x_i) = \sum_{i=1}^{n} \log \int p(x_i|z)p(z)dz$$

Maximum likelihood for DLVM

The log-likelihood function of our dataset $(x_1, \dots, x_n)$ is

$$\ell(\theta) = \sum_{i=1}^{n} \log p(x_i) = \sum_{i=1}^{n} \log \int p(x_i|z)p(z)dz$$

we saw that an approachable lower bound is the IWAE bound

$$\mathcal{L}_K = \mathbb{E}\left[\sum_{i=1}^{n} \log\left(\frac{1}{K}\sum_{k=1}^{K} \frac{p(x_i|z_{ik})p(z_{ik})}{q(z_{ik}|x_i)}\right)\right] \leq \ell(\theta)$$

Properties of the bound

Monotonicity:

$$\mathcal{L}_1(\theta, \gamma) \leq \ldots \leq \mathcal{L}_K(\theta, \gamma) \leq \ell(\theta)$$

Properties of the bound

Monotonicity:

$$\mathcal{L}_1(\theta, \gamma) \leq \ldots \leq \mathcal{L}_K(\theta, \gamma) \leq \ell(\theta)$$

The encoder learns the best KL approximation to the posterior (only for VAE bound, ie K = 1):

$$\mathcal{L}_1(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \ell(\boldsymbol{\theta}) - \mathrm{KL}\left(\prod_{i=1}^{n} q_{\boldsymbol{\gamma}}(\mathbf{z}_i|\mathbf{x}_i) \middle|\middle| \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{x}_i)\right).$$

Properties of the bound

Monotonicity:

$$\mathcal{L}_1(\theta, \gamma) \leq \ldots \leq \mathcal{L}_K(\theta, \gamma) \leq \ell(\theta)$$

The encoder learns the best KL approximation to the posterior (only for VAE bound, ie K = 1):

$$\mathcal{L}_1(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \ell(\boldsymbol{\theta}) - \mathrm{KL}\left(\prod_{i=1}^{n} q_{\boldsymbol{\gamma}}(\mathbf{z}_i|\mathbf{x}_i) \middle|\middle| \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\mathbf{z}_i|\mathbf{x}_i)\right).$$

Regularised autoencoder formulation (only for VAE bound, ie K = 1):

$$\mathcal{L}_1(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \sum_{i=1}^{n} \mathbb{E}_{\mathbf{z}_i \sim q_{\boldsymbol{\gamma}}(\mathbf{z}|\mathbf{x}_i)} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}_i|\mathbf{z}_i)\right] - \mathrm{KL}\left(\prod_{i=1}^{n} q_{\boldsymbol{\gamma}}(\mathbf{z}_i|\mathbf{x}_i) \middle|\middle| \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\mathbf{z}_i)\right).$$

Reconstruction error:

KL regulariser

Why is this a reconstruction error?

- Gaussian observation model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \sigma^2 \mathbf{I}_D)$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \frac{-D}{2}\log(2\pi) - D\log\sigma - \frac{1}{2\sigma^2}||\mathbf{x} - \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z})||_2^2$$

Mean squared error

Why is this a reconstruction error?

- Gaussian observation model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \sigma^2 \mathbf{I}_D)$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \frac{-D}{2}\log(2\pi) - D\log\sigma - \frac{1}{2\sigma^2}||\mathbf{x} - \boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z})||_2^2$$

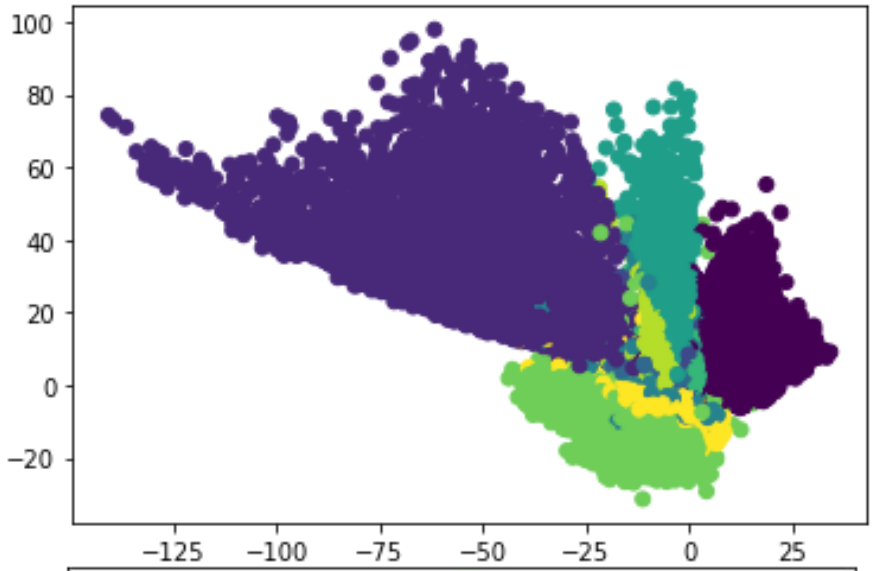
Mean squared error

- Bernoulli observation model $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = \prod_{j=1}^{d} \mathcal{B}(x_j|\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{z})_j)$

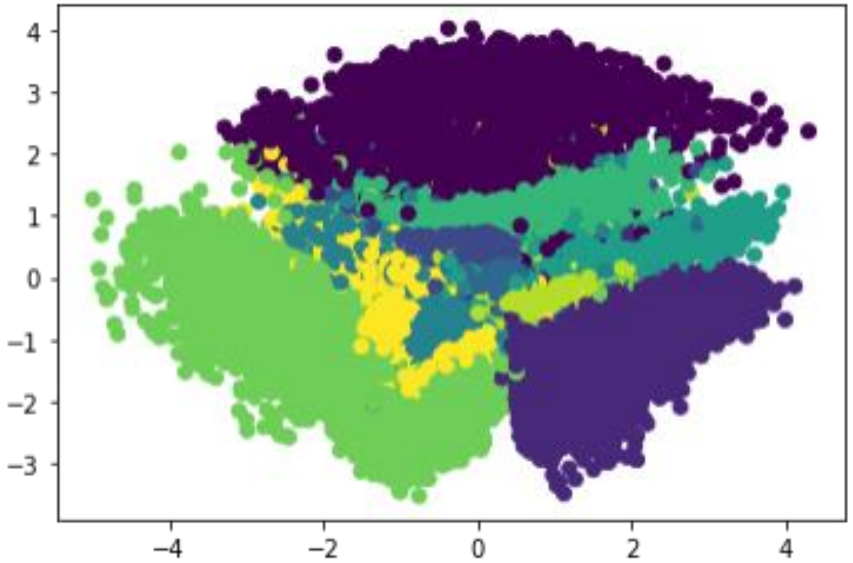
$$\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) = -\sum_{j=1}^{d} \mathrm{XEnt}(\mathbf{x}, \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{z}))$$

Cross-entropy loss

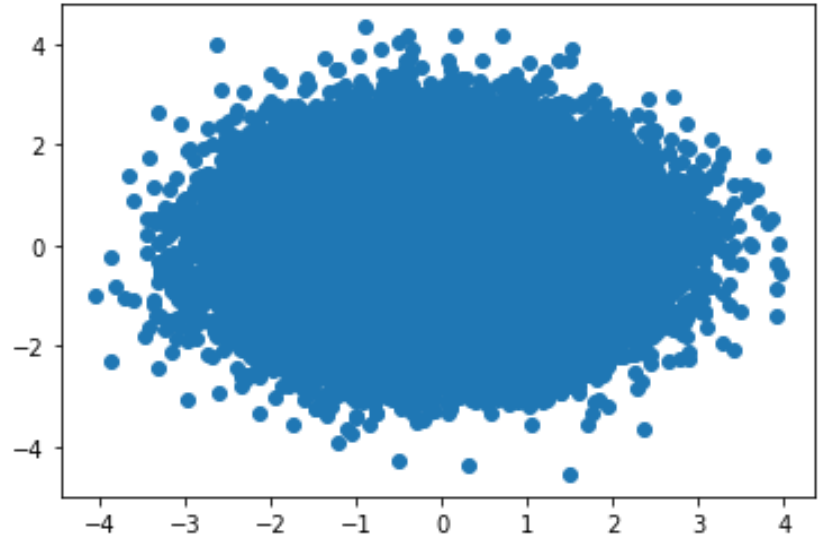
Why is the KL term a regulariser?



2D code space of a standard AE on MNIST



2D code space of a VAE on MNIST



Standard Gaussian samples

How do we actually train VAEs?

- What have we done so far? We have created a family of **lower bounds of the log-likelihood**, but how do optimise them?

- Remember that the IWAE bounds are defined as

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

- **How do we maximise this? We can't even compute this exactly?**

How do we actually train VAEs?

- What have we done so far? We have created a family of **lower bounds of the log-likelihood**, but how do optimise them?

- Remember that the IWAE bounds are defined as

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

- **How do we maximise this? We can't even compute this exactly?**

- The idea is to use stochastic gradient descent (SGD, or one of its variants). What do we need to compute to perform SGD on an objective?

How do we actually train VAEs?

- What have we done so far? We have created a family of **lower bounds of the log-likelihood**, but how do optimise them?

- Remember that the IWAE bounds are defined as

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

- **How do we maximise this? We can't even compute this exactly?**

- The idea is to use stochastic gradient descent (SGD, or one of its variants). What do we need to compute to perform SGD on an objective? **Unbiased estimates of the gradients!**
- As we'll see, it is doable to compute unbiased estimates of $\nabla_{\boldsymbol{\theta},\boldsymbol{\gamma}} \mathcal{L}_K(\boldsymbol{\theta}, \boldsymbol{\gamma})$

Unbiased IWAE gradients: looking at a more general problem

- Let's look at the problem in a more general form, we want unbiased estimates of the gradients of a function of the form

$$f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

- In the next few slides, we will describe a few recipes to compute such estimates. **This is a task that is useful in a lot of ML contexts**, e.g. reinforcement learning, explainability… For more details, and more recipe, you may look at the following nice review

Journal of Machine Learning Research 21 (2020) 1-62

Monte Carlo Gradient Estimation in Machine Learning

Shakir Mohamed[*1] SHAKIR@GOOGLE.COM
Mihaela Rosca[*1 2] MIHAELACR@GOOGLE.COM
Michael Figurnov[*1] MFIGURNOV@GOOGLE.COM
Andriy Mnih[*1] AMNIH@GOOGLE.COM

Let's start with the easy part: ∇_θ

- Note that the first parameter only appears **inside** the expectation

$$f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

Let's start with the easy part: ∇_θ

- Note that the first parameter only appears **inside** the expectation

$$f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

- Therefore, if g and π are nice enough (e.g. when $\nabla_\theta g$ can be dominated), we can use Leibniz's integral rule to get

$$\nabla_\theta f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[\nabla_\theta g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

Let's start with the easy part: ∇_θ

- Note that the first parameter only appears **inside** the expectation

$$f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

- Therefore, if g and π are nice enough (e.g. when $\nabla_\theta g$ can be dominated), we can use Leibniz's integral rule to get

$$\nabla_\theta f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[\nabla_\theta g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

which means that we can get an unbiased estimate of $\nabla_\theta f(\boldsymbol{\theta}, \boldsymbol{\gamma})$ by simply sampling $\mathbf{w}_1, \ldots, \mathbf{w}_K \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})$ and then computing

$$\nabla_\theta f(\boldsymbol{\theta}, \boldsymbol{\gamma}) \approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\theta g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}_k)$$

Let's start with the easy part: ∇_θ

- Note that the first parameter only appears **inside** the expectation

$$f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

- Therefore, if g and π are nice enough (e.g. when $\nabla_\theta g$ can be dominated), we can use Leibniz's integral rule to get

$$\nabla_\theta f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[\nabla_\theta g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

which means that we can get an unbiased estimate of $\nabla_\theta f(\boldsymbol{\theta}, \boldsymbol{\gamma})$ by simply sampling $\mathbf{w}_1, \ldots, \mathbf{w}_K \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})$ and then computing

$$\nabla_\theta f(\boldsymbol{\theta}, \boldsymbol{\gamma}) \approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\theta g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}_k)$$

Often *K*=1 will be enough!

Now the tricky part: ∇_γ

- This parameter appears both **inside** and **outside** the expectation:

$$f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

Now the tricky part: ∇_γ

- This parameter appears both **inside** and **outside** the expectation:

$$f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

- So it's not that simple… Let us write the expectation:

$$f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$$

Now the tricky part: ∇_γ

- This parameter appears both **inside** and **outside** the expectation:

$$f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}(\mathbf{w})} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right]$$

- So it's not that simple… Let us write the expectation:

$$f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$$

- **It makes sense to use Leibniz's rule again:**

$$\nabla_\gamma f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) \right] d\mathbf{w}$$

Now the tricky part: ∇_γ

- It makes sense to use Leibniz's rule again:

$$\nabla_\gamma f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) \right] d\mathbf{w}$$

And now we can use the usual rule to **differentiate a product**:

$$\nabla_\gamma f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right] \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w} + \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w}) \right] d\mathbf{w}$$

Now the tricky part: ∇_γ

- It makes sense to use Leibniz's rule again:

$$\nabla_\gamma f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) \right] d\mathbf{w}$$

And now we can use the usual rule to **differentiate a product**:

$$\nabla_\gamma f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right] \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w} + \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w}) \right] d\mathbf{w}$$

- **Which one of these two terms is easy to unbiasedly estimate?**

Now the tricky part: ∇_γ

- It makes sense to use Leibniz's rule again:

$$\nabla_\gamma f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) \right] d\mathbf{w}$$

And now we can use the usual rule to **differentiate a product**:

$$\nabla_\gamma f(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right] \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w} + \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w}) \right] d\mathbf{w}$$

- **Which one of these two terms is easy to unbiasedly estimate? The first one because it is an expectation!**

$$\int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \right] \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w} \approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}_k) \right]$$

Now the tricky part: ∇_γ

- The real tricky part is what's left: the second term $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w}$

that is not an expected value!

Now the tricky part: ∇_γ

- The real tricky part is what's left: the second term $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w}$

that is not an expected value!

- So, it's not an expected value, but we can turn it into one by dividing/multiplying!

$$\int \left(g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \frac{\nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]}{\pi_{\boldsymbol{\gamma}}(\mathbf{w})} \right) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$$

Now the tricky part: ∇_γ

- The real tricky part is what's left: the second term $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w}$

that is not an expected value!

- So, it's not an expected value, but we can turn it into one by dividing/multiplying!

$$\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \frac{\nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]}{\pi_{\boldsymbol{\gamma}}(\mathbf{w})} \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$$

Now, if we also remark that $\frac{\nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]}{\pi_{\boldsymbol{\gamma}}(\mathbf{w})} = \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})$, we finally get

Now the tricky part: ∇_γ

- The real tricky part is what's left: the second term $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w}$

that is not an expected value!

- So, it's not an expected value, but we can turn it into one by dividing/multiplying!

$$\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \frac{\nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]}{\pi_{\boldsymbol{\gamma}}(\mathbf{w})} \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$$

Now, if we also remark that $\frac{\nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]}{\pi_{\boldsymbol{\gamma}}(\mathbf{w})} = \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})$, we finally get

$$\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w} = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]$$

$$\approx \frac{1}{K} \sum_{k=1}^{K} g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}_k) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w}_k)$$

Now the tricky part: ∇_γ

- The estimate $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w} = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]$

$$\approx \frac{1}{K} \sum_{k=1}^{K} g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}_k) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w}_k)$$

is called the **score function gradient**, or the **REINFORCE** estimate.

Now the tricky part: ∇_γ

- The estimate $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w} = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]$

$$\approx \frac{1}{K} \sum_{k=1}^{K} g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}_k) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w}_k)$$

is called the **score function gradient**, or the **REINFORCE** estimate.

- One big issue is that it can have **potentially very large variance**, and typically requires a lot of samples to be accurate.

Now the tricky part: ∇_γ

- The estimate $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w} = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]$

$$\approx \frac{1}{K} \sum_{k=1}^{K} g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}_k) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w}_k)$$

is called the **score function gradient**, or the **REINFORCE** estimate.

- One big issue is that it can have **potentially very large variance**, and typically requires a lot of samples to be accurate.

- Can we do better? In general, not really. **But in some specific cases, yes!**

Now the tricky part: ∇_γ

- The estimate $\int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \left[\pi_{\boldsymbol{\gamma}}(\mathbf{w})\right] d\mathbf{w} = \mathbb{E}_{\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}} \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w})\right]$

$$\approx \frac{1}{K} \sum_{k=1}^{K} g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}_k) \nabla_\gamma \log \pi_{\boldsymbol{\gamma}}(\mathbf{w}_k)$$

is called the **score function gradient**, or the **REINFORCE** estimate.

- One big issue is that it can have **potentially very large variance**, and typically requires a lot of samples to be accurate.

- Can we do better? In general, not really. **But in some specific cases, yes!**

- In the next slide, we'll see one of such cases: **the Gaussian reparametrisation trick.**

Reparametrisation trick for ∇_γ

- Our goal is still to estimate our tricky term $\nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$

- Again, the main issue is that the density depends on the parameter of interest. Can we destroy this dependence? Can we **push γ away from the density** we're integrating against?

Reparametrisation trick for ∇_γ

- Our goal is still to estimate our tricky term $\nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$

- Again, the main issue is that the density depends on the parameter of interest. Can we destroy this dependence? Can we **push γ away from the density** we're integrating against?

- In some important cases, **yes!** The main example is the **Gaussian case** $\pi_{\boldsymbol{\gamma}} = \mathcal{N}(\mu_{\boldsymbol{\gamma}}, \Sigma_{\boldsymbol{\gamma}})$

Reparametrisation trick for ∇_γ

- Our goal is still to estimate our tricky term $\nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w}$

- Again, the main issue is that the density depends on the parameter of interest. Can we destroy this dependence? Can we **push γ away from the density** we're integrating against?

- In some important cases, **yes!** The main example is the **Gaussian case** $\pi_{\boldsymbol{\gamma}} = \mathcal{N}(\mu_{\boldsymbol{\gamma}}, \Sigma_{\boldsymbol{\gamma}})$

- In this setting, it is clear that sampling $\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}$ can be done by computing

$$\mathbf{w} = \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}$$

where $C_{\boldsymbol{\gamma}}$ is the Cholesky decomposition of the covariance, and $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$

Indeed, $C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \Sigma_{\boldsymbol{\gamma}})$ (linear change of variable).

Reparametrisation trick for ∇_γ

- Sampling $\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}$ can be done by computing $$\mathbf{w} = \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}$$

where $C_{\boldsymbol{\gamma}}$ is the Cholesky decomposition of the covariance, and $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I})$.

Reparametrisation trick for ∇_γ

- Sampling $\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}$ can be done by computing $$\mathbf{w} = \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}$$

where $C_{\boldsymbol{\gamma}}$ is the Cholesky decomposition of the covariance, and $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I})$.

- But now, the only random thing is $\boldsymbol{\varepsilon}$. This means we can rewrite our expectation as an expectation over $\boldsymbol{\varepsilon}$

$$\nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w} = \nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}) p(\boldsymbol{\varepsilon}) d\boldsymbol{\varepsilon}$$

Reparametrisation trick for ∇_γ

- Sampling $\mathbf{w} \sim \pi_{\boldsymbol{\gamma}}$ can be done by computing $$\mathbf{w} = \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}$$

where $C_{\boldsymbol{\gamma}}$ is the Cholesky decomposition of the covariance, and $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$.

- But now, the only random thing is ε. This means we can rewrite our expectation as an expectation over ε

$$\nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w} = \nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}) p(\boldsymbol{\varepsilon}) d\boldsymbol{\varepsilon}$$

…and finally use Leibniz's rule

$$\nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w} = \int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}) \right] p(\boldsymbol{\varepsilon}) d\boldsymbol{\varepsilon}$$

$$\approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}_k) \right]$$

Reparametrisation trick for ∇_γ

- The estimate

$$\nabla_\gamma \int g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mathbf{w}) \pi_{\boldsymbol{\gamma}}(\mathbf{w}) d\mathbf{w} = \int \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}) \right] p(\boldsymbol{\varepsilon}) d\boldsymbol{\varepsilon}$$

$$\approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\gamma \left[g(\boldsymbol{\theta}, \boldsymbol{\gamma}, \mu_{\boldsymbol{\gamma}} + C_{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}_k) \right]$$

is often called the **reparametrisation trick** estimate. It has considerably less variance in practice than the score gradient, but can be less generally applied.

- Beyond Gaussians, this can be done for more complex distributions (Dirichlet, Student's t, GMMs), but this is not easy, in particular for **discrete distributions.**

- It is automatically implemented in many libraries, for instance **Tensorflow Probability**, or **Pytorch distributions**

A quick summary of VAEs/IWAEs so far

- We have defined a graphical model called a **deep latent variable model**

- We have seen how to train this model by doing **approximate maximum likelihood via amortised variational inference**

- We saw that there was an **important interplay between this inference technique, and various sampling techniques** (importance sampling to define the bounds, various methods to estimate its gradients without bias).

2

On d-separation and latent variable models

D-separation

- D-separation means **directed separation.**

- It's a general framework for answering questions à la « $X_A \perp\!\!\!\perp X_B | X_C ?$ » where A, B, C are three subsets of the set of nodes.

D-separation

- D-separation means **directed separation.**

- It's a general framework for answering questions à la « $X_A \perp\!\!\!\perp X_B | X_C$? » where A, B, C are three subsets of the set of nodes.

- **D-separation recipe:** we consider all chains between any node in A and any node in B. Any of these chains is said to be **blocked by *C*** if it includes a node such that either
  - the chain is a v-structure at the node, and neither the node, nor its descendants, are in C
  - the arrows on the chain meet either head-to-tail or tail-to-tail at the node, and the node belongs to C

D-separation

- D-separation means **directed separation.**

- It's a general framework for answering questions à la « $X_A \perp\!\!\!\perp X_B | X_C$? » where A, B, C are three subsets of the set of nodes.

- **D-separation recipe:** we consider all chains between any node in A and any node in B. Any of these chains is said to be **blocked by *C*** if it includes a node such that either
  - the chain is a v-structure at the node, and neither the node, nor its descendants, are in C
  - the arrows on the chain meet either head-to-tail or tail-to-tail at the node, and the node belongs to C

Definition. *We say that A **and** B **are d-separated by** C if all chains between A and B are blocked by C.*

Properties of d-separation

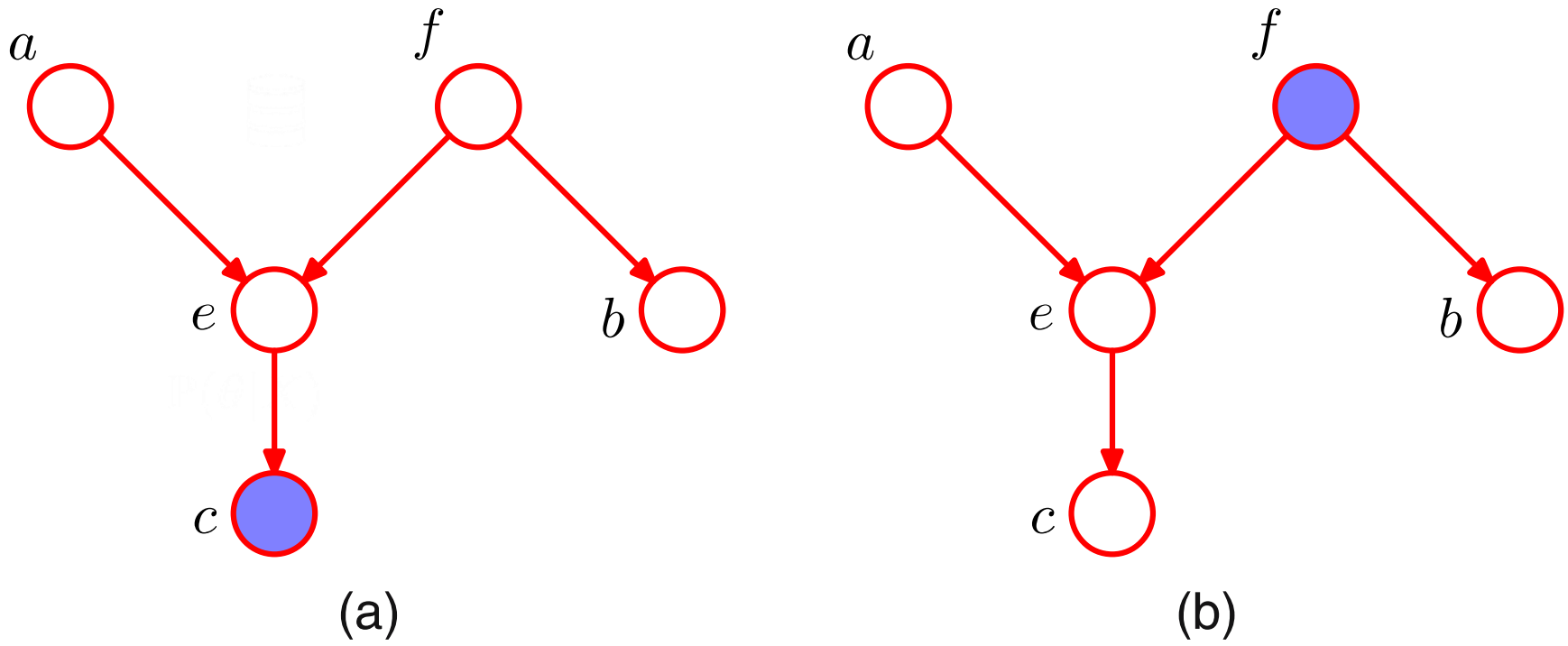
Theorem (soundness of d-separation)**.** *If A and B are d-separated by C and $p \in \mathcal{L}(G)$, then $X_A \perp\!\!\!\perp X_B | X_C$.*

Theorem (completeness of d-separation)**.** *If A and B are not d-separated by C, then there exist $p \in \mathcal{L}(G)$ such that $X_A \not\perp\!\!\!\perp X_B | X_C$.*

For more details, including proofs, see [PGM, Sec. 3.3.2].

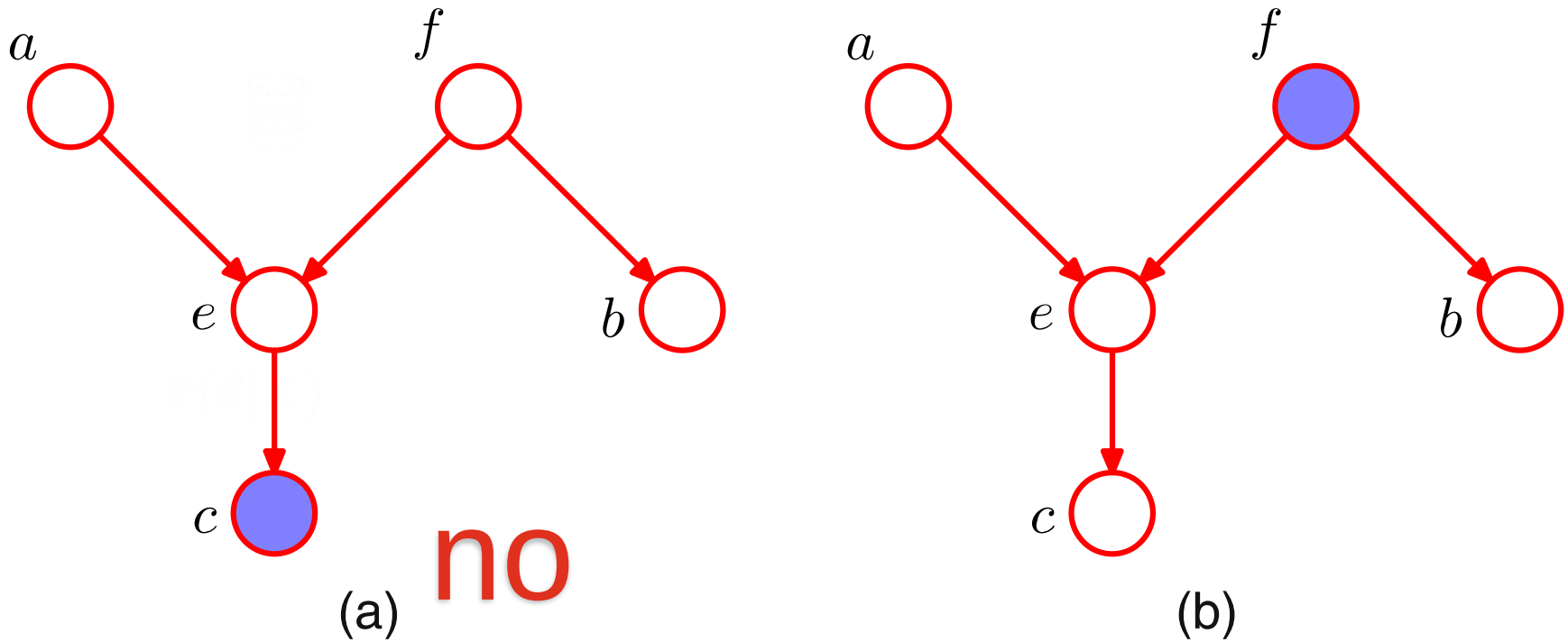
Examples of d-separation

- I took this figure from [Bishop, Fig. 8.22]. Does the blue node d-separate a and b?



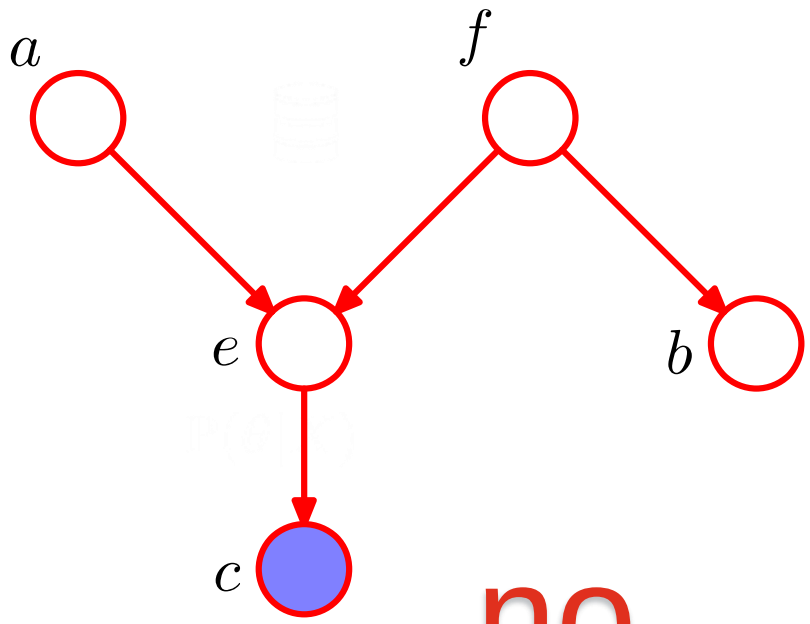
Examples of d-separation

- I took this figure from [Bishop, Fig. 8.22]. Does the blue node d-separate a and b?

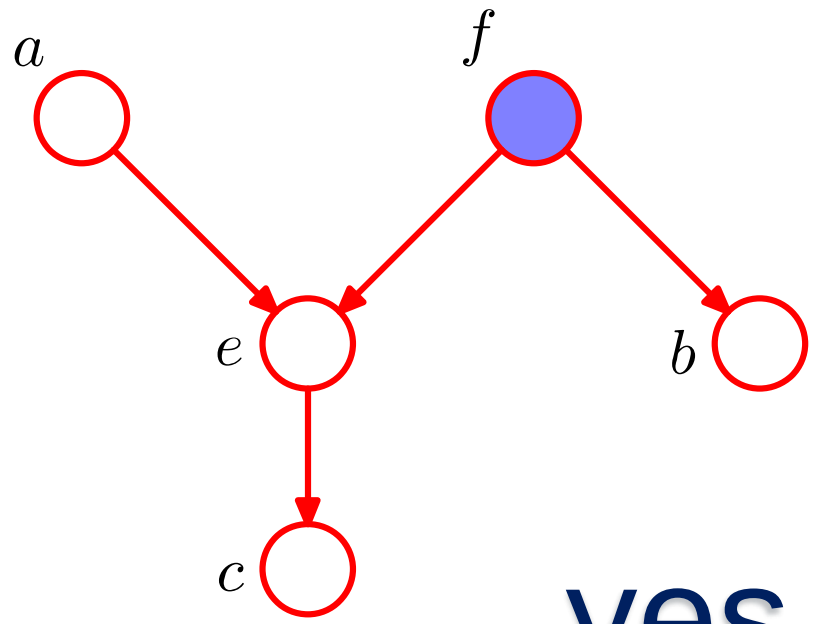


Examples of d-separation

- I took this figure from [Bishop, Fig. 8.22]. Does the blue node d-separate a and b?



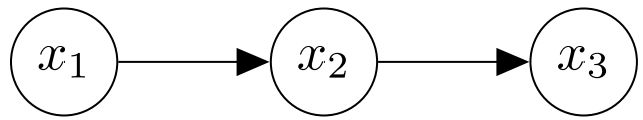
(a) no



(b) yes

Applications of d-separation

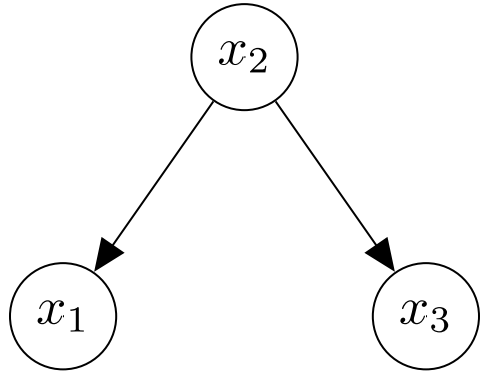
- We previously mentioned conditional independence properties. Try to prove them using d-separation:



$$x_3 \perp\!\!\!\perp x_1 | x_2$$

Applications of d-separation

- We previously mentioned conditional independence properties. Try to prove them using d-separation:



$$x_3 \perp\!\!\!\perp x_1 | x_2$$

3

From VAEs to diffusion models

From VAEs to hierarchical VAEs

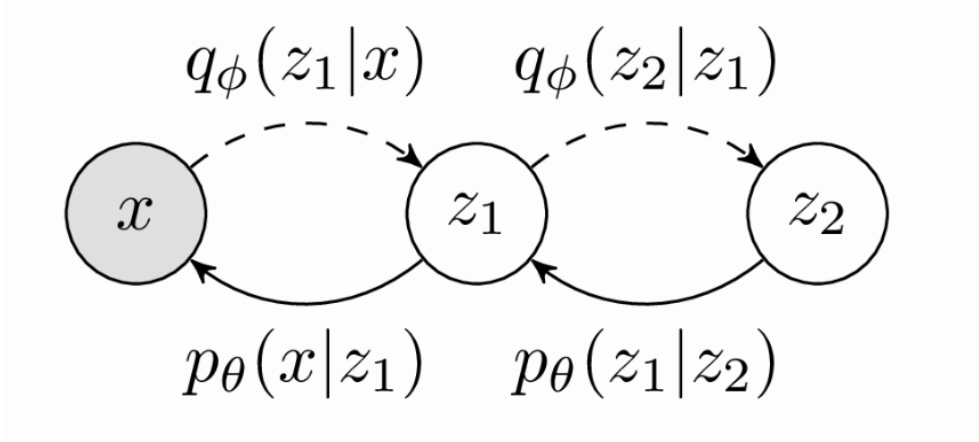
- One way to apply the good old « deeper is better » motto to VAEs is to make them **hierarchical** (aka deep, aka laddered).

- The idea is simply to make **a VAE within a VAE** by choosing the VAE prior to be itself a deep latent variable model.

From VAEs to hierarchical VAEs

- One way to apply the good old « deeper is better » motto to VAEs is to make them **hierarchical** (aka deep, aka laddered).

- The idea is simply to make **a VAE within a VAE** by choosing the VAE prior to be itself a deep latent variable model.

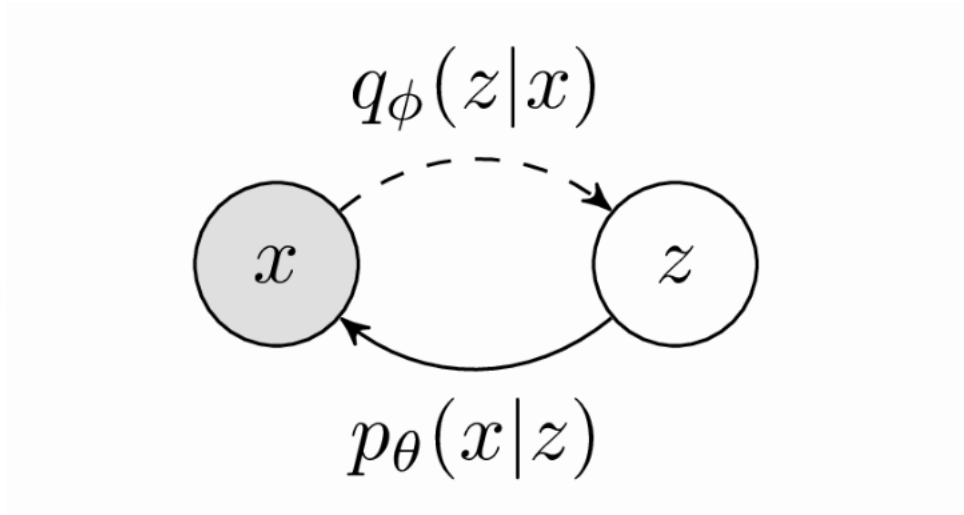


- One can then derive likelihood bounds in the exact same way.

Fig. from Angus Turner's blog

On DAGs and VAEs

- In a standard VAE, there are **two DAGs:** the generative model itself, and the inference model (that we essentially only need for training).

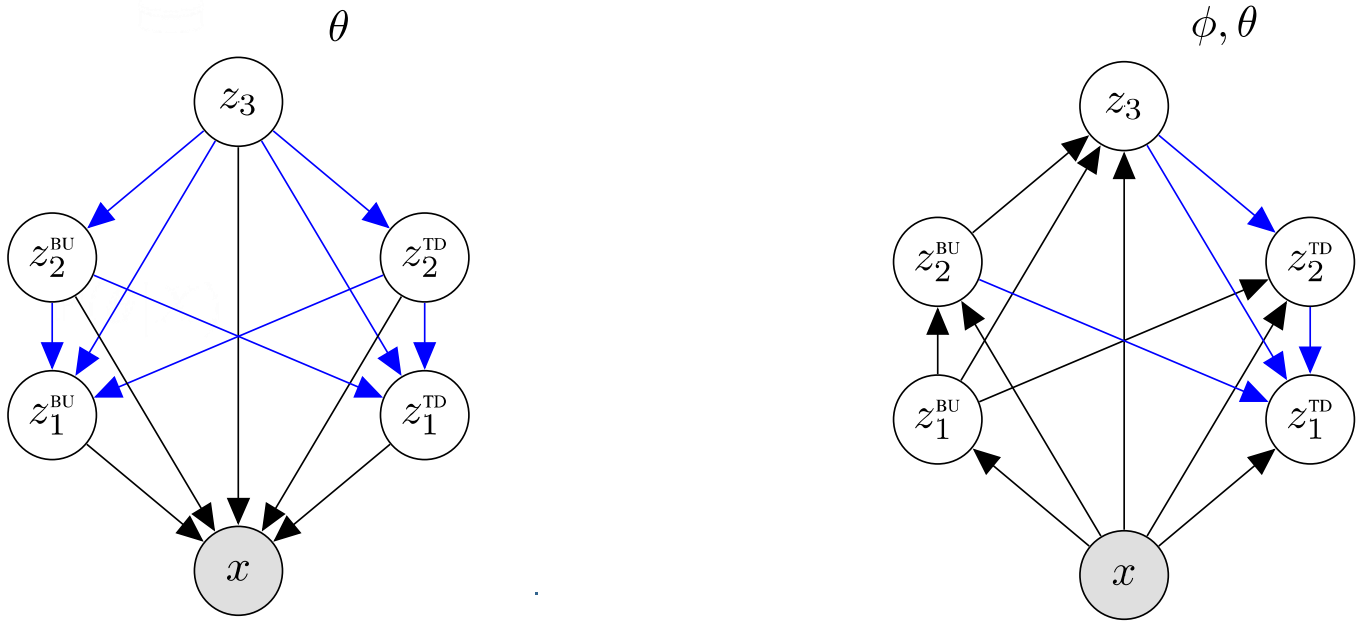


- Since there are only two nodes, there are not many ways of defining DAGs…

Fig. from Angus Turner's blog.

On DAGs and VAEs

- In hierarchical VAEs, there will be two DAGs as well. The generative model is usually the usual « a VAE inside a VAE » one, but since there are more than two nodes, there are many ways of building the inference DAG.

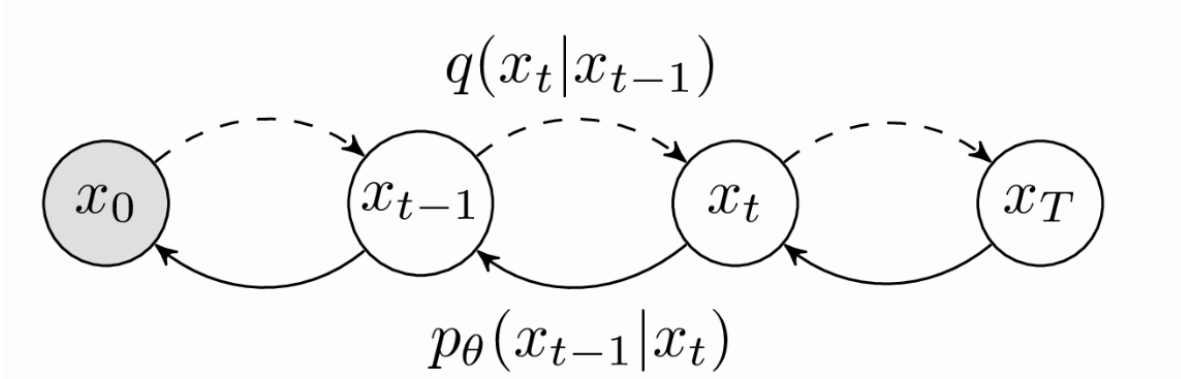


(a) Generative model (b) Inference model

Fig. from Maaløe, Fraccaro, Liévin, and Winther (NeurIPS 2019)

Diffusion models as hierarchical VAEs

- Diffusion models can be seen as a particular kind a hierarchical VAE, with a peculiar stucture:



- ➢ The dimension of all latent spaces are equal to the dimension of the data
- ➢ The **inference model is fixed, and is just adding noise**

$$q(x_t|x_{t-1}) = \mathcal{N}(x_T\ ;\ x_{t-1}\sqrt{1-\beta_t}, \beta_t I)$$

such that

- ➢ The only thing we really model is $q(x_T|x_0) \approx \mathcal{N}(0, I)$, wh**oise the noisy latent variables.** It is generally parametrised as a U-Net.

Fig. from Angus Turner's blog.

Diffusion models as hierarchical VAEs

- For more details on this, see
  - **Angus Turner's blog** https://angusturner.github.io/generative_models/2021/06/29/diffusion-probabilistic-models-I.html
  - **Understanding Diffusion Models: A Unified Perspective**, arXiv preprint from Calvin Luo

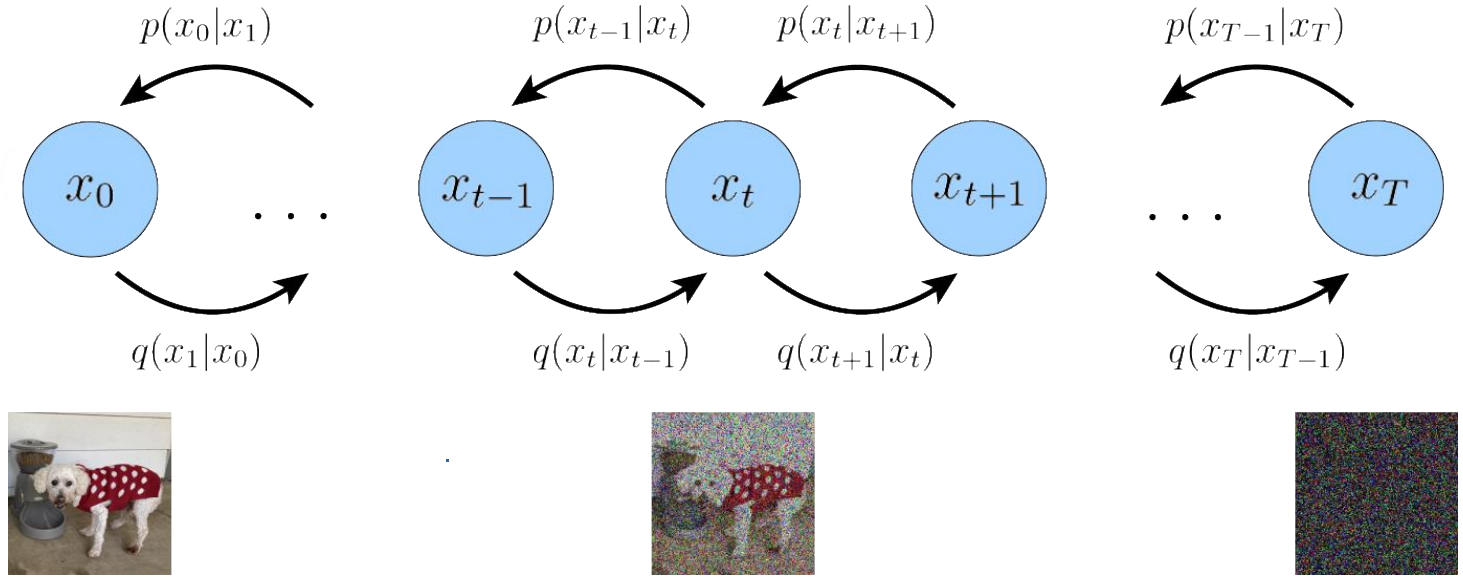


Fig. from Luo (2022)

Diffusion models are not just VAEs

- There are also related to
  - **Score matching**, see e.g. Yang Song's blog https://yang-song.net/blog/2021/score/
  - **Another nice blog post:** https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

4

Deep generative models with exact likelihood

Can we force a DLVM to have an exact likelihood?

- By exact likelihood, I simply mean that I can **compute easily the log-likelihood and its gradients.**

- Can we build DLVM-like models with exact likelihood?

Can we force a DLVM to have an exact likelihood?

- By exact likelihood, I simply mean that I can **compute easily the log-likelihood and its gradients.**

- Can we build DLVM-like models with exact likelihood?

  - Well, that the case of **linear DLVMs** like factor analysis or PPCA, but that's not deep…

Can we force a DLVM to have an exact likelihood?

- By exact likelihood, I simply mean that I can **compute easily the log-likelihood and its gradients.**

- Can we build DLVM-like models with exact likelihood?

  - Well, that the case of **linear DLVMs** like factor analysis or PPCA, but that's not deep…

  - A powerful idea is to **force the decoder to be invertible**, leading to a family of models called **normalising flows**

Why is it beneficial to have an invertible decoder?

- **Normalising flows** consider the following model:

$$\mathbf{z} \sim p(\mathbf{z}), \ \mathbf{x} = f_\theta(\mathbf{z})$$

Simple « prior », typically Gaussian

Invertible nonlinear decoder

Why is it beneficial to have an invertible decoder?

- **Normalising flows** consider the following model:

$$\mathbf{z} \sim p(\mathbf{z}), \ \mathbf{x} = f_\theta(\mathbf{z})$$

Simple « prior », typically Gaussian

Invertible nonlinear decoder

- A key difference with DLVMs is that here, **if you know *z*, you know *x* deterministically**

Why is it beneficial to have an invertible decoder?

- **Normalising flows** consider the following model:

$$\mathbf{z} \sim p(\mathbf{z}), \;\; \mathbf{x} = f_\theta(\mathbf{z})$$

Simple « prior », typically Gaussian

Invertible nonlinear decoder

- A key difference with DLVMs is that here, **if you know *z*, you know *x* deterministically** and the converse, since

$$\mathbf{z} = f_\theta^{-1}(\mathbf{x})$$

Why is it beneficial to have an invertible decoder?

- **Normalising flows** consider the following model:

$$\mathbf{z} \sim p(\mathbf{z}), \;\; \mathbf{x} = f_\theta(\mathbf{z})$$

Simple « prior », typically Gaussian

Invertible nonlinear decoder

- A key difference with DLVMs is that here, **if you know *z*, you know *x* deterministically** and the converse, since

$$\mathbf{z} = f_\theta^{-1}(\mathbf{x})$$

So no « posterior »…

Why is it beneficial to have an invertible decoder?

- The key idea is that, since the decoder is invertible, we can use the **change of variable formula:** $p(\mathbf{x}) = p(\mathbf{z}) \det(\partial f_\theta / \partial \mathbf{z})^{-1}$

- **This leads to an exact log-likelihood**

$$\log p_\theta(\mathbf{x}) = \log p(\mathbf{z}) - \log \det \frac{\partial f_\theta}{\partial \mathbf{z}}$$

- **Key problem: how do we design invertible but flexible functions?**

How do we design invertible layers?

- Essentially two big families of recipes

1. Coupling layers divide the output *y* in two parts:

Any sort of neural net

$$\mathbf{y}_{1:d} = \mathbf{x}_{1:d}$$
$$\mathbf{y}_{(d+1):D} = \mathbf{x}_{(d+1):D} \odot \exp(s_\theta(\mathbf{x}_{1:d})) + t_\theta(\mathbf{x}_{1:d})$$

How do we design invertible layers?

- Essentially two big families of recipes

1. Coupling layers divide the output *y* in two parts:

Any sort of neural net

$$\mathbf{y}_{1:d} = \mathbf{x}_{1:d}$$
$$\mathbf{y}_{(d+1):D} = \mathbf{x}_{(d+1):D} \odot \exp(s_\theta(\mathbf{x}_{1:d})) + t_\theta(\mathbf{x}_{1:d})$$

It is actually easy to invert such a peculiar layer, indeed, I can go from *x* to *y* by a similar layer:

$$\mathbf{x}_{1:d} = \mathbf{y}_{1:d}$$
$$\mathbf{x}_{(d+1):D} = (\mathbf{y}_{(d+1):D} - t_\theta(\mathbf{y}_{1:d})) \odot \exp(-s_\theta(\mathbf{y}_{1:d}))$$

How do we design invertible layers?

- Essentially two big families of recipes

DENSITY ESTIMATION USING REAL NVP

Laurent Dinh*
Montreal Institute for Learning Algorithms
University of Montreal
Montreal, QC H3T1J4

Jascha Sohl-Dickstein
Google Brain

Samy Bengio
Google Brain

1. Coupling layers divide the output *y* in two parts:

Any sort of neural net

$$\mathbf{y}_{1:d} = \mathbf{x}_{1:d}$$
$$\mathbf{y}_{(d+1):D} = \mathbf{x}_{(d+1):D} \odot \exp(s_\theta(\mathbf{x}_{1:d})) + t_\theta(\mathbf{x}_{1:d})$$

It is actually easy to invert such a peculiar layer, indeed, I can go from *x* to *y* by a similar layer:

$$\mathbf{x}_{1:d} = \mathbf{y}_{1:d}$$
$$\mathbf{x}_{(d+1):D} = (\mathbf{y}_{(d+1):D} - t_\theta(\mathbf{y}_{1:d})) \odot \exp(-s_\theta(\mathbf{y}_{1:d}))$$

The Jacobian determinant can also be computed relatively easily, as it is a triangular matrix.

How do we design invertible layers?

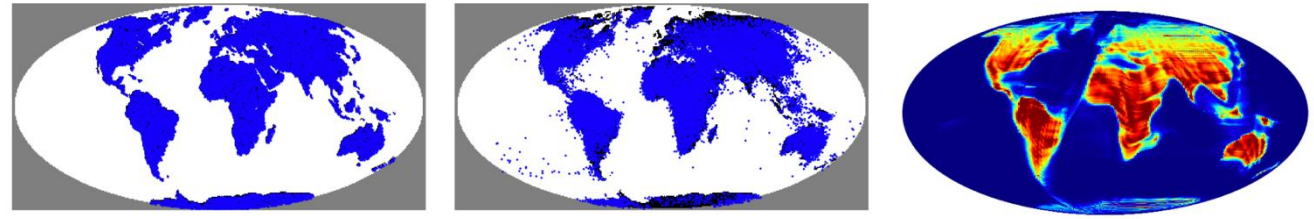
2. Using residual nets and Lipschitzianity

Residual layers generally look like $\mathbf{x}_{l+1} = \mathbf{x}_l + h_\theta(\mathbf{x}_l)$. A sufficient condition for this to be invertible is when h_θ is 1-Lipschitz.

This Lipschitzianity can be implemented using a variety of regularisation techniques. At the end, the inverse is not available in closed-form, but estimators exist.

Some normalising flows successes

Kingma & Dhariwal,
NeurIPS 2019



Normalizing Flows on Tori and Spheres

Danilo Jimenez Rezende [*1] George Papamakarios [*1] Sébastien Racanière [*1] Michael S. Albergo [2]
Gurtej Kanwar [3] Phiala E. Shanahan [3] Kyle Cranmer [2]

Are there other exact likelihood deep generative models?

- Another right avenue is to leverage the good old chain rule

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

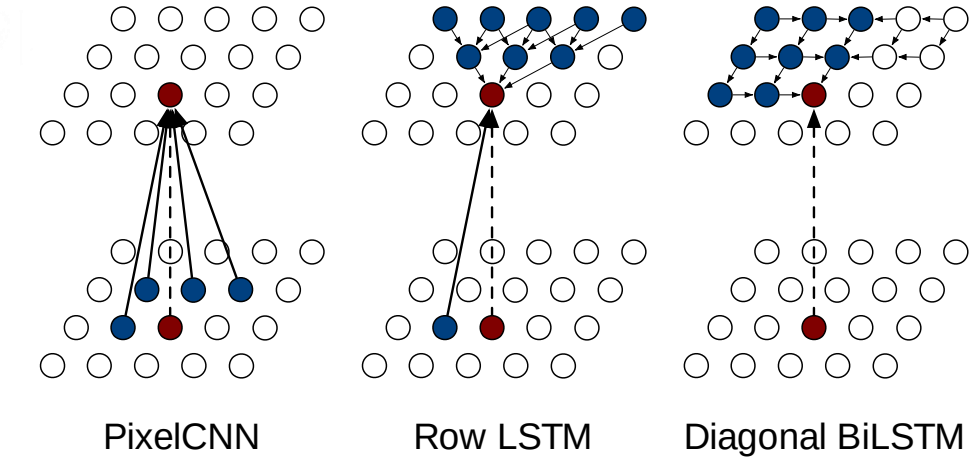
and to **parametrise each factor using a neural net**.

Are there other exact likelihood deep generative models?

- Another right avenue is to leverage the good old chain rule

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

and to **parametrise each factor using a neural net**. The issue is that we have to deal with variable-length inputs… One simple solution to this is for instance to use masks or RNNs.



(van den Oord et al., ICML 2016)

Are there other exact likelihood deep generative models?

- Another right avenue is to leverage the good old chain rule

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

and to **parametrise each factor using a neural net**. The issue is that we have to deal with variable-length inputs… One simple solution to this is for instance to use masks or RNNs.

Figure 1. Image completions sampled from a PixelRNN.

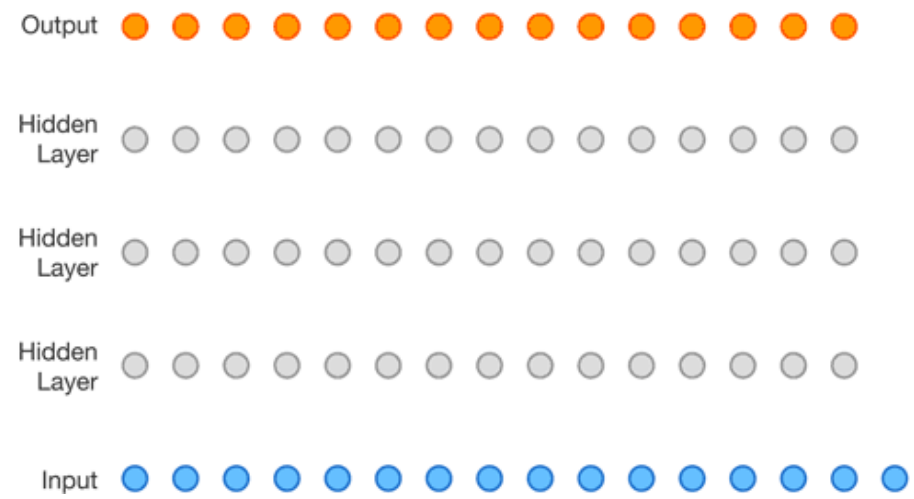
(van den Oord et al., ICML 2016)

Are there other exact likelihood deep generative models?

- Another right avenue is to leverage the good old chain rule

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

and to **parametrise each factor using a neural net**. The issue is that we have to deal with variable-length inputs… One simple solution to this is for instance to use masks or RNNs.



CNN architecture from van den Oord et al. (WaveNet: A Generative Model for Raw Audio, 2016)