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

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.

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*:

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

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

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

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

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.

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:

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

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:

$$\boldsymbol{\eta} = \boldsymbol{f}_\theta(Pa(\boldsymbol{x}))$$

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

Deep latent variable model

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

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

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

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

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

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

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

DLVM for multivariate Bernoulli data

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

- f_θ 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 $p_\theta(z|x)$, we can generate data from the DLVM.
- $p_\theta(z|x)$ is intractable.
- We will apply VI.

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

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

- Note the conditioning by x in the variational approximation.
- Problem: We will need as many parameters as datapoints.

Variational Autoencoder
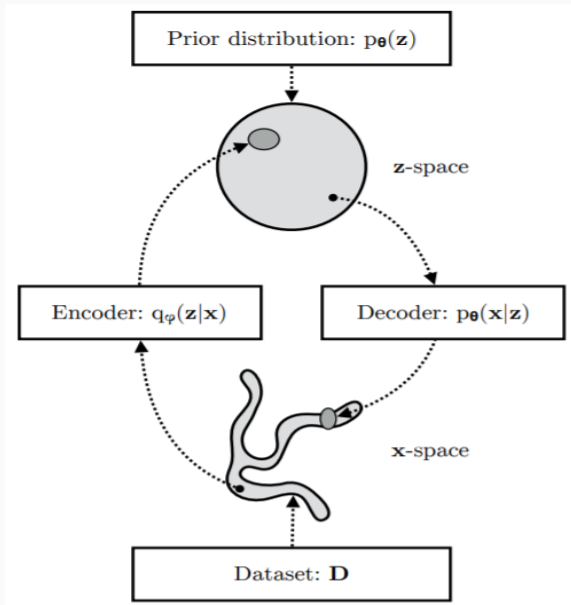
Bayesian inference in Deep latent variable model

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 ϕ resp.

Bayesian inference in DLVM

If we get the gradient of the ELBO wrt θ, ϕ then these parameters can be updated using gradient descent:

$$\theta \leftarrow \theta + \gamma \nabla_{\theta} \mathcal{L}_{\phi,\theta}(x)$$
$$\phi \leftarrow \phi + \gamma \nabla_{\phi} \mathcal{L}_{\phi,\theta}(x), \tag{4}$$

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

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

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

Reparameterisation trick

Amortised approximation:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \boldsymbol{f}_\phi(\boldsymbol{x})$$
$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \prod_d q_\phi(z_d|\boldsymbol{x}) = \prod_d \mathcal{N}(z_d; \mu_d, \sigma_d^2),$$

which we can write using re-parameterisation as:

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

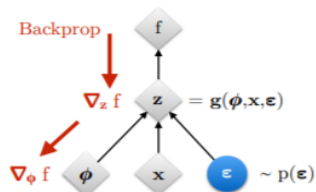- $\boldsymbol{f}_\phi$ in known as the Encoder, since it compresses information from the complex observation space to the latent space.
- You can use the same way of encoding through a full-rank Gaussian.
- This way of conditioning on the data $\boldsymbol{x}$ is known as amortisation.
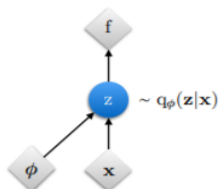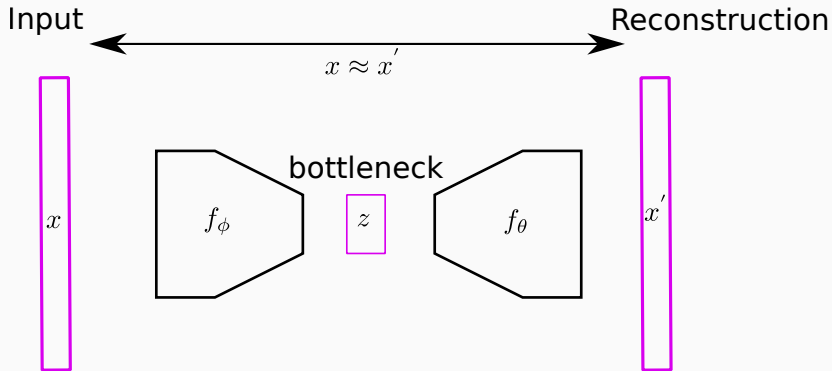
Monte Carlo gradient of ELBO

Reparameterised ELBO:

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}(q_\phi || p_\theta)$$

$$= \underbrace{\mathbb{E}_{p(\epsilon)}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction loss}} - \underbrace{\mathrm{KL}(q_\phi || p_\theta)}_{\text{Regularisation}}, \qquad (6)$$

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

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

Autoencoder

Input   Reconstruction

$x \approx x^{'}$

bottleneck

x f_{ϕ} z f_{θ} $x^{'}$

- Compress and then decompress information.

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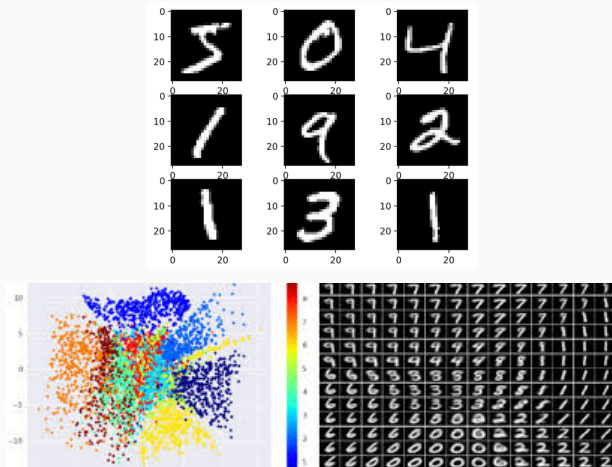 $\mathcal{M} \sim \mathcal{D}$ (Random minibatch of data)
    $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ (Random noise for every datapoint in $\mathcal{M}$)
    Compute $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$ and its gradients $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}}\tilde{\mathcal{L}}_{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

The latent space: aka the CODE

Top: The *x*-space (or the data spcae)
Bottom: The *z*-space (how the posterior looks)

What I left out

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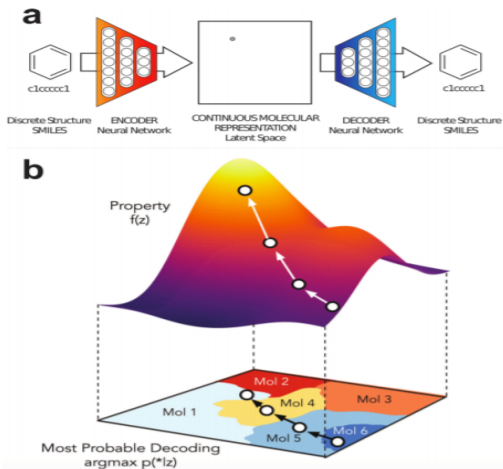3. Hierarchical VAE (more structured latent variables)
4. Clustering in z- space.

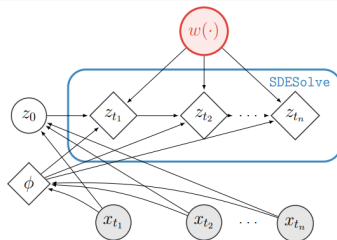**Figure 4.2:** (a) Application of a VAE to chemical design in (Gómez-Bombarelli *et al.*, 2018). A latent continuous representation **z** of molecules is learned on a large dataset of molecules. (b) This continuous representation enables gradient-based search of new molecules that maximizes $f(\mathbf{z})$,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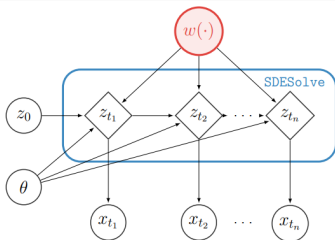
(a) Generation

(b) Recognition

- 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

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

Alternatively, we can use the change of variable formula:

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

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

- $\mathcal{T} : \mathbb{R}^D \to \mathbb{R}^D$ 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:

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

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

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

- "Normalising" since $(\mathcal{T}_K \circ \mathcal{T}_{K-1} \circ \ldots \circ \mathcal{T}_1)^{-1} : \mathbf{x} \mapsto \mathbf{z}_0$.
- "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.
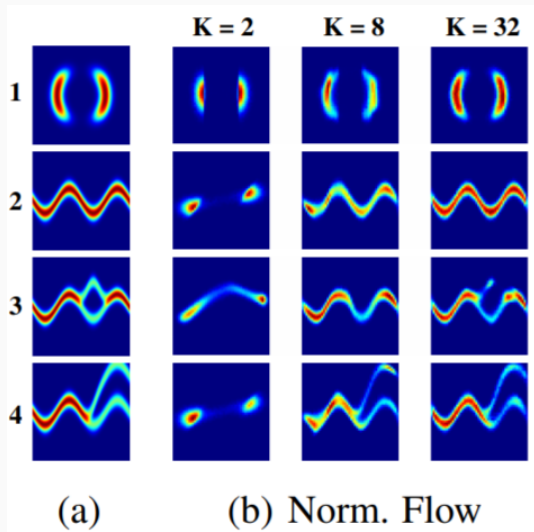- Ensuring $\boldsymbol{f}_\phi$ is invertible and explicitly calculating its inverse are not synonymous.

Normalising flow as VAE encoder

- Begin with an initial distribution $q(z_0)$, a factorised Gaussian.
- Apply a normalising-flow: $f_\phi^K \circ \ldots \circ f_\phi^1$ to get the latent code z_K.

The ELBO is then given by

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{q(z_K)}[\log p_\theta(x, z_K)] - \mathbb{E}_{q(z_0)}[\log q_\phi(z_0)] -$$
$$\mathbb{E}_{q(z_0)} \left[\sum_{k=1}^{K} \log \left| \det \frac{\partial f_\phi^k}{\partial z_k} \right|^{-1} \right]$$

(a) (b) Norm. Flow

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

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

We can then write θ as

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

with its conditional density

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

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

Bayesian inference:

$$\theta \sim q_\phi(\theta|y = y_o).$$

- Amortisation: Inference conditioned on one particular dataset y_o.
- Flow $\mathcal{T}$, implemented as a deep net f_ϕ

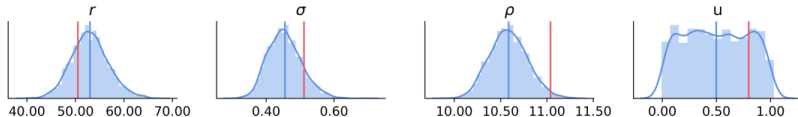## One-shot inference by learning posterior directly

Stochastic Time-Series Model - The Ricker Model.
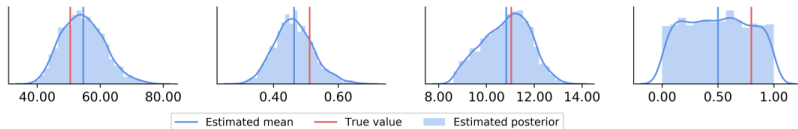
$$\xi_t \sim \mathcal{N}(0, \sigma^2)$$
$$N_{t+1} = rN_t e^{-N_t + \xi_t}$$
$$x_t \sim \text{Pois}(\rho N_t)$$

Neural Inference

ABC

Estimated mean — True value — Estimated posterior

ST Radev et al., 2020.

Normalising flow

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