Latent variable models in biology and ecology

Chapter 5: A gentle introduction to Variational Neural Networks

Sophie Donnet. INRAE

Master 2 MathSV. February 27, 2024

Context

- In statistical learning, two main tasks:
  - **Regression or classification**
  - **Reduction of dimension**

  Neural networks are used to construct the regression function, classifier or encoder-decoder (**autoencoder**).
- **Variational versions** are used when we do not want to optimize a parameter but a **probability distribution**
  - if one wants to structure the latent space
  - if one wants to perform Bayesien inference
- Relies on
  - **Neural networks** : we know already
  - **Variational EM algorithm**: we know already, but anyway it is not complicated

Overview

Basics on regression, classification, reduction of dimension

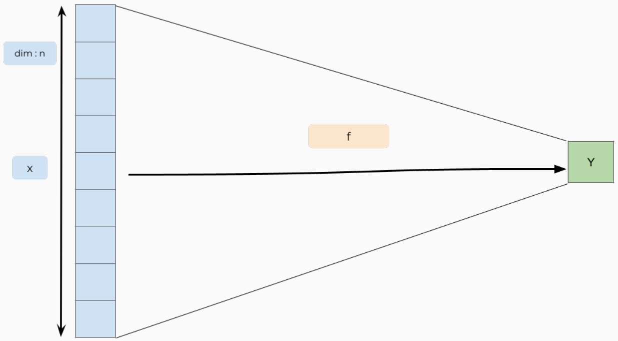
Neural networks

Variational versions of neural networks

Regression or classification

- Let $(\mathbf{X}, \mathbf{Y})$ be our dataset:
  - $(\mathbf{X}, \mathbf{Y}) = (X_i, Y_i)_{i \in 1, \dots, N_{obs}}$
  - $\forall i = 1, \dots, N_{obs}$, **Variables** $X_i \in \mathbb{R}^n$.
  - $Y_i \in \mathcal{Y}$ the variable to explain : **classification** or **regression**
- Looking for a function f **classifier** or **regression**
  - $f : \mathbb{R}^n \mapsto \mathcal{Y}$ and
  - such that
$$Y \approx f(X) \Leftrightarrow \text{Loss}(Y - f(X)) \text{ small}$$
  - If **regression** $\text{Loss}(Y - f(X)) = ||Y - f(X))||^2$
  - If **classification** : Loss = cross-entropy

Regression or classification



Reduction of dimension

Autoencoders are used for the reduction of dimension of (large) datasets.

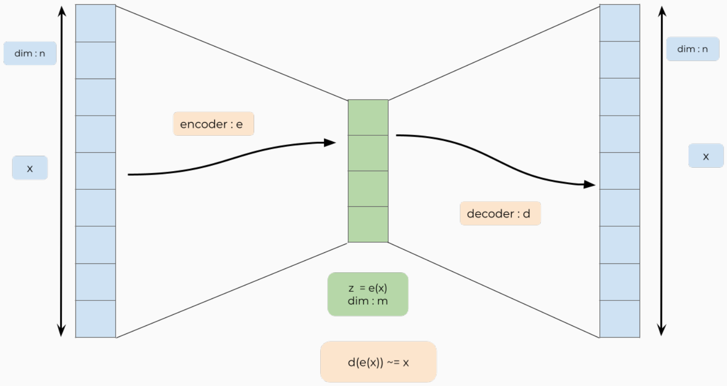
Let X be our dataset: $\mathbf{X} = (X_i)_{i \in 1,\dots,N_{obs}}$

- $\forall i = 1, \dots, N_{obs}$, $X_i \in \mathbb{R}^n$.
- Looking for two functions
  - **Encoder** $e : \mathbb{R}^n \mapsto \mathbb{R}^m$ and
  - **Decoder** $d : \mathbb{R}^m \mapsto \mathbb{R}^n$
- such that

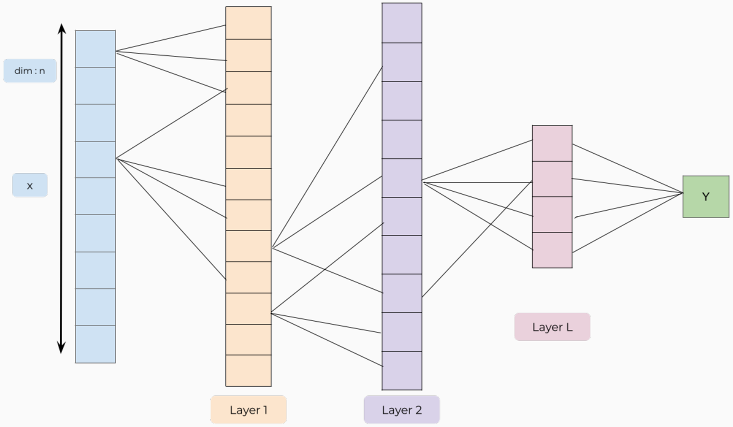
$$X \approx d(e(X)) \Leftrightarrow ||X - d(e(X))||^2 \text{ small}$$

- $Z = e(X)$: **latent variable**

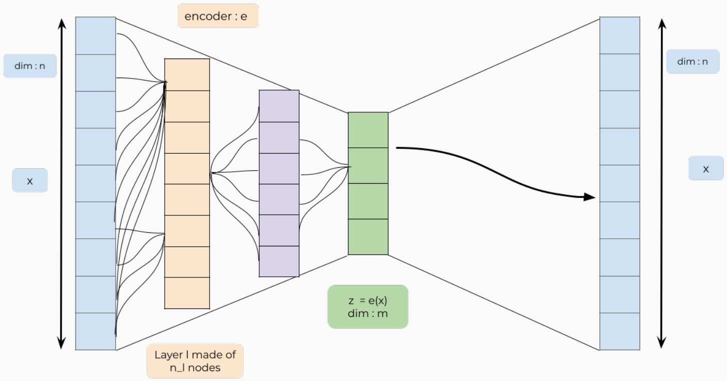
Autoencoder



About f: neural networks



About d and e : neural networks



About neural networks

One neuron : $f_j(\mathbf{X}) = \phi(< w_j, \mathbf{x} > + b_j)$ where

- ϕ the activation function : non linear
- $w_j = (w_j^1, \dots, w_j^n)$ are the weights of the input variables $(x^1, \dots, x^n)$
- b_j is the bias of neuron j.

At each layer ℓ of the neural network:

- Receive $n_{\ell-1}$ input variables $\mathbf{y}^{\ell-1} = (y_1^{\ell-1}, \dots, y_{n_{\ell-1}}^{\ell-1})$
- Create n_ℓ new variables. For variable j of layer l:

$$y_j^\ell = \phi(< w_j^\ell, \mathbf{y}^{\ell-1} > + b_j^\ell)$$

Unknown parameters θ

- $w_j^\ell \in \mathbb{R}^{n_\ell - 1}$, for $\ell = 1, \dots L$, for $j = 1, \dots, n_\ell$,
- $b_j^\ell \in \mathbb{R}$, for $\ell = 1, \dots L$, for $j = 1, \dots, n_\ell$,

Model choice

To choose:

- The number of layers L
- The number of neurons in each layer: n_ℓ :
- possibly $n_\ell > n$
- For **autoencoder** the middle layer $m < n$
- The activation function ϕ (possibly one for the hidden layers ϕ and one ψ for the activation layer)

Learning f, d and e

- **Regression or classification**

$\theta = (w_j^\ell, b_j^\ell)_{j=1\dots,n_\ell,\ell=1,\dots,L}$ are calibrated on a dataset $(X_i, Y_i)_{i=1,\dots,N_{obs}}$ by minimizing the loss function

$$\widehat{\theta} = \mathrm{argmin}_{\theta\in\Theta} \sum_{i=1}^{N_{obs}} \mathrm{Loss}(Y_i - f_\theta(X_i))$$

- **Autoencoder**

$\theta = (w_j^\ell, b_j^\ell)_{j=1\dots,n_\ell,\ell=1,\dots,L}$ are calibrated on a dataset $(X_i)_{i=1,\dots,N_{obs}}$ by minimizing the loss function

$$\widehat{\theta} = \mathrm{argmin}_{\theta\in\Theta} \sum_{i=1}^{N_{obs}} ||X_i - d_\theta \circ e_\theta(X_i)||^2$$

Optimisation by Stochastic gradient descent: see later for a reminder of the principle

PCA versus autoencoder

- Let $P \in M_{n,m}(\mathbb{R})$,
- **Hyp.**:
$$P'P = I_n$$
- Let $P'X_i$ is the projector of vector X_i on the sub-vectorial space generated by the columns of P.
- We are looking for P minimizing the inertia of the projected dataset:

$$\begin{aligned}\widehat{P} &= \text{argmax}_{\{P \in M_{n,m}(\mathbb{R}), P'P = I_n\}} \sum_{i=1}^{N_{obs}} ||P'X_i||^2 \\ &= \text{argmin}_{\{P \in M_{n,m}(\mathbb{R}), P'P = I_n\}} \sum_{i=1}^{N_{obs}} ||X_i - PP'X_i||^2\end{aligned}$$

PCA versus autoencoder

- $W' = e$: **linear** encoder function
- $W = d$: **linear** decoder function
- Note that if you use neural networks with linear activation function and one layer, you will get W not necessarily orthogonal.

Link to a rigourous and clear demonstration

Minimization by Stochastic gradient descent.

Algorithm (by Rumelhart et al (1988))

- Choose an initial value of parameters θ and a learning rate ρ
- Repeat until a minimum is reached:
  - Split randomy the training set into N_B *batches* of size b ($n = b \times N_B$)
  - for each batch B set:

$$\theta := \theta - \rho \frac{1}{b} \sum_{i \in B} \nabla_\theta \left\{ \text{Loss}(f(\mathbf{X}_i, \theta), Y_i) \right\}$$

Remarks:

- Each iteration is called an *epoch*.
- The number of epochs and batches are parameters to tune
- Difficulty comes from the computation of the gradient

Calculus of the gradient for the regression

- $Y \in \mathbb{R}$.
- $R_i = \text{Loss}(f(\mathbf{X}_i, \theta), Y_i) = (Y_i - f(\mathbf{X}_i, \theta))^2$
- For any activation function ϕ (hidden layers) and ψ

Partial derivatives of R_i with respect to the weights of the last layer

- Derivatives of $R_i = \left(Y_i - f(\mathbf{X}_i, \theta)\right)^2 = \left(Y_i - h^{(L+1)}(\mathbf{X}_i)\right)^2$ with respect to $(w_j^{(L+1)})_{j=1\dots J_L}$
- $a^{(L+1)}(\mathbf{X}) = b^{(L+1)} + w^{(L+1)} h^{(L)}(\mathbf{X}) \in \mathbb{R}^J$
- $$\begin{aligned} f(\mathbf{X}, \theta) &= h^{(L+1)}(\mathbf{X}) \\ &= \psi(a^{(L+1)}(\mathbf{X})) \\ &= \psi\left(b^{(L+1)} + \sum_{j=1}^{J_L} w_j^{(L+1)} h_j^{(L)}(\mathbf{X})\right) \end{aligned}$$
- $$\frac{\partial R_i}{\partial w_j^{(L+1)}} = -2\left(Y_i - f(\mathbf{X}_i, \theta)\right) \psi'\left(a^{(L+1)}(\mathbf{X}_i)\right) h_j^{(L)}(\mathbf{X}_i)$$

Partial derivatives of R_i with respect to the weights of the layer $L-1$

- Derivatives of $R_i = \left(Y_i - h^{(L+1)}(\mathbf{X}_i)\right)^2$ with respect to $(w_{jm}^{(L)})_{j=1\ldots J_L, m=1\ldots J_{L-1}}$
- $$\frac{\partial R_i}{\partial w_{jm}^{(L)}} = -2\left(Y_i - f(\mathbf{X}_i, \theta)\right) \psi'\left(a^{(L+1)}(\mathbf{X}_i)\right) \frac{\partial}{\partial w_{jm}^{(L)}} a^{(L+1)}(\mathbf{X}_i)$$

Partial derivatives of R_i with respect to the weights of the layer $L-2$

$$\begin{aligned}
a^{(L+1)}(\mathbf{X}) &= b^{(L+1)} + \sum_{j=1}^{J_L} w_j^{(L+1)} h_j^{(L)}(\mathbf{X}) \\
&= b^{(L+1)} + \sum_{j=1}^{J_L} w_j^{(L+1)} \phi\left(b_j^{(L)} + \sum_{m=1}^{J_{L-1}} w_{jm}^{(L)} h_m^{(L-1)}(\mathbf{X}) \right)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial w_{jm}^{(L)}} a^{(L+1)}(\mathbf{X}_i) &= w_j^{(L+1)} \phi' \left(b_j^{(L)} + \sum_{m=1}^{J_{L-1}} w_{jm}^{(L)} h_m^{(L-1)}(\mathbf{X}_i) \right) \\
&\quad \times h_m^{(L-1)}(\mathbf{X}_i) \\
&= w_j^{(L+1)} \phi'(a_j^L(\mathbf{X}_i)) h_m^{(L-1)}(\mathbf{X}_i)
\end{aligned}$$

Forward-Backward algorithm (at each iteration)

After some light effort, recurrence formula

- Given the current parameters
  - **Forward step** : From layer 1 to layer $L+1$, compute the $a_j^\ell(\mathbf{X}_i), \phi(a_j^\ell(\mathbf{X}_i))$
  - **Backward step** : From layer $L+1$ to layer 1, compute the partial derivatives (recurrence formula update)

Tuning the algorithm

- ρ: learning rate of the gradient descent
    - if ρ too small, really slow convergence with possibly reaching of a local minimum
    - if ρ too large, maybe oscilliation around an optimum without stabilisation
    - Adaptive choice of ρ (decreasing ρ)
- Batch calculation reduces the number of quantities to be stored in the forward / backward

Obviously

Many improved versions of the maximisation algorithm (momentum correction, Nesterov accelerated gradient, etc. . .)

Automatic differentiation

Success of the neural network comes from automatic differentiation, i.e. automatisation of the previously described forward-backward procedure to compute the derivatives : `Tensorflow`

Why variational neural networks?

Regression-Classification : Bayesian inference of the parameters θ

- Prior on θ: $\pi(\theta)$
- Estimation not of θ but of the posterior distribution of θ : $p(\theta|\mathbf{Y})$

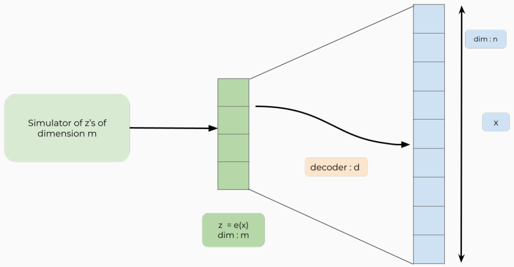
Autoencoder: give a structure on the latent space $\mathbf{Z}$

- Distribution on Z: $\pi(Z)$
- **Point estimation** of θ and **estimation of the posterior distribution of** Z **:** $p(Z|\theta, \mathbf{X})$

Variational : approximation of the distributions

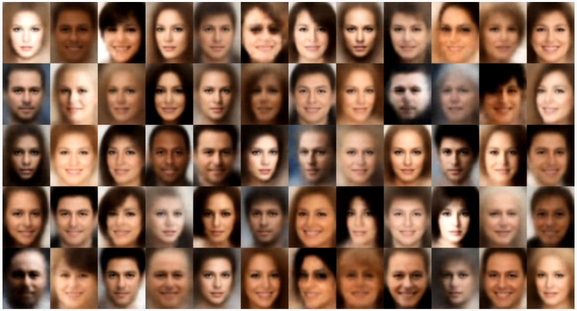
- $p(\theta|\mathbf{Y}) \approx q_{\mathbf{Y}}(\theta)$
- $p(Z|\theta, \mathbf{X}) \approx q_{\mathbf{X}}(Z)$

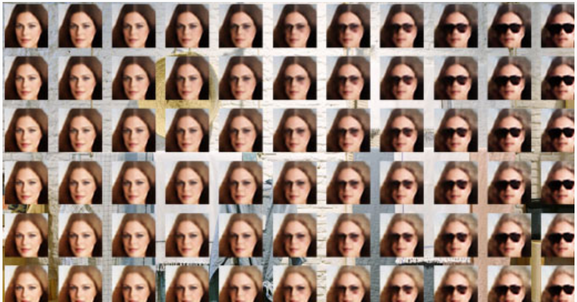
Using the autoencoder to simulate



- The optimization of the autoencoder supplies $(Z_1, \ldots, Z_{N_{obs}}) = (e(x_1), \ldots, e(X_{N_{obs}}))$
- **How can we simulate the** $z's$ such that $d(z)$ looks like my original data?
- How to construct a "machine" able to generate coherent other Z_i.
- Need to constrain/ structure the latent space.

Using the autoencoder to generate images





Probabilistic version of the autoencoder

- **Idea** : put a probabilistic distribution on the latent space and estimate the posterior distribution.
- **A statistical model with latent variables**

$$X_i = d(Z_i) + \epsilon_i$$

$$Z_i \sim_{i.i.d.} N_m(0, I_m)$$

$$\epsilon_i \sim_{i.i.d.} \mathcal{N}_n(0, cI_n)$$

- Likelihood

$$\ell(\mathbf{X}; d) = \int_{\mathbf{Z}} p(\mathbf{X}|\mathbf{Z}; d) p(\mathbf{Z}) d\mathbf{Z}$$

Not explicit

- EM requires the posterior distribution of **Z**

$$p(\mathbf{Z}|\mathbf{X}; d) \propto p(\mathbf{X}|\mathbf{Z}; d) p(\mathbf{Z})$$

The problem

$$\begin{aligned} \mathbf{X}_i &= d_\theta(Z_i) + \epsilon_i \\ Z_i &\sim {}_{i.i.d.} N_m(0, I_m) \\ \epsilon_i &\sim {}_{i.i.d.} \mathcal{N}_n(0, \sigma^2 I_n) \end{aligned}$$

Likelihood

$$\ell(\mathbf{X}; d_\theta) = \int_{\mathbf{Z}} \ell(\mathbf{X}|\mathbf{Z}; d_\theta) p(\mathbf{Z}) d\mathbf{Z}$$

No explicit form, linked ot the fact that $p(\mathbf{Z}|\mathbf{X}; d_\theta)$ is complex

The Evidence Lower BOund (ELBO)

- Let's simplify that distribution $p(\mathbf{Z}|\mathbf{X}; d_\theta)$

$$\begin{aligned} p(\mathbf{Z}|\mathbf{X}; d_\theta) &= q_{\mathbf{X}}(\mathbf{Z}; g, H) \\ \prod_{i=1}^{N_{obs}} p(Z_i|X_i; d_\theta) &\approx \prod_{i=1}^{N_{obs}} q_{X_i}(Z_i; g, H) \\ q_{X_i}(Z_i; g, h) &= \mathcal{N}_m(g(\mathbf{X}_i), H(g(\mathbf{X}_i))) \end{aligned}$$

  where g and H are chosen such that $D_{\mathrm{KL}}(q(\mathbf{Z}; \mathbf{X}, g, H), p(\mathbf{Z}|\mathbf{X}; d_\theta))$ is small

- Replace the likelihood by the ELBO

$$\begin{aligned} \mathrm{ELBO}(d_\theta, g, H) &= \ell(\mathbf{X}; d_\theta) - D_{\mathrm{KL}}(q(\mathbf{Z}; \mathbf{X}, g, H), p(\mathbf{Z}|\mathbf{X}; d)) \\ &= \mathbb{E}_{q_{\mathbf{X}}(\mathbf{Z}; g, H)}[\log p(\mathbf{X}|\mathbf{Z}; d_\theta)] - D_{\mathrm{KL}}(q_{\mathbf{X}}(\mathbf{Z}; g, H), p(\mathbf{Z})) \end{aligned}$$

Optimization: minimize $-\mathbf{ELBO}(d, g, H)$

$$-\mathrm{ELBO}(d, g, H) = -\mathbb{E}_{q_{\mathbf{X}}(\mathbf{Z}; g, H)}[\log p(\mathbf{X}|\mathbf{Z}; d_\theta)] + D_{\mathrm{KL}}(q_{\mathbf{X}}(\mathbf{Z}; g, h), p(\mathbf{Z}))$$

- **Reconstruction** term

$$-\mathbb{E}_{q_{\mathbf{X}}(\mathbf{Z}; g, H)}[\log p(\mathbf{X}|\mathbf{Z}; d_\theta)] = \mathbb{E}_{q_{\mathbf{X}}(\mathbf{Z}; g, H)} \left[\sum_{i=1}^{N_{obs}} \frac{||\mathbf{X}_i - d_\theta(Z_i)||^2}{2\sigma^2} \right]$$

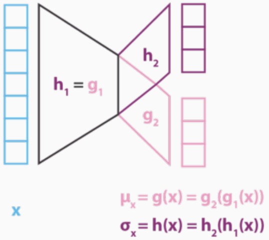
- **Regularisation** term : D_{KL}
- σ^2 : variance parameter which balances regularisation and reconstruction

About d_θ, g and H

d_θ neural network function as before

About g and H : called the "encoder part"

- $H(X)$ is a covariance so
  - it should be a square symmetric matrix
  - **Simplification**: diagonal matrix $H(\mathbf{X}) = diag(h^2(X))$ where $h(X) \in \mathbb{R}^m$
- $h(\mathbf{X}) = h_2(h_1(\mathbf{X}))$, $g(\mathbf{X}) = g_2(g_1(\mathbf{X}))$, $g_1 = h_1$
- g_2,g_2, h_1 neural networks



About the expectation

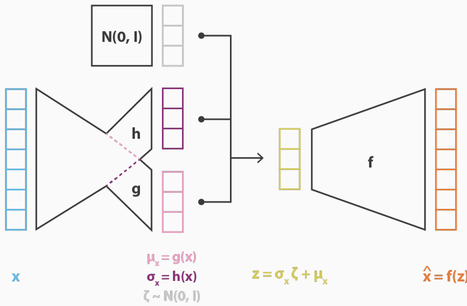
- $\mathbb{E}_{q_{\mathbf{X}}(\mathbf{Z};g,h)}\left[\sum_{i=1}^{N_{obs}} \frac{||\mathbf{X}_i - d_\theta(Z_i)||^2}{2\sigma^2}\right]$ can not be evaluated.
- Monte Carlo approximation on 1 realization
- Reparametrisation trick

$$Z_i^{sim} = g(X_i) + diag(h(X_i))\zeta_i, \quad \text{with } \xi_i \sim \mathcal{N}_m(0, \mathbb{I}_m)$$

$$\mathbb{E}_{q_{\mathbf{X}}(\mathbf{Z};g,h)}\left[\sum_{i=1}^{N_{obs}} \frac{||\mathbf{X}_i - d_\theta(Z_i)||^2}{2\sigma^2}\right] \approx \sum_{i=1}^{N_{obs}} \frac{||\mathbf{X}_i - d_\theta(Z_i^{(sim)})||^2}{2\sigma^2}$$

$$\sum_{i=1}^{N_{obs}} \frac{||\mathbf{X}_i - d_\theta(g(X_i) + diag(h(X_i))\zeta_i)||^2}{2\sigma^2}$$

Finally...



$$\text{loss} = C\,\|x - \hat{x}\|^2 + \mathrm{KL}[\,N(\mu_x, \sigma_x), N(0, I)\,] = C\,\|x - f(z)\|^2 + \mathrm{KL}[\,N(g(x), h(x)), N(0, I)\,]$$

Principal of variational Bayesian inference

- Approximate the posterior $p(\theta|Y)$ by $q(\theta)$ where $q \in \mathcal{R}$
- $\mathcal{R}$ family of simpler distributions. **Example**: $q(\cdot) = \mathcal{N}(\mu, \Sigma)$
- Approximating = Minimizing

$$D_{\mathrm{KL}}(q(\theta), p(\theta|\mathbf{Y})) = \mathbf{E}_q \left[\log \frac{q(\theta)}{p(\theta|\mathbf{Y})} \right]$$

The Magik trick

$$D_{\mathrm{KL}}(q(\theta), p(\theta|\mathbf{Y})) = \log \ell(\mathbf{Y}) + \left[\underbrace{- \mathbf{E}_q[\log \ell(\mathbf{Y}|\theta)\pi(\theta)] + \mathbf{E}_q[\log q(\theta)]}_{\mathcal{F}(q)} \right]$$

- $\log \ell(\mathbf{Y})$ independent of q
- Minimizing the Kullback–Leibler divergence w.r. to q is equivalent to minimizing $\mathcal{F}(q)$ with respect to q

$$\begin{aligned} \mathcal{F}(q) &= -\mathbf{E}_q[\log \ell(\mathbf{Y}|\theta)\pi(\theta)] + \mathbf{E}_q[\log q(\theta)] && (1) \\ &= -\mathbf{E}_q[\log \ell(\mathbf{Y}|\theta)] + \mathbf{E}_q\left[\log \frac{q(\theta)}{\pi(\theta)}\right] && (2) \\ &= D_{\mathrm{KL}}(q, \pi) - \mathbf{E}_q[\log \ell(\mathbf{Y}|\theta)] && (3) \end{aligned}$$

Parametrization of q

Choose a **parametric** form in $q = q_\eta$.

- For example: $q = \mathcal{N}(\mu, \Sigma)$

$$\hat{\eta} = \arg\min_\eta \mathcal{F}(\eta) = \arg\min_\eta D_{\mathrm{KL}}(q_\eta, \pi) - \mathbf{E}_{q_\eta}[\log \ell(\mathbf{Y}|\theta)]$$

- Optimisation by gradient descent
- **BUT** expectation not explicit

Monte Carlo approximation

- With neural networks, $\mathbf{E}_{q_\eta}[\log \ell(\mathbf{Y}|\theta)]$ not explicit (activation functions non linear)
- Approximation by Monte Carlo : assume that $\theta^{(m)} \sim q_\eta$, $m = 1, \dots, M$

$$\widehat{\mathcal{F}}(\eta) = \frac{1}{M}\sum_{m=1}^{M} \log \frac{q_\eta(\theta^{(m)})}{\pi(\theta^{(m)})} - \log \ell(\mathbf{Y}|\theta^{(m)})$$

- **Problem**: we lost the explicit dependence in η through the simulations $\theta^{(m)}$
- **Solution** : reparametrisation

$$\xi^{(m)} \sim \mathcal{N}(0, \mathbf{I}) \quad \text{and} \quad \theta^{(m)} = \phi(\xi^{(m)}, \eta)$$

$$\widehat{\mathcal{F}}(\eta) = \frac{1}{M}\sum_{m=1}^{M} \log q_\eta(\phi(\xi^{(m)}, \eta)) - \log \pi(\phi(\xi^{(m)}, \eta)) - \log \ell(\mathbf{Y}|\phi(\xi^{(m)}, \eta))$$

Remarks

$$\widehat{\mathcal{F}}(\eta) = \frac{1}{M}\sum_{m=1}^{M} \log q_\eta(\phi(\xi^{(m)},\eta)) - \log \pi(\phi(\xi^{(m)},\eta)) - \log \ell(\mathbf{Y}|\phi(\xi^{(m)},\eta))$$

- People take $M = 1$
- $D_{\mathrm{KL}}(q_\eta, \pi)$ may be explicit (for Gaussian distributions for instance) but not used in practice
- $\xi^{(m)}$ are resimulated each time we compute the gradients

More details for the regression case

- θ are the parameters (weights and bias)
- Prior gaussian distribution on θ : $\theta \sim \mathcal{N}(0, \mathbb{I})$
- If regression $Y_i = f_\theta(X_i) + \epsilon_i, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$

$$-\ell(\mathbf{Y}, \phi(\xi^{(m)}, \eta)) = \left[\sum_{i=1}^{N_{obs}} \frac{||Y_i - f_{\phi(\xi^{(m)}, \eta)}(X_i)||^2}{2\sigma^2} \right]$$

Conclusion

- Easy to understand all the tools
- Now, how easy is it to encode this?