VI masterclass: VAE and Normalising Flows

Sanmitra Ghosh

November 9, 2022

MRC Biostatistics Unit,
University of Cambridge
sanmitra.ghosh@mrc-bsu.cam.ac.uk

VAE

Introduction

This talk is based on two papers

- "An Introduction to Variational Autoencoders" DP Kingma et al., 2019.
- "Normalizing Flows for Probabilistic Modeling and Inference" by George Papamakarios et al., 2021

Applications: Image resynthesis

Figure 4.4: VAEs can be used for image resynthesis. In this example by White, 2016, an original image (left) is modified in a latent space in the direction of a *smile vector*, producing a range of versions of the original, from smiling to sadness.

Applications: Text generation

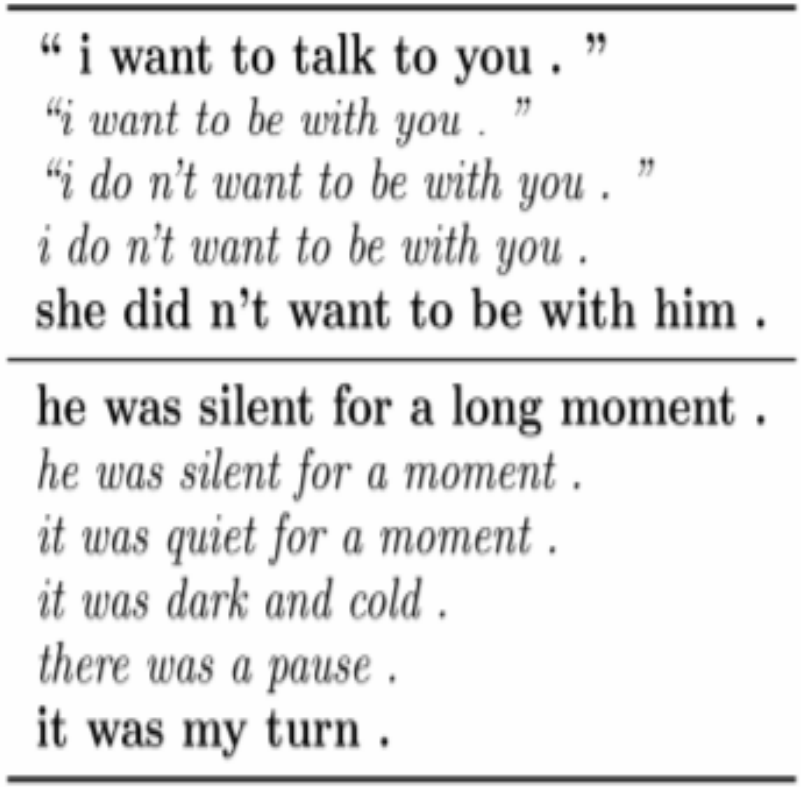


Figure 4.3: An application of VAEs to interpolation between pairs of sentences, from (Bowman *et al.*, 2015). The intermediate sentences are grammatically correct, and the topic and syntactic structure are typically locally consistent.

Unsupervised learning

A lot of machine learning is concerned with *supervised learning*:

$$p_\theta(\boldsymbol{y}|\boldsymbol{x})$$

where $\boldsymbol{y}$ is a label (response) and $\boldsymbol{x}$ are the features (covariates). Example: $\boldsymbol{x}$ can be a set of images while $\boldsymbol{y}$ tags the objects in each of that image.

However, large amount of data is unlabelled! What we want is

$$p_\theta(\boldsymbol{x}).$$

Variational Autoencoder (VAE) is a model + inference technique that is designed to learn a flexible $p_\theta(\cdot)$ using large amounts of data.

This task is known as *unsupervised learning* in machine learning.

Parameterizing Conditional Distributions with Neural Networks

Example: In case of neural network based image classification neural networks parameterize a categorical distribution $p_{\theta}(\boldsymbol{y}|\boldsymbol{x})$, over a class labels $\boldsymbol{y}$ conditioned on an image $\boldsymbol{x}$:

$$\boldsymbol{p} = \boldsymbol{f}_{\theta}(\boldsymbol{x})$$
$$p_{\theta}(\boldsymbol{y}|\boldsymbol{x}) = \text{Categorical}(\boldsymbol{y}; \boldsymbol{p})$$

You use this type of models all the time. Think GLMs.

- $\boldsymbol{f}_{\theta}$ is a Neural Network.

For the rest of the talk, it suffices to understand $\boldsymbol{f}_{\theta}$ simply as a differentiable nonlinear transformation of its input

Graphical models

Consider a DAG. We can factorise the joint distribution of the variables as:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M) = \prod_{j=1}^{M} p_{\boldsymbol{\theta}}(\boldsymbol{x}_j | Pa(\boldsymbol{x}_j)),$$

where $Pa(\boldsymbol{x}_j)$ is the set of parent variables of node j.

A more flexible way to parameterise such conditional distributions is with neural networks:

$$\boldsymbol{\eta} = \boldsymbol{f}_{\boldsymbol{\theta}}(Pa(\boldsymbol{x}))$$
$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_j | Pa(\boldsymbol{x}_j)) = p_{\boldsymbol{\theta}}(\boldsymbol{x}_j | \boldsymbol{\eta}).$$

Deep latent variable model

By introducing a latent variable $\boldsymbol{z}$ (the Code) we can come up with an implicit yet flexible distribution:

$$p_{\theta}(\boldsymbol{x}) = \int p_{\theta}(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z}, \tag{1}$$

which is simply the (single datapoint) marginal likelihood of the data, or the evidence, when taken as a function of $\boldsymbol{\theta}$. Now of course the joint distribution can be factorised as:

$$p_{\theta}(\boldsymbol{x}, \boldsymbol{z}) = p_{\theta}(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}).$$

By parameterising the conditional distribution with a deep neural neural network:

$$\begin{aligned} \boldsymbol{z} &\sim p(\boldsymbol{z}) \\ \boldsymbol{\eta} &= \boldsymbol{f}_{\theta}(\boldsymbol{z}) \\ p_{\theta}(\boldsymbol{x}|\boldsymbol{z}) &= p_{\theta}(\boldsymbol{x}|\boldsymbol{\eta}), \end{aligned} \tag{2}$$

we get what is known as a deep latent variable model (**DLVM**).

DLVM for multivariate Bernoulli data

$$
\begin{aligned}
\boldsymbol{z} &\sim \mathcal{N}(0, 1) \\
\boldsymbol{p} &= \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}) \\
\log p(\boldsymbol{x}|\boldsymbol{z}) &= \sum_{d=1}^{D} \log \text{Bern}(x_d; p_d) \\
&= \sum_{d=1}^{D} x_d \log p_D + (1 - x_d) \log(1 - p_d).
\end{aligned}
$$

- $\boldsymbol{f}_{\boldsymbol{\theta}}$ is known as the decoder.
- Decoder since it maps a latent Gaussian to a complex one.

.

Bayesian inference in Deep latent variable model

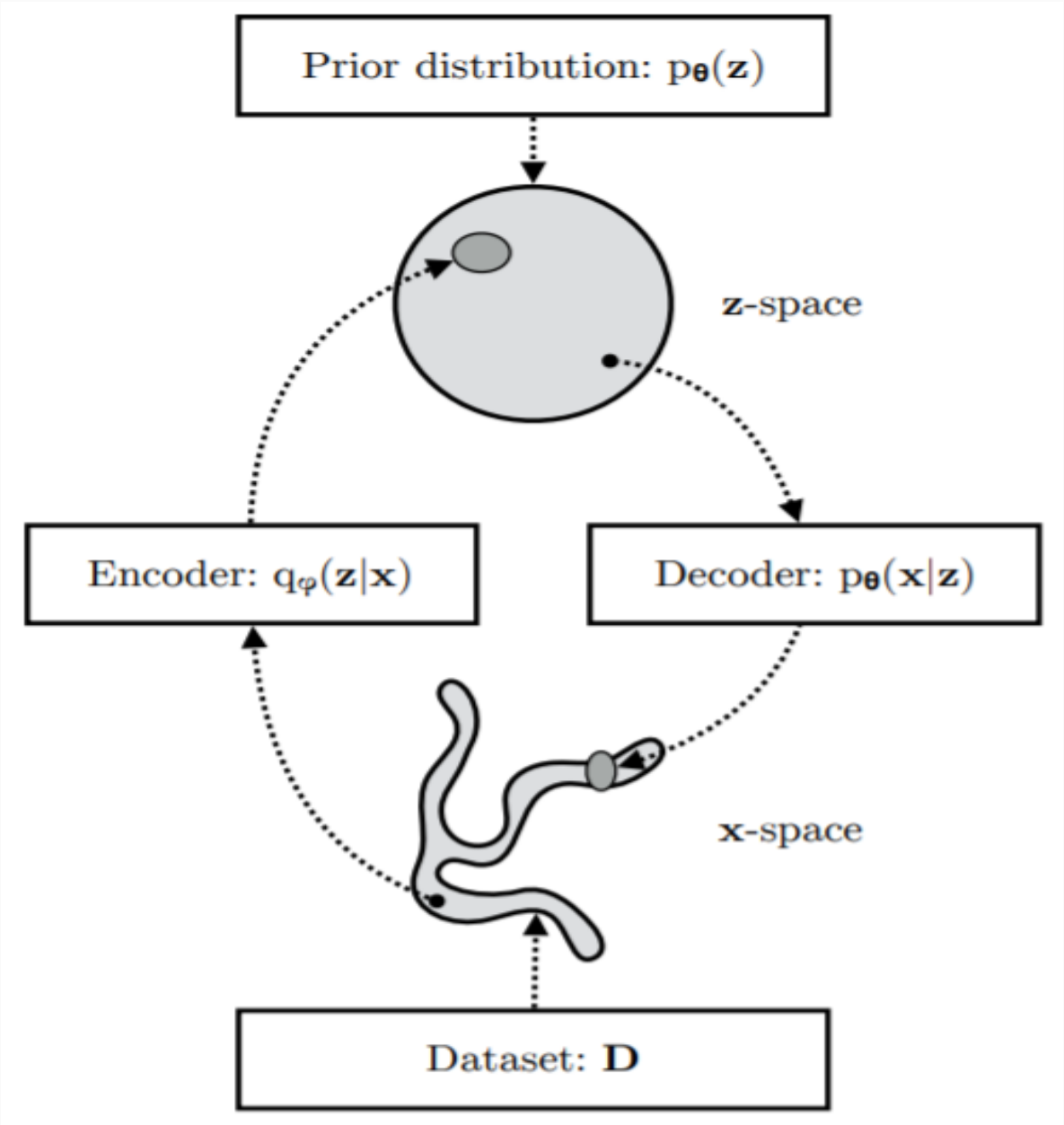
- If we know the posterior $p_{\theta}(\boldsymbol{z}|\boldsymbol{x})$, we can generate data from the DLVM.
- $p_{\theta}(\boldsymbol{z}|\boldsymbol{x})$ is intractable.
- We will apply VI.

$$q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) \approx p_{\theta}(\boldsymbol{z}|\boldsymbol{x}).$$

The task then is to find the appropriate parameter values $\phi, \boldsymbol{\theta}$ that minimises: $\mathrm{KL}(q_{\phi}||p_{\theta})$, or maximises the ELBO.

- Note the conditioning by $\boldsymbol{x}$ in the variational approximation.
- Problem: We will need as many parameters as datapoints.

Variational Autoencoder



Bayesian inference in Deep latent variable model

We optimise the evidence lower bound (ELBO):

$$\begin{aligned}\mathcal{L}_{\phi,\theta}(\boldsymbol{x}) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})] \\ &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_\phi||p_\theta) \\ &\leq \log p_\theta(\boldsymbol{x}),\end{aligned} \tag{3}$$

- Notice that the ELBO is now a function of both model and variational parameters, $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ resp.

Bayesian inference in DLVM

If we get the gradient of the ELBO wrt $\boldsymbol{\theta}, \boldsymbol{\phi}$ then these parameters can be updated using gradient ascent:

$$\begin{aligned} \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \gamma \nabla_{\boldsymbol{\theta}} \boldsymbol{\mathcal{L}}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{x}) \\ \boldsymbol{\phi} &\leftarrow \boldsymbol{\phi} + \gamma \nabla_{\boldsymbol{\phi}} \boldsymbol{\mathcal{L}}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{x}), \end{aligned} \tag{4}$$

the trouble is that the expectations in ELBO equation are intractable. In practise we use a Monte Carlo estimator$\hat{\nabla}_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{x})$ of the gradient of the ELBO, obtained using the reparameterisation trick.

- We will be using the **pathwise estimator****.

** [Glasserman 1991; Fu 2006; Kingma+ 2014; Rezende+ 2014; Titsias+ 2014]

Reparameterisation trick

Write $\boldsymbol{z} = g(\boldsymbol{\epsilon}, \boldsymbol{\phi}, \boldsymbol{x})$, i.e in terms of a differentiable transformation.

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[h(\boldsymbol{z})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[h(\boldsymbol{z})],$$

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[h(\boldsymbol{z})] = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{\epsilon})}[h(\boldsymbol{z})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\phi}} h(\boldsymbol{z})],$$

Reparameterisation trick

Amortised approximation:

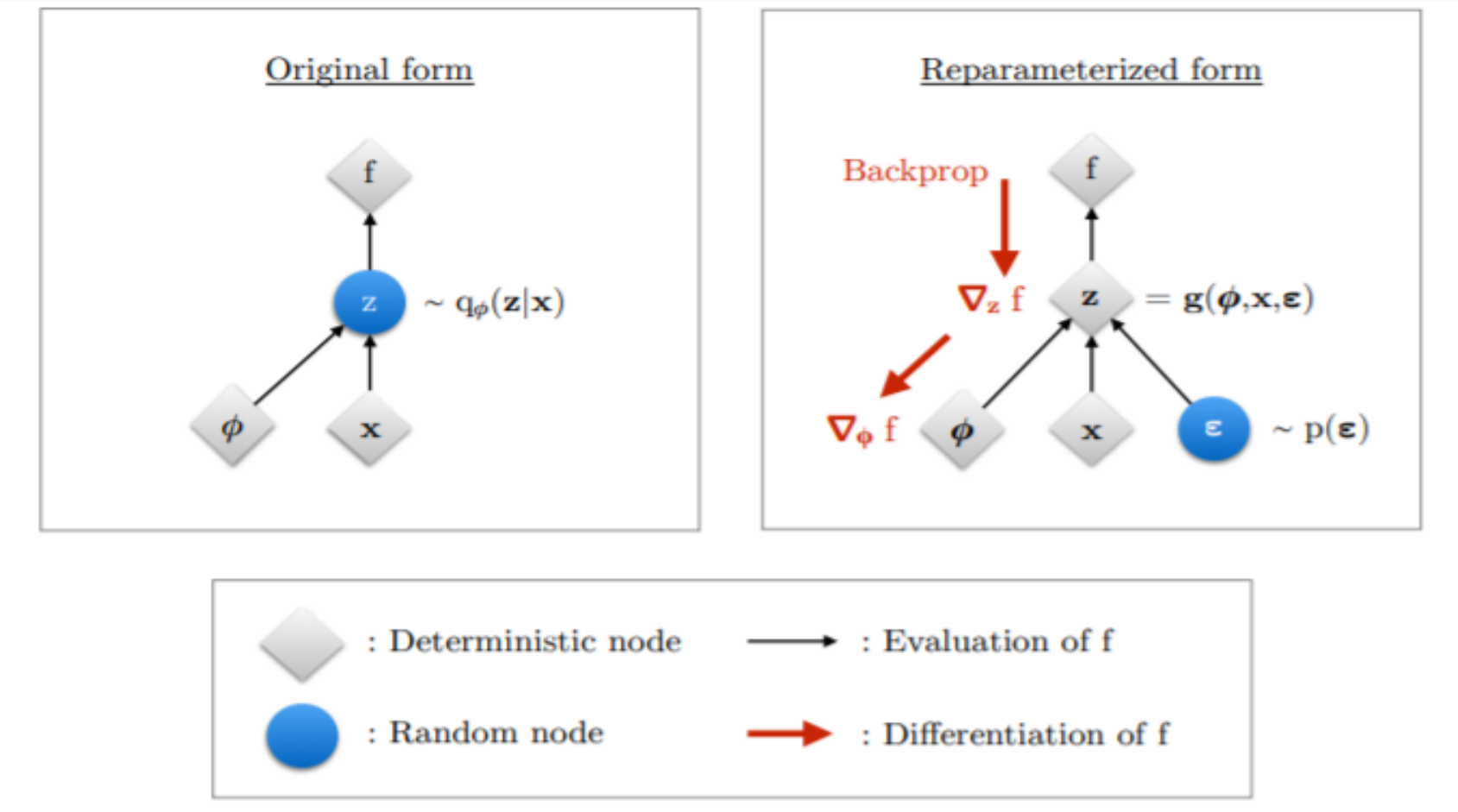
$$
\begin{aligned}
(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) &= \boldsymbol{f}_{\phi}(\boldsymbol{x}) \\
q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) &= \prod_{d} q_{\phi}(z_d|\boldsymbol{x}) = \prod_{d} \mathcal{N}(z_d; \mu_d, \sigma_d^2),
\end{aligned}
$$

which we can write using re-parameterisation as:

$$
\begin{aligned}
\boldsymbol{\epsilon} &\sim \mathcal{N}(0, 1) \\
(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) &= \boldsymbol{f}_{\phi}(\boldsymbol{x}) \\
\boldsymbol{z} &= \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}.
\end{aligned} \tag{5}
$$

- $\boldsymbol{f}_{\phi}$ in known as the Encoder, since it compresses information from the complex observation space to the latent space.
- You can use the same way of encoding through a full-rank Gaussian.
- This way of conditioning on the data $\boldsymbol{x}$ is known as amortisation.

Reparameterisation trick graph



Monte Carlo gradient of ELBO

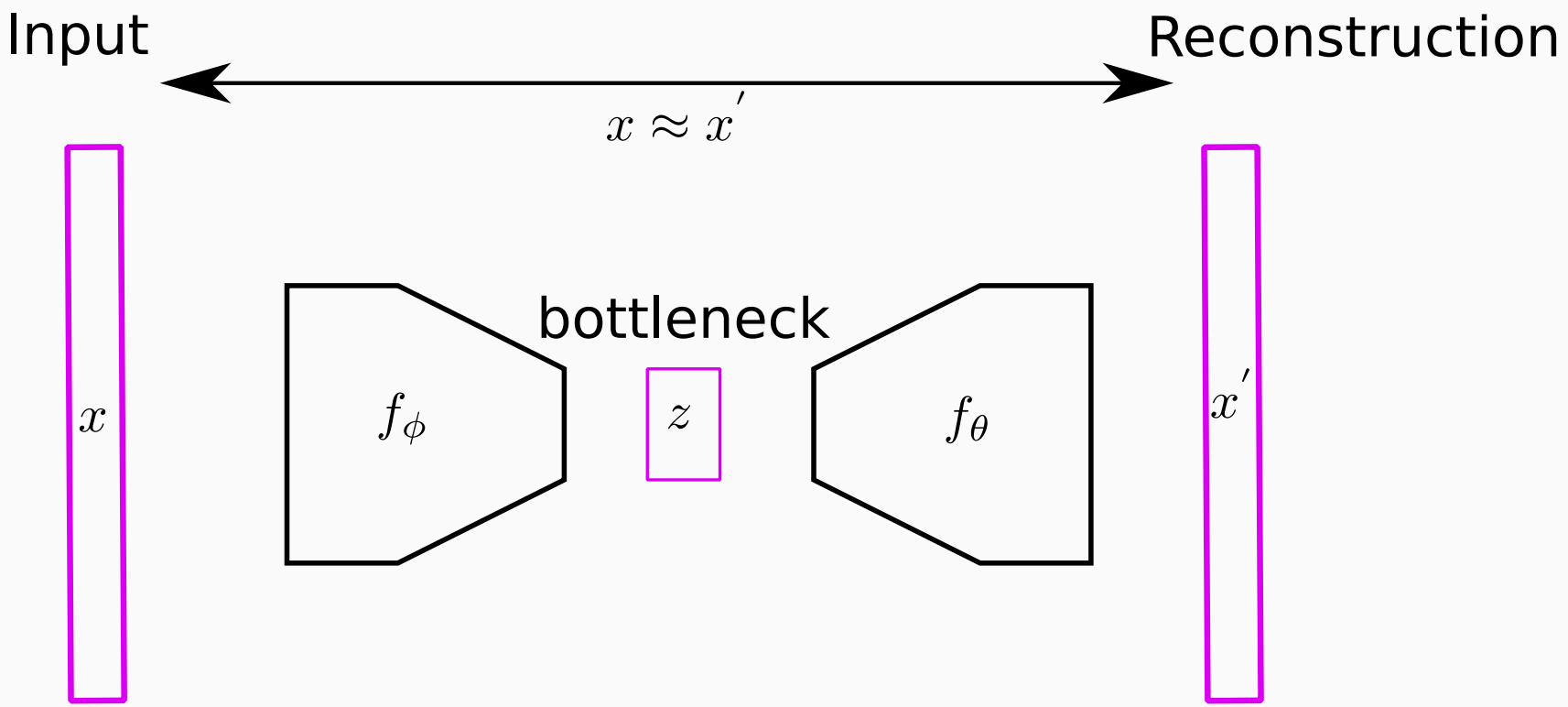
Reparameterised ELBO:

$$\begin{aligned}\mathcal{L}(\phi, \theta) &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\phi||p_\theta) \\ &= \underbrace{\mathbb{E}_{p(\epsilon)}[\log p_\theta(x|z)]}_{\text{Reconstruction loss}} - \underbrace{\mathrm{KL}(q_\phi||p_\theta)}_{\text{Regularisation}},\end{aligned} \tag{6}$$

where $z = g(\epsilon, \phi, x)$.

- We can now evaluate an unbiased Monte Carlo gradient estimate $\hat{\nabla}_{\phi,\theta}\mathcal{L}(\phi, \theta)$.

Autoencoder



- Compress and then decompress information.

Training the VAE

Algorithm 1: Stochastic optimization of the ELBO. Since noise originates from both the minibatch sampling and sampling of $p(\boldsymbol{\epsilon})$, this is a doubly stochastic optimization procedure. We also refer to this procedure as the *Auto-Encoding Variational Bayes* (AEVB) algorithm.

Data:

$\mathcal{D}$: Dataset

$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$: Inference model

$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$: Generative model

Result:

$\boldsymbol{\theta}, \boldsymbol{\phi}$: Learned parameters

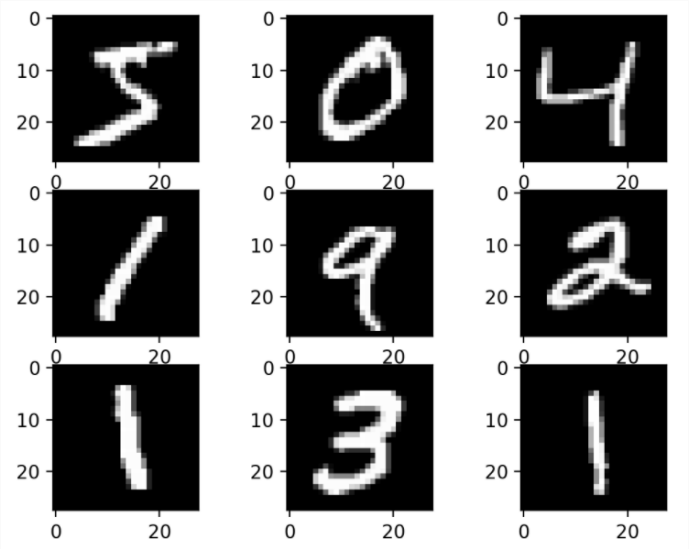
$(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow$ Initialize parameters

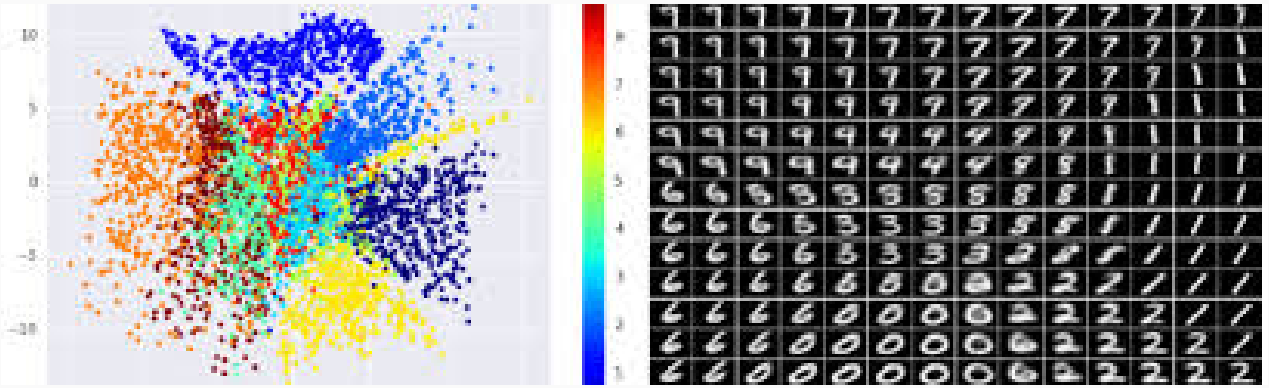
while *SGD not converged* **do**

- $\mathcal{M} \sim \mathcal{D}$ (Random minibatch of data)
- $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ (Random noise for every datapoint in $\mathcal{M}$)
- Compute $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$ and its gradients $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$
- Update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using SGD optimizer

end

The latent space: aka the CODE





Top: The $\boldsymbol{x}$-space (or the data spcae)

Bottom: The $\boldsymbol{z}$-space (how the posterior looks)

What I left out

1. Blurriness.
2. KL annealing.
3. Hierarchical VAE (more structured latent variables)
4. Clustering in $\boldsymbol{z}$- space.

Applications: Chemical design

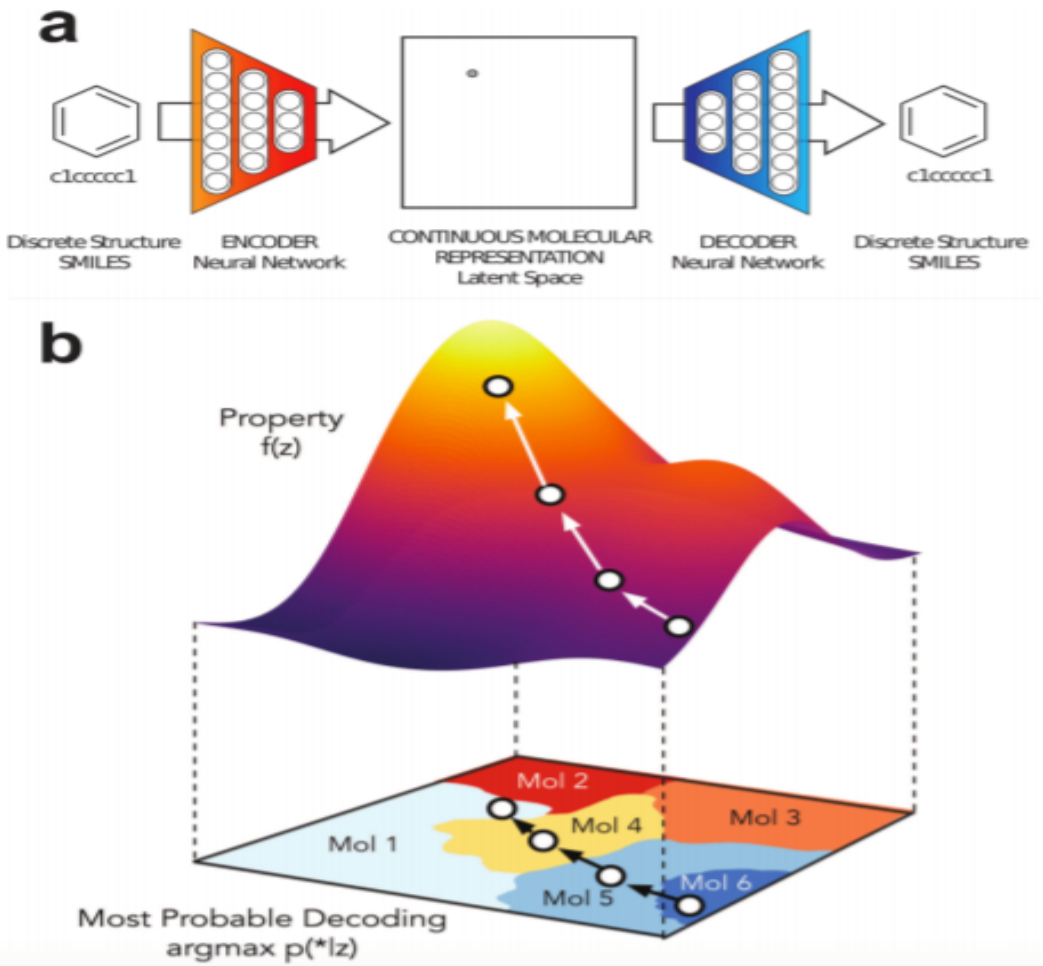
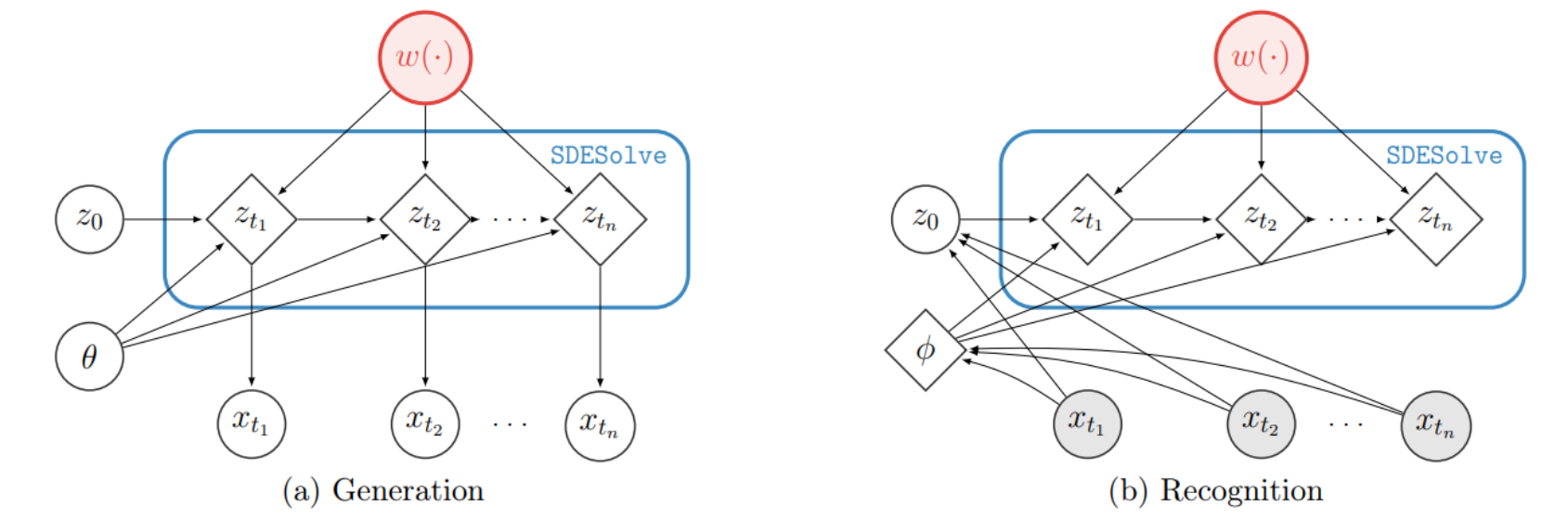


Figure 4.2: (a) Application of a VAE to chemical design in (Gómez-Bombarelli *et al.*, 2018). A latent continuous representation $\mathbf{z}$ of molecules is learned on a large dataset of molecules. (b) This continuous representation enables gradient-based search of new molecules that maximizes $f(\mathbf{z})$, a certain desired property.

Applications: Latent SDE



- The latent code is now a diffusion process*.
- Requires a differentiable SDE solver.

* Xuechen Li et al., 2020.

Normalising flows

Density modelling

Deep latent variable model

$$\begin{aligned} \boldsymbol{z} &\sim p(\boldsymbol{z}) \\ \boldsymbol{\eta} &= \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}) \\ p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) &= p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{\eta}), \end{aligned} \tag{7}$$

Alternatively, we can use the change of variable formula:

$$\begin{aligned} \boldsymbol{z} &\sim p(\boldsymbol{z}) \\ \boldsymbol{x} &= \mathcal{T}(\boldsymbol{z}) \\ \log p(\boldsymbol{x}) &= \log p(\boldsymbol{z}) + \log \left|\det J_{\mathcal{T}}(\boldsymbol{z})\right|^{-1} \\ &= \log p(\mathcal{T}^{-1}(\boldsymbol{x})) + \log \left|\det J_{\mathcal{T}^{-1}}(\boldsymbol{z})\right| \end{aligned} \tag{8}$$

- $\mathcal{T} : \mathbb{R}^D \to \mathbb{R}^D$ is an invertible and differentiable function.
- $p(\boldsymbol{z})$ is a simple base distribution like $\mathcal{N}(0, \mathrm{I})$.

Density modelling

- $\mathcal{T} : \mathbb{R}^D \to \mathbb{R}^D$ is an invertible and differentiable function.
- They are composable.

Consider the composition $\mathcal{T}_1 \circ \mathcal{T}_2$. Then we have:

$$(\mathcal{T}_1 \circ \mathcal{T}_2)^{-1} = \mathcal{T}_1^{-1} \circ \mathcal{T}_2^{-1}$$

$$\det J_{\mathcal{T}_1 \circ \mathcal{T}_2} = \det J_{\mathcal{T}_2}(\mathcal{T}_1(\boldsymbol{z})) \cdot J_{\mathcal{T}_1}(\boldsymbol{z}).$$

Normalising flow

- $\mathcal{T} : \mathbb{R}^D \to \mathbb{R}^D$ is an invertible and differentiable function.
- They are composable.

Chain together multiple transformations:

$$\boldsymbol{z}_K = \mathcal{T}_K \circ \mathcal{T}_{K-1} \circ \ldots \circ \mathcal{T}_1(\boldsymbol{z}_0), \quad \boldsymbol{z}_0 \sim p(\boldsymbol{z})$$

$$\boldsymbol{x} = \boldsymbol{z}_K$$

$$\log p(\boldsymbol{x}) = \log(\boldsymbol{z}_K) = \log p(\boldsymbol{z}_0) + \sum_{k=1}^{K} \log \left| \det J_{\mathcal{T}_k}(\boldsymbol{z}_k) \right|^{-1}.$$

- "Normalising" since $(\mathcal{T}_K \circ \mathcal{T}_{K-1} \circ \ldots \circ \mathcal{T}_1)^{-1} : \boldsymbol{x} \mapsto \boldsymbol{z}_0$.
- "Flow" term comes from the dynamical systems perspective.
- Learning transformation from data introduced by Tabak and Turner (2013).
- Introduced in Machine Learing by Rezende and Mohamed (2015).
- Parallel work during same period by Moselhy and Marzouk (2012).

Normalising flow

- In practise we implement either $\mathcal{T}$ or $\mathcal{T}^{-1}$ using a neural network $\boldsymbol{f}_\phi$.
- the forward transformation $\mathcal{T}$ is used when sampling, and the inverse transformation $\mathcal{T}^{-1}$ is used when evaluating densities.
- In either case, we must ensure that the model is invertible and has a tractable Jacobian determinant.
- Lower triangular $\boldsymbol{f}_\phi$ ensures Jacobian determinant is sum of diagonal elements.
- Ensuring $\boldsymbol{f}_\phi$ is invertible and explicitly calculating its inverse are not synonymous.

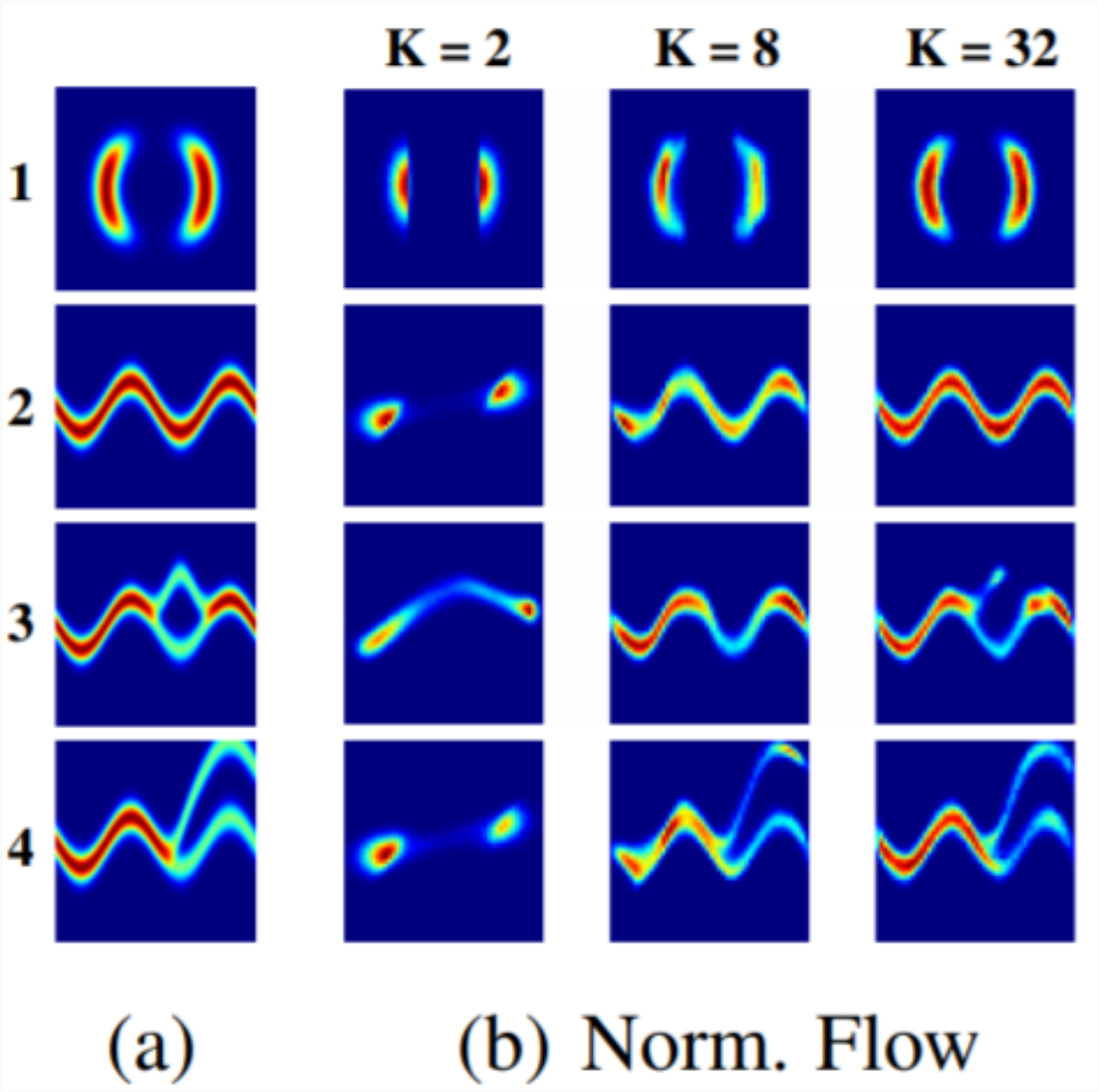
Normalising flow as VAE encoder

- Begin with an initial distribution $q(\boldsymbol{z}_0)$, a factorised Gaussian.
- Apply a normalising-flow: $\boldsymbol{f}_{\phi}^{K} \circ \ldots \circ \boldsymbol{f}_{\phi}^{1}$ to get the latent code $\boldsymbol{z}_K$.

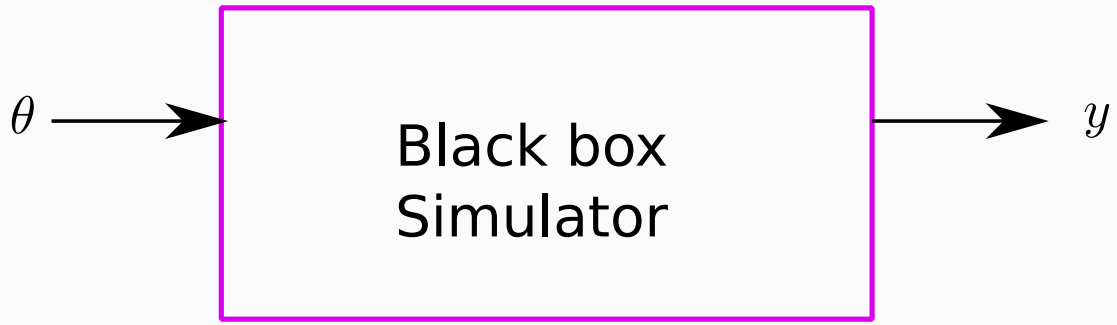
The ELBO is then given by

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q(\boldsymbol{z}_K)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}_K)] - \mathbb{E}_{q(\boldsymbol{z}_0)}[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}_0)] - \mathbb{E}_{q(\boldsymbol{z}_0)}\left[\sum_{k=1}^{K} \log \left| \det \frac{\partial \boldsymbol{f}_{\phi}^{k}}{\partial \boldsymbol{z}_k} \right|^{-1}\right]$$

Example: Inference of implicit simulators



Example



- The likelihood $p(\boldsymbol{y}|\boldsymbol{\theta})$ is intractable.
- E.g Stochastic epidemic models*
- Inference of $p(\boldsymbol{\theta}|\boldsymbol{y})$ is generally handled using ABC.
- However, ABC requires large number of simulations.

* TJ McKinley et al., 2018.

Application: Learning $p(\theta|y)$, amortised Bayesian inference

- Task: Infer the posterior $p(\boldsymbol{\theta}|\boldsymbol{y})$ of an implicit simulator model.

We can then write $\boldsymbol{\theta}$ as

$$\boldsymbol{\theta} = \mathcal{T}^{-1}(\boldsymbol{\zeta}; \boldsymbol{y}) \quad \text{with} \quad \boldsymbol{\zeta} \sim \mathcal{N}(0, \mathbb{I}),$$

with its conditional density

$$q_\phi(\boldsymbol{\theta}|\boldsymbol{y}) = p(\boldsymbol{\zeta} = \mathcal{T}(\boldsymbol{\theta}; \boldsymbol{y})) \left|\det J_\mathcal{T}\right| .$$

We can learn the parameters ϕ by maximising the log probability, with samples from $p(\boldsymbol{y}, \boldsymbol{\theta})$.

Bayesian inference:

$$\boldsymbol{\theta} \sim q_\phi(\boldsymbol{\theta}|\boldsymbol{y} = \boldsymbol{y}_o).$$

- Amortisation: Inference conditioned on one particular dataset $\boldsymbol{y}_o$.
- Flow $\mathcal{T}$, implemented as a deep net $\boldsymbol{f}_\phi$.

One-shot inference by learning posterior directly

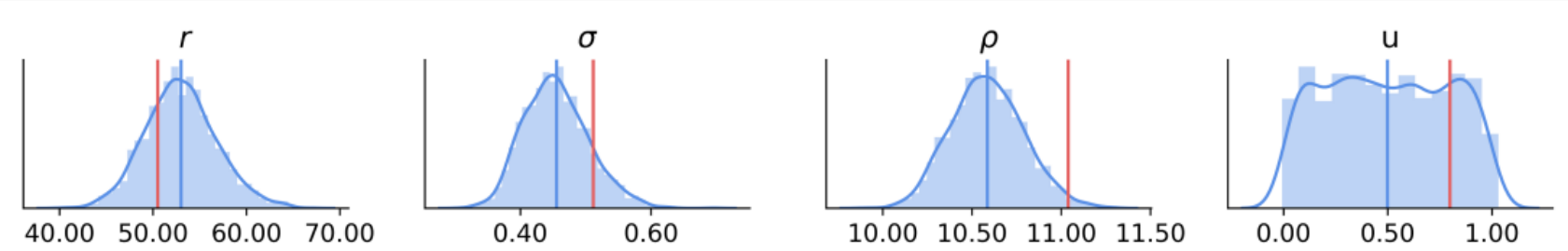
Stochastic* Time-Series Model - The Ricker Model.

$$\xi_t \sim \mathcal{N}(0, \sigma^2)$$

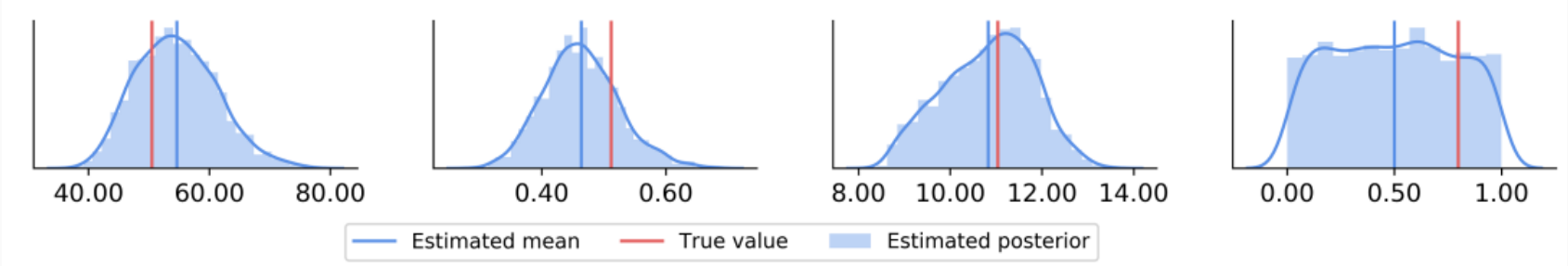
$$N_{t+1} = rN_t e^{-N_t + \xi_t}$$

$$x_t \sim \text{Pois}(\rho N_t)$$

Neural Inference



ABC



ST Radev et al., 2020. *SN Wood 2010.

Normalising flow

- I haven't discussed anything about how one designs $\boldsymbol{f}_\phi$.
- That will be the frontier talk today.