Neural nets and latent variable models

Statistiques au sommet de Rochebrune

Pierre Gloaguen

March 25, 2024

Université Bretagne Sud

Neural networks

Neural network

- A parameterized function:

$$\begin{array}{rccc} f_\theta: & \mathbb{R}^{d_x} & \to & \mathbb{R}^{d_y} \\ & x & \mapsto & f(x|\theta) \end{array} ,$$

  which is a composition of non linear functions.
- The function depends both on θ and some intermediate non linear functions.

Neural network

- A parameterized function:

$$\begin{aligned} f_\theta : \quad \mathbb{R}^{d_x} &\to \mathbb{R}^{d_y} \\ x &\mapsto f(x|\theta) \quad , \end{aligned}$$

  which is a composition of non linear functions.
- The function depends both on θ and some intermediate non linear functions.
- For instance, for $d_x = 2, d_y = 1$, consider a set of *b*iases and **W**eights $\theta = \{b_1 \in \mathbb{R}^4, b_2 \in \mathbb{R}, \mathbf{W}_1 \in \mathbb{R}^{4\times 2}, \mathbf{W}_2 \in \mathbb{R}^{1\times 4}$;

$$f(x|\theta) = \tanh\left(\mathbf{W}_2 \cdot \text{sigmoid}\left(\mathbf{W}_1 x + b_1\right) + b_2\right) \quad .$$

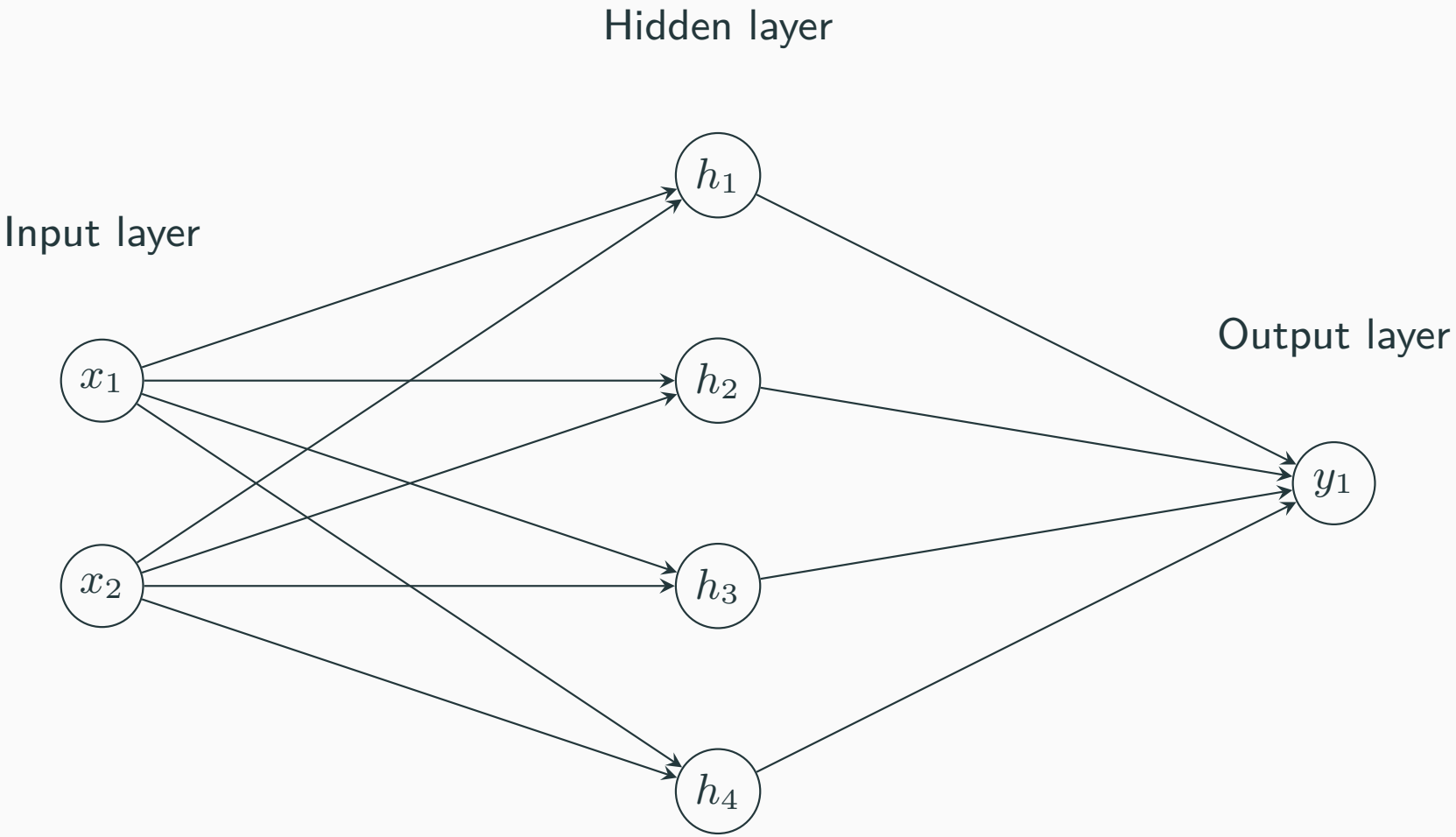
  We can introduce some intermediary notations and write

$$\begin{aligned} \sigma_1(z) &:= \text{sigmoid}(z) \\ \sigma_2(z) &:= \tanh(z) \\ h^{(1)} &:= \sigma_1(\mathbf{W}_1 x + b_1) \quad , \end{aligned}$$

  We then have

$$f(x|\theta) = \sigma_2(\mathbf{W}_2 h^{(1)} + b_2) \quad .$$

Graphical representation of a neural network

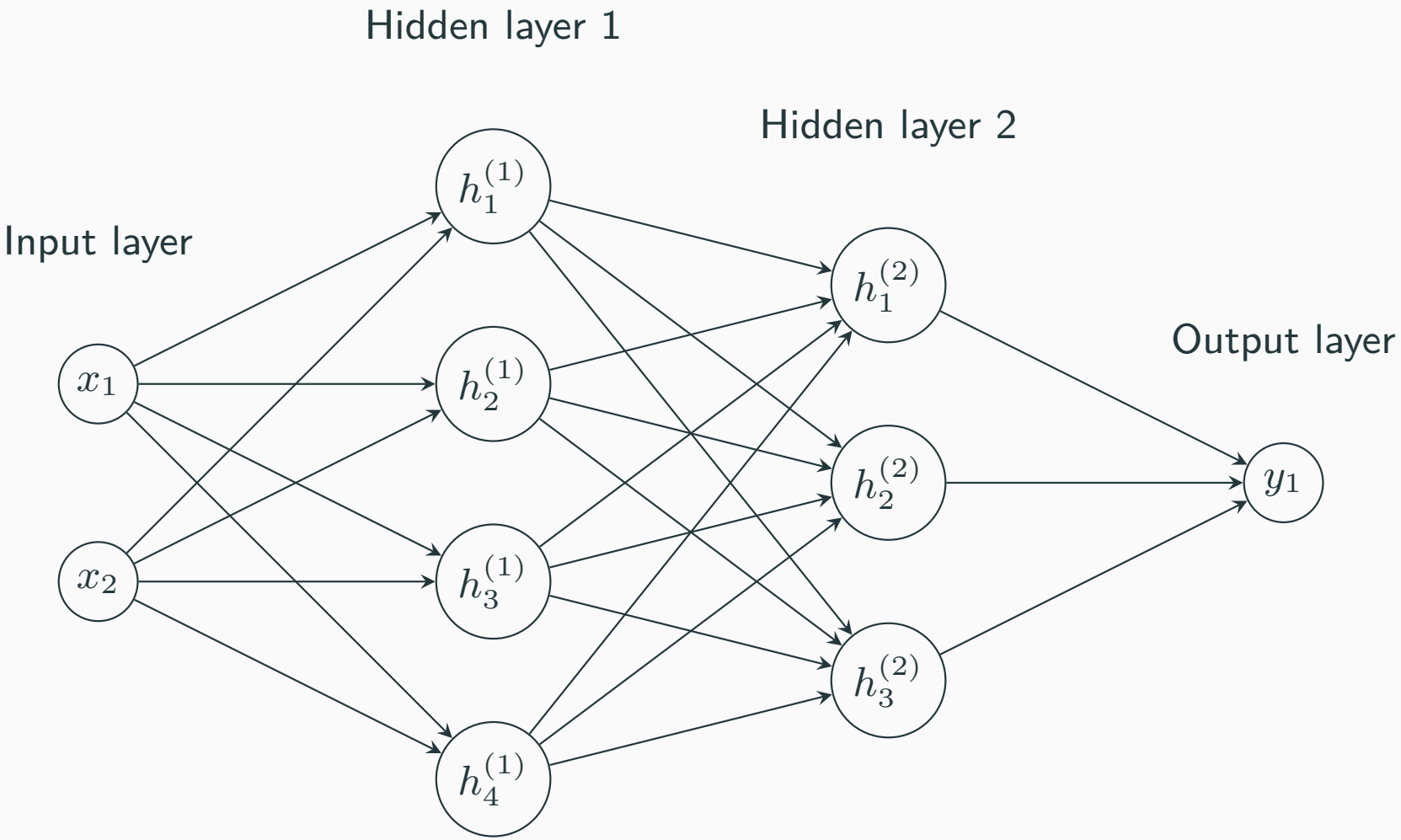


Deep neural network

- Build a sequence of vectors $(h^{(k)})_{0 \leq k \leq D}$ where D is the depth of a network:

$$
\begin{aligned}
h^{(0)} &= x \\
h^{(k)} &= \sigma_k(\mathbf{W}_k h^{(k-1)} + b_k), \text{ for } 1 \leq k \leq D \\
f(x|\theta) &= h^{(D)}
\end{aligned}
$$

Graphical representation of a deep neural network



Classical non linear functions

- There exists a dictionary of classical non linear functions
  - $\mathrm{ReLU}(x) = \max(0, x)$;
  - $\mathrm{sigmoid}(x) = \frac{1}{1+\mathrm{e}^{-x}}$;
  - $\tanh(x) = \frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}}$;
  - etc. . .

Convolution filter

- In the fully connected context, each neuron is a linear combination of all neurons of previous layer;
- No relation is imposed between each linear combination (over the rows of $\mathbf{W}$).
- For computational efficiency, and to impose some invariance, one can impose that linear combinations only apply locally, and are the same on the different regions of each layer.

Simple convolution filter

- For instance, to create a layer which gives the local differences in the vector:

$$x = \begin{pmatrix} 2 \\ 1 \\ -6 \\ 4 \\ 5 \end{pmatrix}$$

- Define a kernel

$$\begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$

The resulting *convolutional layer* is a vector h of size $5 - 3 + 1$ where, for $1 \leq i \leq 3$:

$$h = \begin{pmatrix} -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{pmatrix} \times x = \begin{pmatrix} -8 \\ 3 \\ 11 \end{pmatrix}$$

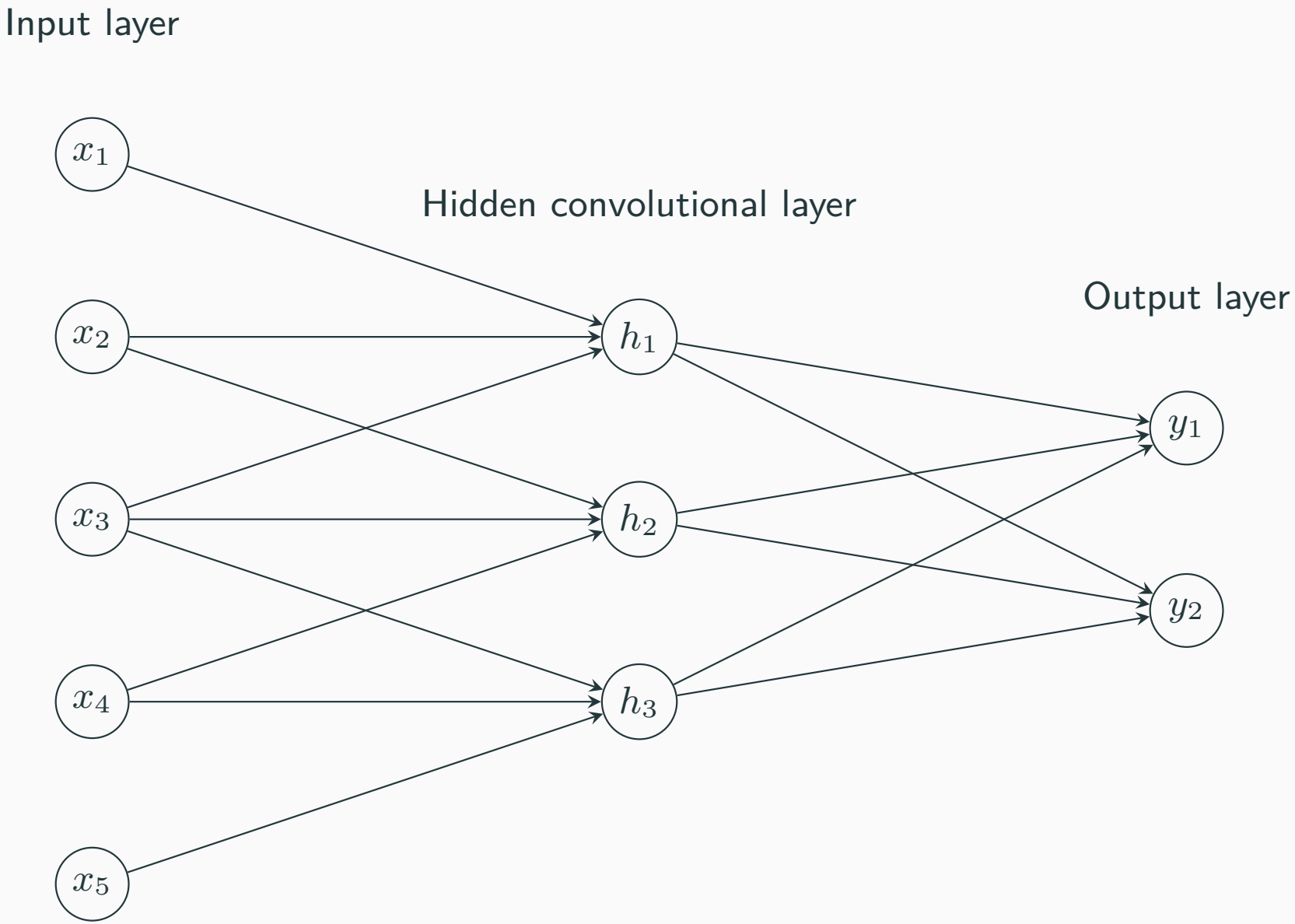
- *Remark* Of course, technical extensions exists to deal with bounds of x (stride and padding).

Convolutional layer

- A convolutional layer imposes a sparse weight matrix whose parameters are shared among rows:
- For instance, if the convolutional filter implies 3 elements at once:

$$\mathbf{W} = \begin{pmatrix} w_1 & w_2 & w_3 & 0 & \dots & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & \dots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & w_1 & w_2 & w_3 \end{pmatrix}$$

Convolutional neural network



Neural nets as predictors

- Neural nets were first used for classical learning tasks:
  - Classification;
  - Regression.
- In this context, we have a data set $(x_{1:n}, y_{1:n})$:
  - x_k are the features (explanatory variables):
  - y_k is the response, a variable to predict (a label or a value).
- $f(x_k|\theta)$ is a prediction for y_k.

Neural nets as generators

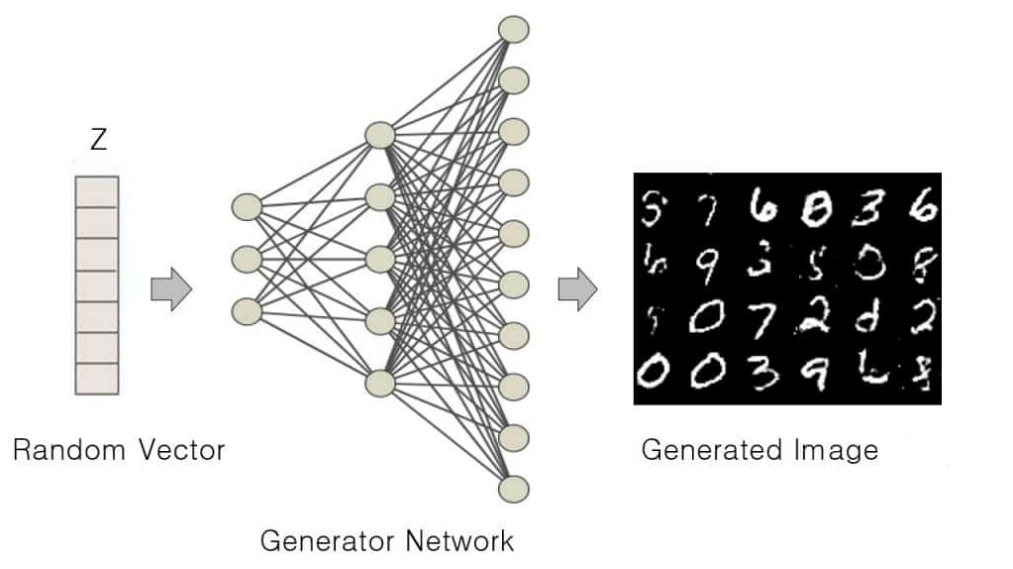
- Another use of neural nets is for generative modelling;
- Probabilistic framework where the high dimensional data $\mathbf{Y}$ are assumed to arise from a lower dimension random variable $\mathbf{Z}$ with a known distribution.

$$\mathbf{Z} \sim \text{Distribution}(\cdot)$$

$$\mathbf{Y}|\mathbf{Z} = f(\mathbf{Z}|\theta) + \mathbf{E},$$

where:

- f is a neural network;
- $\mathbf{E}$ is some random variable (reconstruction error term).



Learning parameters of a neural net: cost function

Classical losses functions

- The parameter θ is learnt by minimizing a loss function $\mathcal{L}(\theta)$:
  - In the regression case (for $y_i \in \mathbb{R}$):

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i|\theta))^2$$

  - In the K classes classification (encoding $y_i = (0, \ldots, 0, \overbrace{1}^{k\text{-th value}}, 0, \ldots 0)^T$), and with $f(x_i|\theta)$ being a vector of probabilities:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} y_i^T \log f(x_i|\theta).$$

  - In the generative model case, **ideally**, the loglikelihood:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \log p(y_i|\theta) = \frac{1}{n} \sum_{i=1}^{n} \log \int p(y_i|z\theta) p(z) \mathrm{d}z$$

- Note that, in general, we have:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(\theta),$$

Learning parameters of a neural net: stochastic gradient descent (SGD)

Gradient descent to find $\hat{\theta} = \textbf{argmin}_\theta \mathcal{L}(\theta)$.

Starting from $\theta^{(0)}$ and building a sequence $(\theta^{(\ell)})_{\ell \geq 0}$ such that, for $\ell \geqslant 1$:

$$\theta^{(\ell)} = \theta^{(\ell-1)} + \gamma_\ell \nabla_\theta \mathcal{L}(\theta^{(\ell-1)}),$$

where γ_ℓ is a sequence of step sizes decreasing (at an appropriate rate) to 0.

Learning parameters of a neural net: stochastic gradient descent (SGD)

Gradient descent to find $\hat{\theta} = \textbf{argmin}_\theta \mathcal{L}(\theta)$.

Starting from $\theta^{(0)}$ and building a sequence $(\theta^{(\ell)})_{\ell \geq 0}$ such that, for $\ell \geqslant 1$:

$$\theta^{(\ell)} = \theta^{(\ell-1)} + \gamma_\ell \nabla_\theta \mathcal{L}(\theta^{(\ell-1)}),$$

where γ_ℓ is a sequence of step sizes decreasing (at an appropriate rate) to 0.

Stochastic gradient descent

In general, with n observations, we have:

$$\begin{aligned} \nabla_\theta \mathcal{L}(\theta) &= \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}_i(\theta) \\ &= \mathbb{E}_J[\nabla_\theta \mathcal{L}_J(\theta)], \qquad \text{where } J \sim \mathcal{U}[\![1; n]\!]. \end{aligned}$$

Thus, for $1 \leq m \ll n$,

$$\widehat{\nabla}_\theta \mathcal{L}(\theta) := \frac{1}{m} \sum_{k=1}^{m} \nabla_\theta \mathcal{L}_{J_k}(\theta) \text{ where } J_k \overset{\text{i.i.d.}}{\sim} \mathcal{U}[\![1; n]\!],$$

is an unbiased Monte Carlo estimator of the wanted gradient.

Learning parameters of a neural net: Chain rule and automatic differentiation

- SGD still requires to compute the gradient of $\nabla \mathcal{L}_i(\theta)$;
- Recall that $f(x|\theta)$ is built by setting:

$$h^{(0)} = \begin{pmatrix} 1 \\ x \end{pmatrix} \in \mathbb{R}^{d_0} \qquad \text{with } d_0 = d_x + 1$$

$$h^{(1)} = \sigma_1(\theta_1 h^{(k-1)}), \qquad \text{with } \theta_1 = \left(b_1, \mathbf{W}_1\right),\ b_1 \in \mathbb{R}^{d_1}, \mathbf{W}_1 \in \mathbb{R}^{d_1 \times d_0}$$

$$\vdots$$

$$f(x|\theta) = \sigma_D(\theta_D h^{(D-1)}), \qquad \text{with } \theta_d \in \mathbb{R}^{d_y \times d_{D-1}}$$

- Then[1]:

$$\nabla_\theta \mathcal{L}_i(\theta) = \begin{pmatrix} \frac{\partial \mathcal{L}_i(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial \mathcal{L}_i(\theta)}{\partial \theta_D} \end{pmatrix}$$

[1] As the gradient is a vector, all partial derivatives are implicitly vectorized here

Learning parameters of a neural net: Backward propagation algorithm

Notations

For $1 \leqslant k \leqslant D$

- $z^{(k)} := \theta_k h^{((k-1))}$, and recall that $h^{(k)} = \sigma_k(z^{(k)})$;
- $\frac{\partial h^{(k)}}{\partial z^{(k)}}$ is a $\mathbb{R}^{d_k}$-vector of elementwise derivatives;
- $\odot$: elementwise product for a vector, $\times$: matrix product;

Algorithm

- For all $1 \leqslant k \leqslant D$:

$$\overset{d_k \times d_{k-1} \text{ matrix}}{\frac{\partial \mathcal{L}_i(\theta)}{\partial \theta_k}} = \left(\delta_k \odot \frac{\partial h^{(k)}}{\partial z^{(k)}}\right) \times {h^{(k-1)}}^T,$$

  where $\delta_k := \frac{\partial \mathcal{L}_i(\theta)}{\partial h^{(k)}} \in \mathbb{R}^{d_k}$

- For any $1 \leqslant k < D$, δ_k satisfies the **backward recursion**:

$$\delta_k = \theta_{k+1}^T \times \left(\delta_{k+1} \odot \frac{\partial h^{(k+1)}}{\partial z^{(k+1)}}\right) \quad .$$

Learning parameters of a neural net:

- Note that most times are used multiple times, and easily computed;
- Computation of simple derivatives is easily done using automatic differentiation during the forward pass;
- A backward pass computes the wanted gradients.

Latent variable models

Latent variable models examples

Neural network generative model

For $1 \leq k \leq n$:

$$Z_k \sim \mathcal{N}_{d_z}(0, I_{d_z}) \qquad \text{Prior distribution}$$

$$Y_k | Z_k \sim \mathcal{N}_{d_y}\left(\text{NeuralNet}(Z|\theta), \sigma^2 I_{d_y}\right), \qquad \text{Observation model}$$

- Used to generate complex structures

Latent variable models examples

- On n sites, observers count abundances of p species resulting in a $n \times p$ matrix $\mathbf{Y}$
- On each site k, m covariates are measure, resulting in a matrix $n \times m$ matrix $\mathbf{X}$
- The intensity of presence is supposed to depend on covariates through a linear combination:

Poisson log-normal model

$$\mathbf{Z} = (Z_{i,j})_{1 \leq i \leq n, 1 \leq j \leq p}, Z_{i,j} \overset{\text{ind.}}{\sim} \mathcal{N}(0,1) \quad \text{Prior distribution}$$

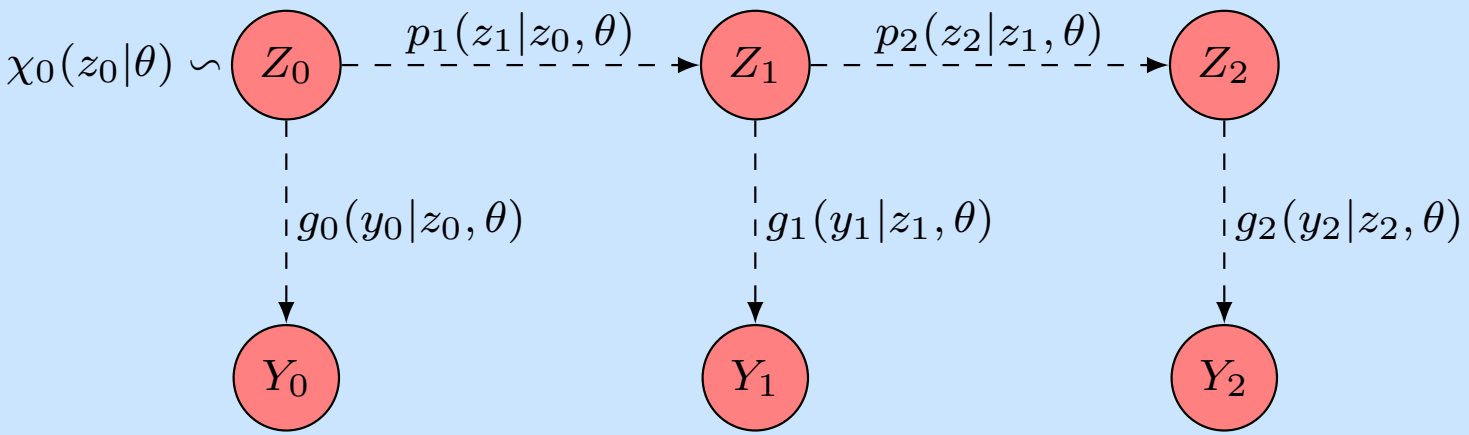
$$\mathbf{Y}|\mathbf{Z} \sim \mathcal{P}\text{oisson}(\exp(\mathbf{X}\theta + \mathbf{Z})), \quad \text{Observation model}$$

where θ is a $m \times p$ unknown matrix to estimate.

- Used in joint species modelling

Latent variable models examples

Hidden Markov models



$$
\begin{aligned}
Z_0 &\sim \chi_0(z_0,\theta) \\
Z_t|\{Z_{t-1}=z_{t-1}\} &\sim p_t(z_k|z_{k-1},\theta)
\end{aligned}
\qquad \text{Prior distribution}
$$

$$
Y_t|\{Z_t=z_t\} \sim g_t(y_t|z_t,\theta) \qquad \text{Observation model}
$$

- Use to model incomplete time series

Inference in latent variable models

- Classical learning objectives:
  - Computing the log-likelihood:

$$\log p(y_{1:n}) = \log \int_{\Omega_{\mathbf{Z}}} p(z)p(y_{1:n}|z)\mathrm{d}z$$

  - Learning the posterior distribution:

$$p(z|y_{1:n}) = \frac{p(z)p(y_{1:n}|z)}{p(y_{1:n})}$$

  - In a bayesian context, learn the full posterior

$$p(z, \theta|y_{1:n}) = \frac{p(z, \theta)p(y_{1:n}|z, \theta)}{p(y_{1:n})}$$

Variational autoencoder

Poisson log-normal model

$$\mathbf{Z} = (Z_{i,j})_{1 \leq i \leq n, 1 \leq j \leq p}, Z_{i,j} \overset{\text{ind.}}{\sim} \mathcal{N}(0,1) \quad \text{Prior distribution}$$

$$\mathbf{Y}|\mathbf{Z} \sim \mathcal{P}\text{oisson}(\exp(\mathbf{X}\theta + \mathbf{Z})), \quad \text{Observation model}$$

where θ is a $m \times p$ unknown matrix to estimate.

- In this context, we aim at learning θ.
- Ideally, we aim at maximizing the log-likelihood:

$$\log p(\mathbf{Y}|\theta) = \log \int_{\Omega_{\mathbf{Z}}} p(\mathbf{Y}|z,\theta)p(z)\mathrm{d}z$$

- However the integral is not tractable in general;
- Approximating it by Monte Carlo is not likely to work;

Evidence lower Bound

- Let $q(z)$ be a probability density function of $\Omega_{\mathbf{Z}}$:

$$
\begin{aligned}
\log p(\mathbf{Y}|\theta) &= \log \int_{\Omega_{\mathbf{Z}}} p(\mathbf{Y}|z,\theta)p(z)\mathrm{d}z \\
&= \log \int_{\Omega_{\mathbf{Z}}} \frac{p(\mathbf{Y}|z,\theta)p(z)}{q(z)} q(z)\mathrm{d}z \\
&= \log \mathbb{E}_q \left[\frac{p(\mathbf{Y}|Z,\theta)p(z)}{q(z)} \right] \\
&\geqslant \mathbb{E}_q \left[\log \frac{p(\mathbf{Y}|Z,\theta)p(Z)}{q(Z)} \right] && \text{Jensen's inequality} \\
&= \mathbb{E}_q[\log p(\mathbf{Y}|Z,\theta)] - \mathbb{E}_q \left[\log \frac{q(Z)}{p(Z)} \right] \\
&= \mathbb{E}_q[\log p(\mathbf{Y}|Z,\theta)] - \mathsf{KL}\left(q(z) \parallel p(z)\right) \\
&:= \mathsf{ELBO}(q,\theta)
\end{aligned}
$$

- Note that there is equality iif $q(z) = p(z|\mathbf{Y},\theta)$.

Variaional inference

- Instead of maximizing the likelihood, variational inference maximizes the ELBO with respect to q and θ:

$$\left(\hat{q}, \hat{\theta}\right) = \text{argmax}_{q,\theta} \text{ELBO}(q, \theta).$$

- As q is a p.d.f., this maximization is unfeasible (unless $p(z|\mathbf{Y}, \theta)$ is known).
- In variational inference, the family of possible distributions for q is restricted to a "simple" parameteric family;
- For instance, when there is one latent variable per observation, one can set

$$q(z_{1:n}) = \prod_{i=1}^{n} q(z_i),$$

  where $q(z_i)$ is the p.d.f. of a $\mathcal{N}_{d_z}\left(\mu_i, \sigma_i^2 I_{d_z}\right)$.
- Note that choosing a family for q imposes to choose both a dependence structure and parameteric distributions.

Example on PLN

Pose a mean field gaussian family for the approximation of $\mathbf{Z}|Y$:

$$q(\mathbf{Z}) = \prod_{i=1}^{n}\prod_{j=1}^{p} q_{i,j}(z_{i,j}), \text{ where } q_{i,j}(z_{i,j}) : \text{ p.d.f. of } \mathcal{N}\left(\mu_{i,j}, \sigma_{i,j}^2\right)$$

$$\begin{aligned}\text{ELBO}(q,\theta) &= \mathbb{E}_q[\log p(\mathbf{Y}|Z,\theta)] - \text{KL}\left(q(z) \parallel p(z)\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{p}\left(\mathbb{E}_{q_{i,j}}[\log p(Y_{i,j}|Z_{i,j},\theta)] - \text{KL}\left(q_{i,j}(z) \parallel p(z)\right)\right).\end{aligned}$$

Example on PLN

Pose a mean field gaussian family for the approximation of $\mathbf{Z}|Y$:

$$q(\mathbf{Z}) = \prod_{i=1}^{n}\prod_{j=1}^{p} q_{i,j}(z_{i,j}), \text{ where } q_{i,j}(z_{i,j}) : \text{ p.d.f. of } \mathcal{N}\left(\mu_{i,j}, \sigma_{i,j}^2\right)$$

$$\begin{aligned}\text{ELBO}(q,\theta) &= \mathbb{E}_q[\log p(\mathbf{Y}|Z,\theta)] - \text{KL}\left(q(z) \parallel p(z)\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{p}\left(\mathbb{E}_{q_{i,j}}[\log p(Y_{i,j}|Z_{i,j},\theta)] - \text{KL}\left(q_{i,j}(z) \parallel p(z)\right)\right).\end{aligned}$$

- For a current estimate $\hat{\theta}^{(0)}$, maximizing w.r.t. q results in $n \times p$ maximizations of the functions

$$f(\mu_{i,j}, \sigma_{i,j}^2) = Y_{i,j}\mu_{i,j} - \exp\left((\mathbf{X}\hat{\theta}^{(0)})_{i,j} + \mu_{i,j}^2 + \frac{\sigma_{i,j}^2}{2}\right) - \frac{1}{2}\left(\mu_{i,j}^2 + \sigma_{i,j}^2 - \log\sigma_{i,j}^2\right)$$

- This results in $2 \times n \times p$ parameters for the distribution of $\mathbf{Z}|\mathbf{Y}$.
- Note that no explicit link is made between each estimated $(\mu_{i,j}, \sigma_{i,j}^2)$ especially, the dependence on $Y_{i,j}$ is not explicit.

Variational autoencoder

- To potentially reduce the number of parameters, and create functional link between $Y_{i,j}$ and $\mu_{i,j}$, one can add an *amortization* constraint:

$$q(\mathbf{Z}) = \prod_{i=1}^{n} \prod_{j=1}^{p} q_{i,j}(z_{i,j}),$$
$$q_{i,j}(z_{i,j}) \sim \mathcal{N}\left(\mu_{i,j}, \sigma_{i,j}^2\right)$$
$$(\mu_{i,j}, \sigma_{i,j}^2) = \mathsf{NeuralNet}(Y_{i,j}|\lambda),$$

where λ is a parameter common to all $Y_{i,j} \mapsto (\mu_{i,j}, \sigma_{i,j}^2)$.

Variational autoencoder

- To potentially reduce the number of parameters, and create functional link between $Y_{i,j}$ and $\mu_{i,j}$, one can add an *amortization* constraint:

$$q(\mathbf{Z}) = \prod_{i=1}^{n} \prod_{j=1}^{p} q_{i,j}(z_{i,j}),$$
$$q_{i,j}(z_{i,j}) \sim \mathcal{N}\left(\mu_{i,j}, \sigma_{i,j}^2\right)$$
$$(\mu_{i,j}, \sigma_{i,j}^2) = \mathsf{NeuralNet}(Y_{i,j}|\lambda),$$

  where λ is a parameter common to all $Y_{i,j} \mapsto (\mu_{i,j}, \sigma_{i,j}^2)$.

- The ELBO becomes a function of (θ, λ). The part to maximize in λ is:

$$f(\lambda) = \sum_{i,j=1}^{n,p} \left\{ Y_{i,j}\mu^{\lambda}(Y_{i,j}) - \exp\left((\mathbf{X}\hat{\theta}^{(0)})_{i,j} + (\mu^{\lambda}(Y_{i,j}))^2 + \frac{\sigma^{\lambda}(Y_{i,j})}{2} \right) \right.$$
$$\left. -\frac{1}{2}\left((\mu^{\lambda}(Y_{i,j}))^2 + \sigma^{\lambda}(Y_{i,j}) - \log \sigma^{\lambda}(Y_{i,j}) \right) \right\}$$

Variational autoencoder

- This idea comes from neural nets generative models;
- This framework enables representation learning (*encoding*) within a generating (*decoding*) model;

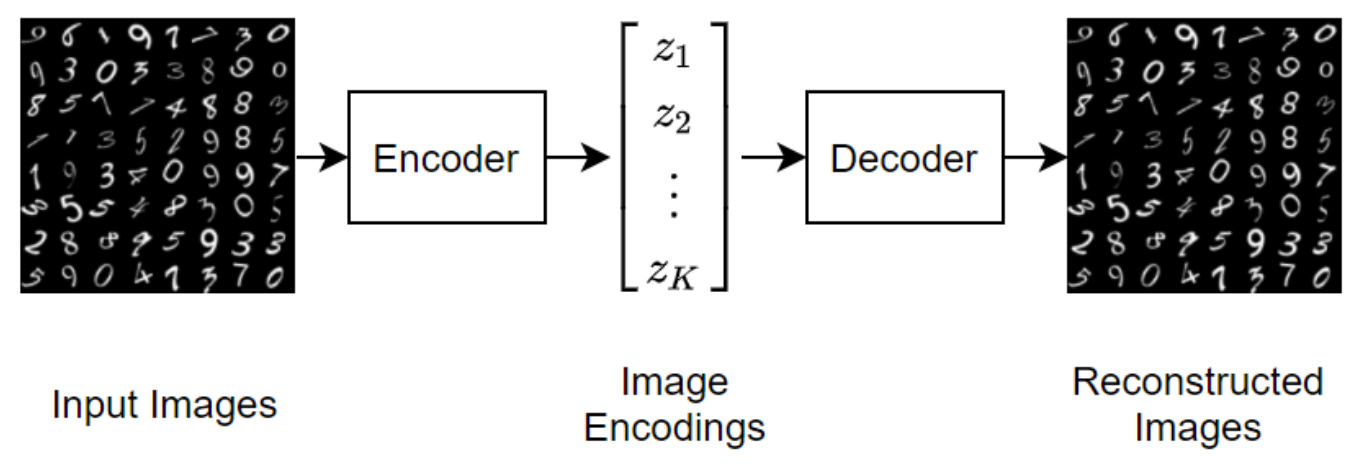


Figure 2: *Source: fr.mathworks.com/*

Gradient of the ELBO

- In general, the ELBO[2]

$$\text{ELBO}(\lambda, \theta) = \mathbb{E}_{q^\lambda}[\log p(\mathbf{Y}|Z, \theta)] - \text{KL}\left(q^\lambda(z) \parallel p(z)\right),$$

  does not have an explicit form, because of the expectation.

- The KL term is not a problem in general;

- Stochastic gradient approach requires to compute

$$\nabla_{\lambda,\theta}\mathbb{E}_{q^\lambda}[\log p(\mathbf{Y}|Z, \theta)] \quad .$$

- For a fixed λ, we have that:

$$\nabla_\theta \mathbb{E}_{q^\lambda}[\log p(\mathbf{Y}|Z, \theta)] = \mathbb{E}_{q^\lambda}[\nabla_\theta \log p(\mathbf{Y}|Z, \theta)] \quad ,$$

  which can be estimated by Monte Carlo, using samples from q^λ.

- For a fixed θ, however:

$$\nabla_\lambda \mathbb{E}_{q^\lambda}[\log p(\mathbf{Y}|Z, \theta)] \neq \mathbb{E}_{q^\lambda}[\nabla_\lambda \log p(\mathbf{Y}|Z, \theta)] \quad ,$$

  as the integration is w.r.t. q^λ.

[2] We now denote q^λ to higlight the dependence of the distribution in this parameter

Gradient of the ELBO (2)

$$\nabla_\lambda \mathbb{E}_{q^\lambda}[\log p(\mathbf{Y}|Z,\theta)] = \nabla_\lambda \int_{\Omega_\mathbf{Z}} \log p(\mathbf{Y}|Z,\theta) q^\lambda(z) \mathrm{d}z$$
$$\neq \mathbb{E}_{q^\lambda}[\nabla_\lambda \log p(\mathbf{Y}|Z,\theta)] \quad ,$$

- Nonetheless, using the fact $\nabla_\lambda q^\lambda = q^\lambda \times \nabla_\lambda \log q^\lambda$, we have that:

$$\nabla_\lambda \mathbb{E}_{q^\lambda}[\log p(\mathbf{Y}|Z,\theta)] = \mathbb{E}_{q^\lambda}[\nabla_\lambda \log p(\mathbf{Y}|Z,\theta)]$$
$$+ \mathbb{E}_{q^\lambda}\left[\log p(\mathbf{Y}|Z,\theta) \times \nabla_\lambda \log q^\lambda(\mathbf{Z})\right] \quad ,$$

  which can be estimated by Monte Carlo, using samples from q^λ.
- This direct Monte Carlo estimator can have high variance;
- Different variance reduction techniques can be used;
- A popular one is the reparametrization trick:

Reduction of variance through the reparametrization trick

- Suppose that you can write

$$\mathbf{Z} = r(\varepsilon, \lambda) \quad ,$$

  where ε is a random variable whose distribution does not depend on λ, and $r(\cdot, \lambda)$ is a known function
- For instance, if $\mathbf{Z} \sim \mathcal{N}\left(\lambda_1, \lambda_2^2\right)$, take $\varepsilon \sim \mathcal{N}(0, 1)$, and $r(\varepsilon, \lambda) = \lambda_1 + \lambda_2 \varepsilon$.
- In this case:

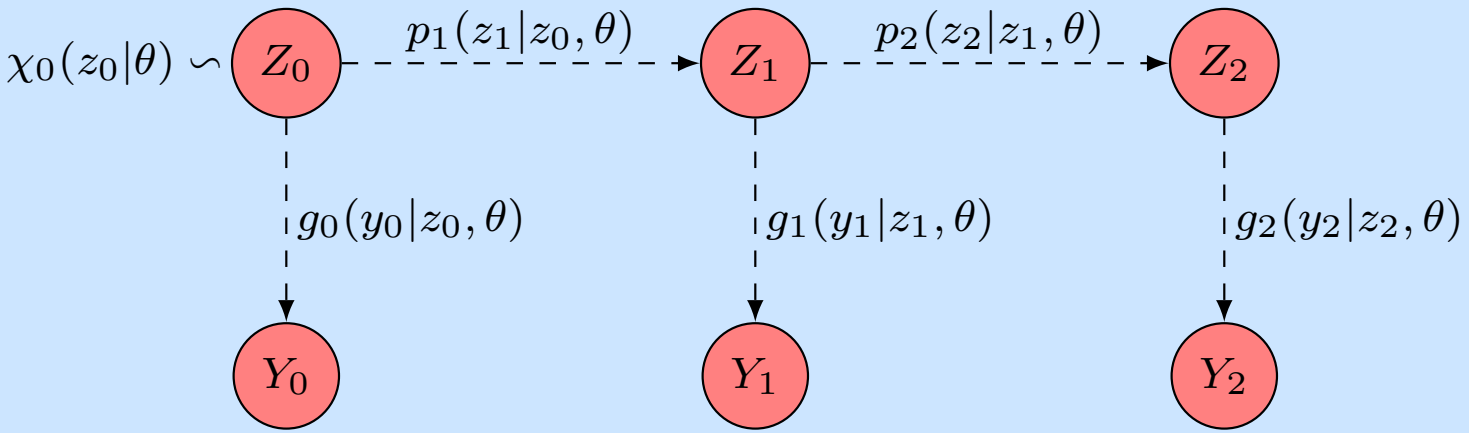
$$\nabla_\lambda \mathbb{E}_{q^\lambda}[\log p(\mathbf{Y}|Z, \theta)] = \mathbb{E}_\varepsilon[\nabla_\lambda \log p(\mathbf{Y}|r(\varepsilon, \lambda), \theta)] \quad ,$$

  which can be estimated without bias, by Monte Carlo sampling, using samples from ε.

Variational inference and Hidden Markov models

Back to HMM

Hidden Markov models



$$
\begin{aligned}
Z_0 &\sim \chi_0(z_0,\theta) \\
Z_t|\{Z_{t-1}=z_{t-1}\} &\sim p_t(z_k|z_{k-1},\theta)
\end{aligned}
$$

Prior distribution

$$Y_t|\{Z_t=z_t\} \sim g_t(y_t|z_t,\theta)$$

Observation model

Posterior distribution in HMM

In this context, the posterior distribution $Z_{0:n}|Y_{0:n}$ satisfies the following factorization:

$$p_{0:n}(z_{0:n}|Y_{0:n},\theta) := p(z_n|Y_{0:n},\theta)\prod_{t=1}^{n} p(z_{t-1}|z_t, y_{0:(t-1)},\theta)$$

and has the structure of Markov chain.

Variational HMM, or VHMM

Variational HMM, or VHMM

Variational family for HMM

- Idea (Campbell et al. (2021)), parameterize the variational with the same backward decomposition as the target distribution:

$$q^{\lambda}_{0:n}(z_{0:n}) = q^{\lambda}_{n}(z_n) \prod_{k=1}^{n} q^{\lambda}_{t-1|t}(z_{k-1}|z_k).$$

- How can we parameterize the distribution of each brick for efficient estimation?

Variational kernels

- For each $t \geqslant 0$, $q_t^\lambda(z_t)$ aims at mimicking the filtering distribution (i.e. $Z_t|Y_{0:t}$);
- We impose that $q_t^\lambda(z_t)$ is the p.d.f. of a gaussian distribution with natural parameter η_t^λ defined in the following way:
  - $\eta_0 = \text{NeuralNet}(h_0|\lambda)$, where $\text{NeuralNet}(y_0|\lambda)$;
  - For $t > 0$, $\eta_t = \text{NeuralNet}(h_t|\lambda)$, where $\text{NeuralNet}(h_{t-1}, y_t|\lambda)$.
- This recursion (called a recurrent neural network), mimicks the predict-update steps of filtering recursions in HMM.

[3] Recall that $p(z_{t-1}|z_t, y_{0:t-1}, \theta) \propto p(z_{t-1}|y_{0:t-1}, \theta)p(z_t|z_t, \theta)$

Variational kernels

- For each $t \geqslant 0$, $q_t^\lambda(z_t)$ aims at mimicking the filtering distribution (i.e. $Z_t|Y_{0:t}$);
- We impose that $q_t^\lambda(z_t)$ is the p.d.f. of a gaussian distribution with natural parameter η_t^λ defined in the following way:
  - $\eta_0 = \text{NeuralNet}(h_0|\lambda)$, where $\text{NeuralNet}(y_0|\lambda)$;
  - For $t > 0$, $\eta_t = \text{NeuralNet}(h_t|\lambda)$, where $\text{NeuralNet}(h_{t-1}, y_t|\lambda)$.
- This recursion (called a recurrent neural network), mimicks the predict-update steps of filtering recursions in HMM.
- Impose $q_{t-1|t}^\lambda(z_{k-1}|z_k)$ to be the p.d.f. of a gaussian distribution with parameter $\eta_{t-1|t}^\lambda$;
- To link[3] $q_{t-1|t}^\lambda(z_{k-1}|z_k)$ to q_{t-1}^λ and z_t, we impose that:

$$\eta_{t-1|t}^\lambda = \eta_{t-1}^\lambda + \vec{\eta}_t^\lambda,$$

where $\vec{\eta}_t^\lambda = \text{NeuralNet}(z_t|\lambda)$ is a gaussian natural parameter.

[3] Recall that $p(z_{t-1}|z_t, y_{0:t-1}, \theta) \propto p(z_{t-1}|y_{0:t-1}, \theta)p(z_t|z_t, \theta)$

Variational learning for HMM

- This allows to learn representation of times series;
- This factorization is prone to online learning (i.e. learning of (λ, θ) seeing each y_t once);
- **Example model:** Chaotic recurrent neural network with dimension 20 for both X and Y

$$X_0 \sim \mathcal{N}(0, Q), X_t = X_{t-1} + \frac{\Delta}{\tau}\left(\gamma W \tanh\left(X_{t-1}\right) - X_{t-1}\right) + \eta,\ t \geq 1,\ \eta \sim \mathcal{N}(0, Q)$$
$$Y_t = X_t + \epsilon,\ \epsilon \sim St(2),\ t \geq 0,$$

Variational learning for HMM

- Online learning of the law of $X_{0:n}|Y_{0:n}$;
- Learnt on a train time series, used to represent a test time series.

Sequence	Smoothing RMSE	Filtering RMSE
Training	0.281	0.311
Eval	0.278 ($\pm$ 0.01)	0.305 ($\pm$ 0.014)

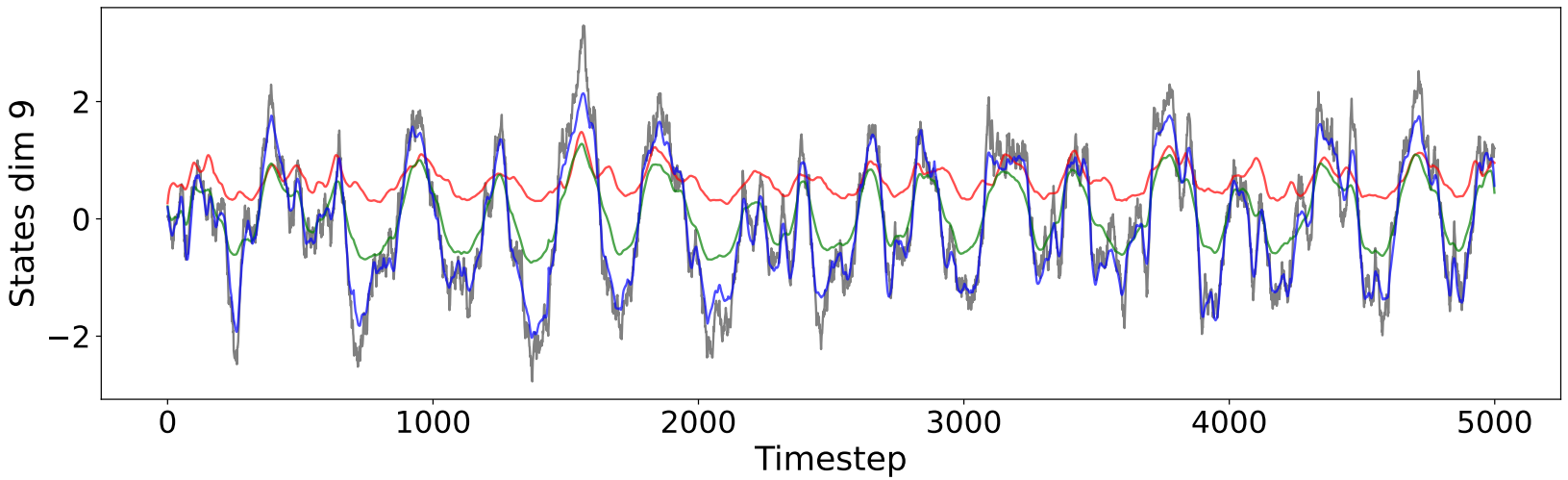


Figure 3: *Black:* True hidden signal, *Red, Green, Blue:* Estimated signal after processing of 1, 10000, 100000 observations

Conclusion

- Neural networks are useful tools to design non linear functions;
- Variational autoencoder is classical variational inference with an additional constraints;
- These ideas are useful bricks that can be implemented for many classical latent variable models.

References

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe. 2017. "Variational Inference: A Review for Statisticians." *Journal of the American Statistical Association*, no. just-accepted.

Campbell, Andrew, Yuyang Shi, Thomas Rainforth, and Arnaud Doucet. 2021. "Online Variational Filtering and Parameter Learning." In *Advances in Neural Information Processing Systems*, 34:18633–45.

Kingma, Diederik P, and Max Welling. 2013. "Auto-Encoding Variational Bayes." *arXiv Preprint arXiv:1312.6114*.

Mallat, Stéphane. 2019. "L'apprentissage Par Réseaux de Neurones Profonds." Cours du collège de France, https://www.college-de-france.fr/fr/agenda/cours/apprentissage-par-reseaux-de-neurones-profonds.