

Prédiction probabiliste de séries temporelles multivariées de mobilité avec des modèles basés sur l'apprentissage profond

Spring school Data Science
Centrale Casablanca - 8-12 Mai 2023

Paul DE NAILLY



Plan de présentation

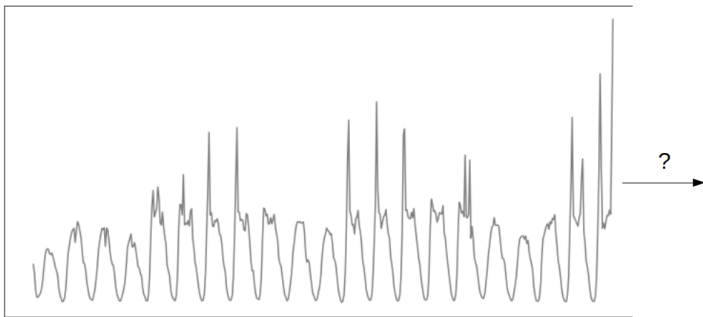
- 1 Contexte et objectifs
- 2 Quelques rappels : Deep Learning
- 3 Les réseaux récurrents pour la prédiction probabiliste
- 4 Présentation de différentes architectures de modèles
- 5 Entraînement et évaluation
- 6 Conclusions

Présentation disponible sur : <https://github.com/pdenailly/Atelier-Summer-School>

Contexte et objectifs

Contexte

La prédiction de séries temporelles

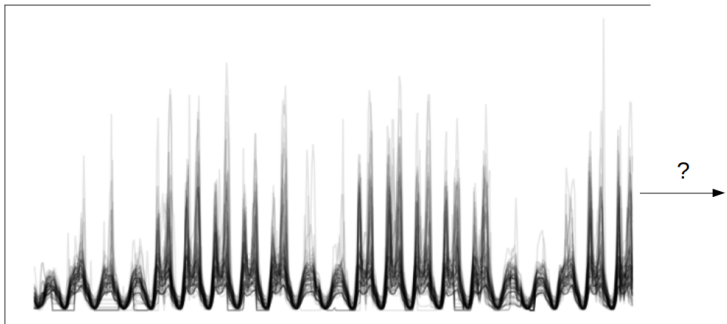


Des méthodes pour la prédiction de séries temporelles

- Modèles de régression linéaire
- Modèles autorégressifs : AR, ARMA, ARIMA, SARIMA
- Machine learning : méthodes à noyau, forêts aléatoires
- Deep learning : Réseaux de neurone récurrents (RNNs)

Contexte

Problème des séries multivariées - grandes dimensions

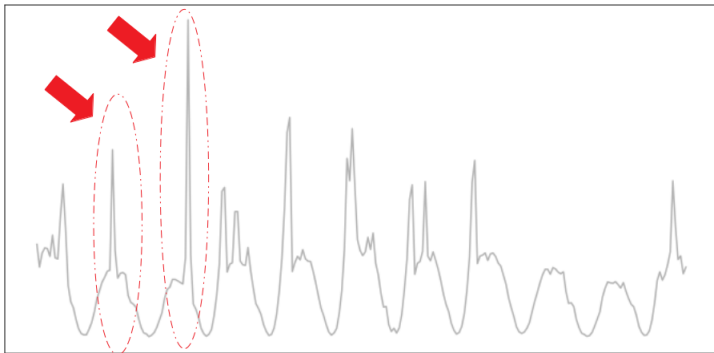


Si la série est multivariée (dimension N) :

- 1 Dépendances entre séries \rightarrow Matrice de covariance (Σ)
- 2 Equivaut à estimer $O(N^2)$ paramètres, potentiellement variables dans le temps (si Σ pleine).

Contexte

Problème des prédictions probabilistes - avec incertitude



De nombreuses situations où prédire sous un certain seuil de risque est plus pertinent !

Prédire des enveloppes de confiance au lieu de prédictions déterministes ?

Prédiction dans un cadre probabiliste de séries temporelles multivariées et corrélées avec des méthodes basées sur le *deep learning*, en utilisant les travaux proposés par [Salinas et al., 2019].

Dans cet atelier :

- 1 **Une présentation** des architectures pour la prédiction probabiliste de séries multivariées, leurs variantes, les hyperparamètres, etc..
- 2 **Un cas pratique** avec la librairie GluonTS de Python, appliqué à des données de trafic de vélos.

Quelques rappels : Deep Learning

Rappels : Deep Learning

Principe général d'un réseau de neurones : passer d'un ensemble de données en entrée à des sorties *via* le calcul de représentations des entrées en couches cachées successives. Les poids de passage d'une couche à l'autre doivent être estimés.

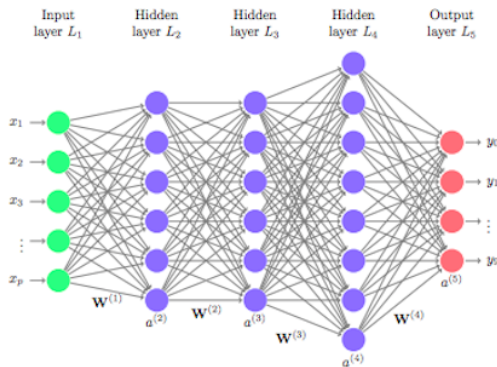
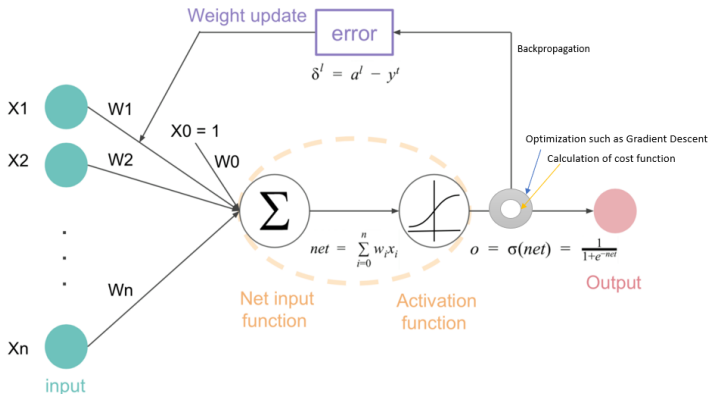


Image : UC Business Analytics R Programming Guide

Rappels : Deep Learning

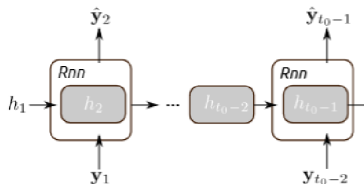
Estimation des poids : backpropagation.



Test de modèles avec différentes valeurs d'hyperparamètres : nombre de couches, taux d'apprentissage, etc..

Rappels : Deep Learning

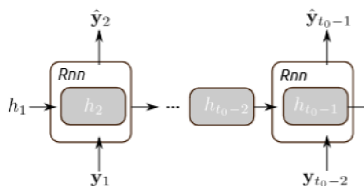
Les réseaux de neurone récurrents (RNNs)



- Adaptés à la prédiction de séries temporelles
- Différents types de cellules possibles : LSTM ou GRU

Rappels : Deep Learning

Les réseaux de neurone récurrents (RNNs)



- Hyperparamètres :

- ▶ Pas d'apprentissage
- ▶ Taille de couche cachée
- ▶ Nombre de couches
- ▶ Dropout
- ▶ Taille de lot (batch)

Les RNNs pour la prédiction probabiliste

Formalisation

Soit $y_{i,t}$ une valeur/un comptage en un point i (avec $i \in \{1, \dots, N\}$) et une tranche de temps t .

On peut écrire $\mathbf{y}_t = \{y_{1,t}, \dots, y_{N,t}\}$.

But

Estimer la distribution $p(\mathbf{y}_{t_0:T} | \mathbf{y}_{1:t_0-1}, \boldsymbol{\chi}_{1:T})$, les futures distributions de probabilité des séries temporelles de t_0 à T connaissant les données jusqu'à $t_0 - 1$ et les données exogènes $\boldsymbol{\chi}_{1:T}$.

Réseau de neurone récurrent probabiliste

Prédiction probabiliste des séries en t , \mathbf{y}_t , suivant le système :

$$\begin{aligned}\mathbf{h}_t &= f(\mathbf{U}\mathbf{z}_t + \mathbf{W}\mathbf{h}_{t-1}), \\ \mathbf{y}_t | \mathbf{h}_t &\sim g(\mathbf{y}_t; \Phi(\mathbf{V}\mathbf{h}_t))\end{aligned}$$

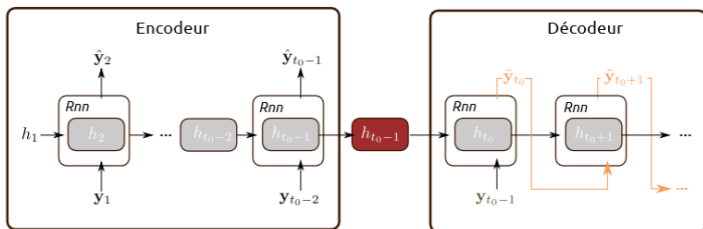
avec $\mathbf{z}_t = \{\mathbf{y}_{t-1}, \mathbf{x}_t\}$.

- $g \implies$ modèle de distribution des comptages.
- $\mathbf{V}\mathbf{h}_t \implies$ ensemble des paramètres de la distribution g .
- Besoin d'estimer les matrices \mathbf{U} , \mathbf{W} et \mathbf{V} .

Réseau de neurone récurrent probabiliste ⚙️

Utilisation d'une architecture de type Encodeur-Décodeur.

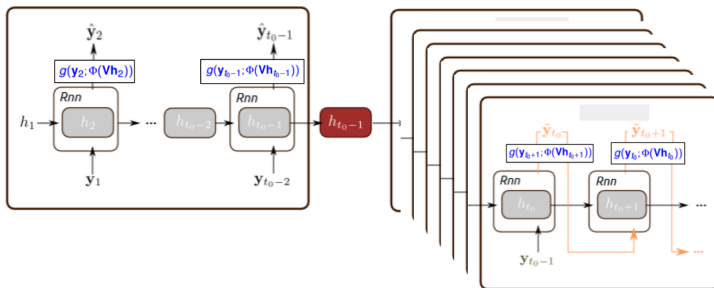
- L'**encodeur** intègre en entrée une séquence d'observations, et la transforme en "contexte" (la dernière couche cachée).
- Le **décodeur** récupère ensuite la sortie émise par l'encodeur, pour l'associer à un ensemble d'observations.



Réseau de neurone récurrent probabiliste ⚙️

Utilisation d'une architecture de type Encodeur-Décodeur.

- L'encodeur
- Le décodeur



- **Cadre probabiliste** : y tirés des distributions g + échantillons de Monte Carlo sur le décodeur pour obtenir un échantillon de prédictions.

Présentation de quelques architectures de modèles issues de l'article de [Salinas et al., 2019]

Architecture de base

Encodeur-décodeur avec une distribution **gaussienne multivariée** en sortie.

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{z}_t + \mathbf{W}\mathbf{h}_{t-1})$$

$$\mathbf{x}_t | \mathbf{h}_t \sim \mathcal{N}(\mathbf{x}_t; \mu(\mathbf{h}_t), \Sigma(\mathbf{h}_t)) \Rightarrow \text{gaussienne multivariée}$$

Architecture de base

Encodeur-décodeur avec une distribution **gaussienne multivariée** en sortie.

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{z}_t + \mathbf{W}\mathbf{h}_{t-1})$$

$$\mathbf{x}_t | \mathbf{h}_t \sim \mathcal{N}(\mathbf{x}_t; \mu(\mathbf{h}_t), \Sigma(\mathbf{h}_t)) \Rightarrow \text{gaussienne multivariée}$$

Architecture de base

Encodeur-décodeur avec une distribution **gaussienne multivariée** en sortie.

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{z}_t + \mathbf{W}\mathbf{h}_{t-1})$$

$$\mathbf{x}_t | \mathbf{h}_t \sim \mathcal{N}(\mathbf{x}_t; \mu(\mathbf{h}_t), \Sigma(\mathbf{h}_t)) \implies \text{gaussienne multivariée}$$

avec :

- $\mathbf{x}_t = m(\mathbf{y}_t)$, une transformation de \mathbf{y}_t avec une fonction $m(\cdot)$.

Architecture de base

Encodeur-décodeur avec une distribution **gaussienne multivariée** en sortie.

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{z}_t + \mathbf{W}\mathbf{h}_{t-1})$$

$$\mathbf{x}_t | \mathbf{h}_t \sim \mathcal{N}(\mathbf{x}_t; \mu(\mathbf{h}_t), \Sigma(\mathbf{h}_t)) \implies \text{gaussienne multivariée}$$

avec :

- $\mu(\mathbf{h}_t) = (\mu_1(h_{1,t}), \dots, \mu_N(h_{N,t}))$ des **transformations** de \mathbf{h}_t en paramètres de **moyenne** de la gaussienne.
- $\Sigma(\mathbf{h}_t) = (\Sigma_1(h_{1,t}), \dots, \Sigma_N(h_{N,t}))$ des **transformations** de \mathbf{h}_t en paramètres de **covariance** de la gaussienne.

Architecture de base

Encodeur-décodeur avec une distribution **gaussienne multivariée** en sortie.

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{z}_t + \mathbf{W}\mathbf{h}_{t-1})$$
$$\mathbf{x}_t | \mathbf{h}_t \sim \mathcal{N}(\mathbf{x}_t; \mu(\mathbf{h}_t), \Sigma(\mathbf{h}_t)) \implies \text{gaussienne multivariée}$$

Plusieurs variantes de modèles au niveau de :

- Prise en compte de la matrice de covariance Σ ,
- Transformation des données en entrée $m(\cdot)$.

Variantes du modèle

Gestion de la matrice de covariance Σ

- **Première stratégie : matrice pleine**

- ▶ $\Sigma(h_t) \implies$ matrice symétrique $N \times N$.
- ▶ $O(N^2)$ paramètres à estimer.

Variantes du modèle

Gestion de la matrice de covariance Σ

- **Première stratégie : matrice pleine**

- ▶ $\Sigma(h_t) \implies$ matrice symétrique $N \times N$.
- ▶ $O(N^2)$ paramètres à estimer.

- **Deuxième stratégie : matrice de faible rang + diagonale**

- ▶ $\Sigma(h_t) = D(h_t) + J(h_t)J(h_t)^T$
avec D une matrice diagonale ($N \times N$), et J une matrice ($N \times r$).
- ▶ $O(N \times r)$ paramètres à estimer, r peut être choisis petit.

$$\Sigma(\mathbf{h}_t) = \begin{bmatrix} d_1(\mathbf{h}_{1,t}) & & 0 \\ & \ddots & \\ 0 & & d_N(\mathbf{h}_{N,t}) \end{bmatrix} + \begin{bmatrix} j_1(\mathbf{h}_{1,t}) \\ \vdots \\ j_N(\mathbf{h}_{N,t}) \end{bmatrix} \begin{bmatrix} j_1(\mathbf{h}_{1,t}) & \vdots & j_N(\mathbf{h}_{N,t}) \end{bmatrix}^T$$

Variantes du modèle

Transformation des données y en entrée

Deux problèmes possibles :

- 1 Les séries peuvent suivre une **loi de puissance** : beaucoup de séries aux valeurs faibles, quelques séries avec de fortes valeurs.
- 2 Les séries peuvent ne pas être distribuées normalement.

Comment y répondre ?

Variantes du modèle

Transformation des données y en entrée

Passage $y \xrightarrow{m(.)} x$ via deux stratégies :

- **Première stratégie : Transformation par la moyenne**

Chaque y est divisé par la moyenne de chaque série.

- **Deuxième stratégie : Transformation via des Copules Gaussiennes**

- ▶ Transforme les séries pour que marginalement, elles suivent des distributions gaussiennes.
- ▶ Passage des observations y à des variable aléatoire uniforme u (via la fonction de répartition) puis des variables gaussiennes x .

Variantes du modèle

Passage par un processus gaussien GP

Il est possible d'utiliser un processus gaussien en sortie :

$$x_{i,t} \sim GP(x_{i,t}; \tilde{\mu}(h_{i,t}), \tilde{d}(h_{i,t}), \tilde{j}(h_{i,t}))$$

Avantage :

- Les paramètres des transformations $\tilde{\mu}$, \tilde{d} et \tilde{j} sont partagés entre toutes les séries
- Possibilité d'apprendre ces paramètres sur un petit nombre de séries à chaque itération d'apprentissage. On évite ainsi le sur-apprentissage et on gère bien les grandes dimensions.

Variantes du modèle

Récapitulatif

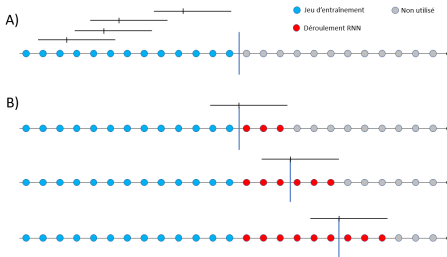
Modèle	Architecture	Covariance	Transf.
Vec-LSTM-fullrank-scaling	LSTM global	Pleine	Moyenne
Vec-LSTM-lowrank-Copula	LSTM global	Faible rang	Copules
GP-Copula	GP	Faible rang	Copules

Nous testerons ces trois modèles dans le cas pratique avec GluonTS.

Entraînement et évaluation

Entraînement (A)/Evaluation (B)

Grille d'hyperparamètres non estimés par le modèle : nombre de couches cachées, pas d'apprentissage, taille de batch, etc.

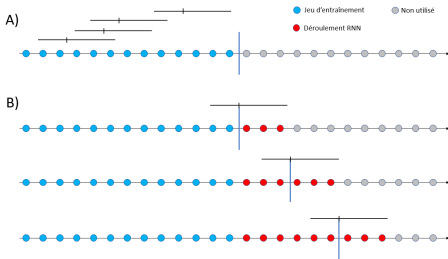


- Pour chaque jeu d'hyperparamètres, estimation de \mathbf{U} , \mathbf{W} et \mathbf{V} via la maximisation de la log vraisemblance $L_Y(\mathbf{U}, \mathbf{V}, \mathbf{W})$:

$$L_Y(\mathbf{U}, \mathbf{V}, \mathbf{W}) = \frac{1}{|Ba|(T)} \sum_{Ba=1}^T \sum \log p(\mathbf{y}_t | \mathbf{h}_t, \mathbf{x}_t)$$

Entraînement (A)/Evaluation (B)

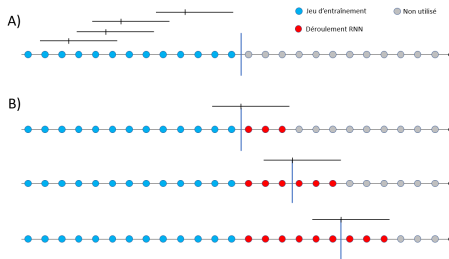
Grille d'hyperparamètres non estimés par le modèle : nombre de couches cachées, pas d'apprentissage, taille de batch, etc.



- Augmentation des données : à chaque batch plusieurs fenêtres de tailles T échantillonnées aléatoirement dans la base d'entraînement.

Entraînement (A)/Evaluation (B)

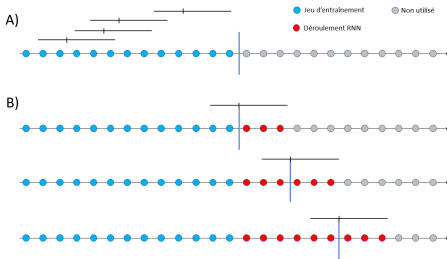
Grille d'hyperparamètres non estimés par le modèle : nombre de couches cachées, pas d'apprentissage, taille de batch, etc.



- Couverture de l'ensemble de la base de validation/test avec une fenêtre glissante.

Entraînement (A)/Evaluation (B)

Grille d'hyperparamètres non estimés par le modèle : nombre de couches cachées, pas d'apprentissage, taille de batch, etc.



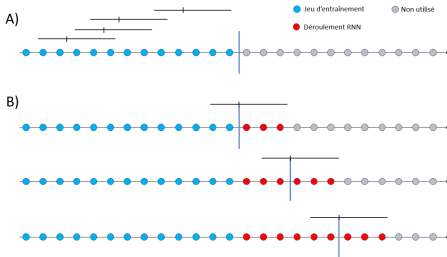
- Calcul de métriques moyennées sur les M fenêtres pour comparer observations et prédictions.

$$MSE = \frac{1}{N(T - t_0 + 1)} \sum_{i,t} (y_{i,t} - \hat{y}_{i,t})^2$$

avec $\hat{y}_{i,t}$ une prédiction.

Entraînement (A)/Evaluation (B)

Grille d'hyperparamètres non estimés par le modèle : nombre de couches cachées, pas d'apprentissage, taille de batch, etc.



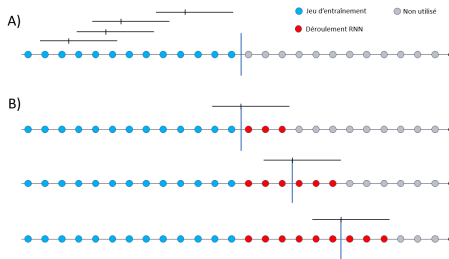
- Calcul de métriques moyennées sur les M fenêtres pour comparer observations et prédictions.

$$CRPS(F, y) = \int_{-\infty}^{+\infty} (F(\alpha) - 1(y \leq \alpha))^2 d\alpha$$

avec F la distribution cumulée prédite, calculée empiriquement et $1(y \leq \alpha)$ une fonction indicatrice qui vaut 1 si $y \leq \alpha$.

Entraînement (A)/Evaluation (B)

Grille d'hyperparamètres non estimés par le modèle : nombre de couches cachées, pas d'apprentissage, taille de batch, etc.



- Comparaison des modèles avec différentes architectures et/ou différents hyperparamètres.

Conclusions

- Modèles adaptés à la prédiction probabiliste de séries multivariées et corrélées.
- Capacité à prendre en compte des séries de grande dimension grâce aux matrices de covariance de faible rang et aux processus gaussiens (GP).

Atelier pratique

- 1 Ouvrez le dépôt github du cas pratique à l'adresse suivante : <https://github.com/pdenailly/Atelier-Summer-School>.
- 2 Suivez le notebook python *Prédictions_Vélos.ipynb* :
 - ▶ Sur google colab, ouvrez le lien colab. Vous pouvez enregistrer une copie modifiable
 - ▶ En local avec jupyter-notebook.
- 3 Importez les données *bike_data.csv* et le fichier python *rolling_dataset.py* dans le répertoire courant du projet (sur colab ou en local si vous utilisez jupyter).

Références I

David Salinas, Michael Bohlke-Schneider, Laurent Callot, Roberto Medico, and Jan Gasthaus. High-dimensional multivariate forecasting with low-rank gaussian copula processes. *Advances in neural information processing systems*, 32, 2019.