Prédiction de trafic réseau : Comparaison de modèles ARIMA, SVR et RNN

Mohamed Khoukh
Sidi Mohamed Raid Ghallabi
Maël Chan Peng
INP-ENSEEIHT
mael.chanpeng@etu.toulouse-inp.fr
sidimohamedraid.ghallabi@etu.toulouse-inp.fr
mohammed.khoukh@etu.toulouse-inp.fr

***Abstract*—La prédiction de trafic réseau constitue un enjeu clé pour l'optimisation de l'allocation dynamique des ressources, notamment dans les architectures satellites. Ce rapport compare trois approches de prévision de séries temporelles : ARIMA (modèle statistique), SVR (Support Vector Regression), et RNN (Réseaux de Neurones Récurrents). L'objectif est de mesurer leur précision prédictive et leur complexité, selon différents horizons temporels. Nous montrons notamment que les RNN sont bien adaptés aux données séquentielles non linéaires, malgré une phase d'apprentissage plus coûteuse.**

I. Méthode 1 : SVR - Support Vector Regression

A. Principe général

Le Support Vector Regression (SVR) est une extension du SVM (Support Vector Machine) pour des tâches de régression. Contrairement au SVM qui cherche une frontière de décision, le SVR cherche à approximer une fonction continue $f(x)$ en maximisant une marge d'erreur tolérable ε autour de cette fonction. L'objectif est donc de trouver une fonction $f(x)$ qui dévie le moins possible des vraies valeurs, tout en autorisant une certaine tolérance à l'erreur.

$$f(x) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x_i, x) + b \tag{1}$$

où $K(x_i, x)$ est une fonction noyau mesurant la similarité entre l'entrée x et les points d'entraînement x_i, α_i et α_i^* sont les coefficients associés aux vecteurs de support, et b est un biais.

B. Choix du noyau

Nous avons testé plusieurs noyaux pour adapter le modèle à la non-linéarité du trafic réseau :

- **Linéaire** : efficace pour des données simples ou faiblement non-linéaires.
- **Polynomiale** : permet de capturer des relations plus complexes.
- **RBF (Radial Basis Function)** : le plus performant dans notre cas, car il capture des non-linéarités fines du trafic.

La figure **??** illustre l'impact du choix du noyau dans un modèle SVR. Trois types de noyaux sont comparés :

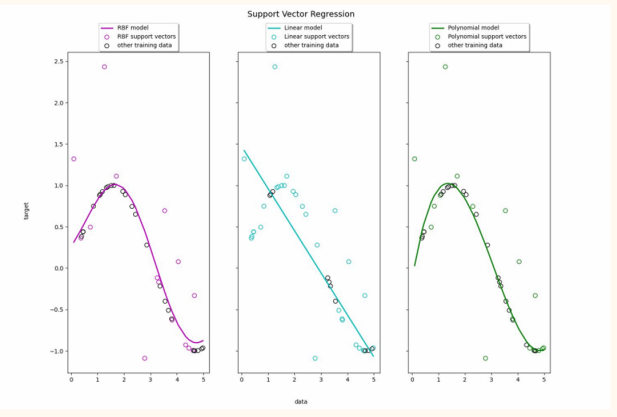

Figure 1: Comparaison entre 3 types de noyaux

- **RBF (Radial Basis Function)** : Le modèle (à gauche) capture efficacement les non-linéarités présentes dans les données. Il s'adapte de manière souple aux courbures du signal, ce qui le rend particulièrement adapté aux séries temporelles complexes comme le trafic réseau.
- **Linéaire** : Le modèle (au centre) suppose une relation linéaire entre les variables. Il est rapide à entraîner, mais son pouvoir prédictif est limité dès que les données s'écartent de cette hypothèse. Les écarts entre les prédictions et les valeurs réelles sont ici plus marqués.
- **Polynomial** : Le modèle (à droite) parvient à modéliser des relations non-linéaires, mais peut souffrir d'un surapprentissage (overfitting) si le degré du polynôme est mal choisi. Dans ce cas, il suit bien la tendance globale, mais montre une plus grande sensibilité au bruit.

Les cercles pleins représentent les vecteurs de support, c'est-à-dire les points ayant un rôle direct dans la définition de la fonction de prédiction. On observe que chaque noyau sélectionne un ensemble de vecteurs différents, ce qui influence directement la forme de la courbe prédite. Dans nos expériences sur le trafic satellite, le noyau RBF a offert le meilleur compromis entre précision et robustesse.

C. Implementation de l'algorithme SVR avec deux types de noyaux

Nous avons implémenté le modèle SVR en Python avec deux noyaux différents : RBF et polynomial. L'objectif était de prédire le trafic réseau sur une fenêtre temporelle de 500s à 1500s. Les résultats montrent que le noyau RBF offre de meilleures performances globales avec une erreur plus faible. Les prédictions restent plus au moins proches des tendances réelles, malgré la forte variabilité du trafic. Ces résultats confirment la capacité du noyau RBF, à prédire le traffic.

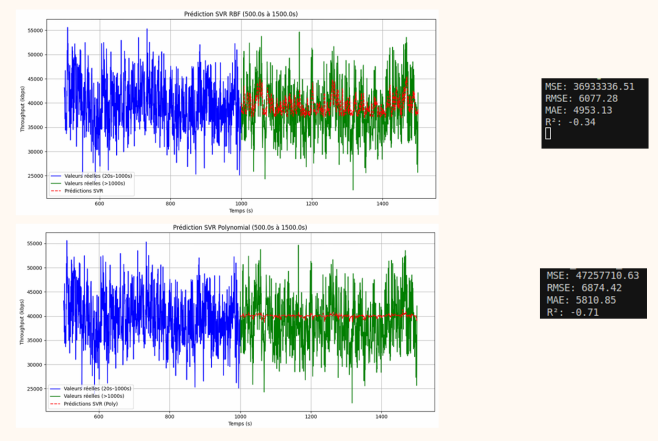

Figure 2: Prédiction du trafic par SVR avec noyaux RBF (haut) et polynomial (bas)

D. Impact de l'horizon temporel

Nous avons testé l'influence du paramètre de *lag* dans le modèle SVR avec noyau RBF. Le *lag* détermine la profondeur temporelle utilisée pour prédire l'instant t. Trois valeurs ont été comparées : 1 (0.1s), 10 (1s) et 100 (10s).

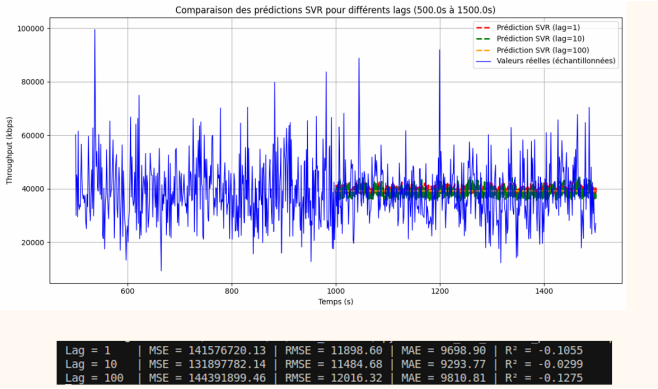

Figure 3: Comparaison des prédictions SVR pour différents lags (fenêtre 500s à 1500s)

Les résultats montrent que le lag = 10 (soit un historique d'1 seconde) donne les meilleures performances globales :

- **RMSE** = 11484.68, **MAE** = 9293.77, R^2 = -0.03

Cela indique qu'un compromis entre horizon court et long est préférable. Un lag trop faible (lag = 1) ne permet pas de capturer suffisamment d'informations, tandis qu'un lag trop long (lag = 100) introduit du bruit inutile dans les entrées. Ces résultats confirment que le choix du lag a un impact direct sur la qualité de prédiction.

E. Application aux différents scenarios

Pour évaluer la robustesse du modèle SVR, nous avons appliqué l'algorithme avec noyau RBF sur les six scénarios de simulation. Chaque scénario correspond à une configuration réseau différente : faible charge (Scénarios 2 et 3), charge normale (Scénarios 1 et 4), et forte surcharge (Scénarios 5 et 6).

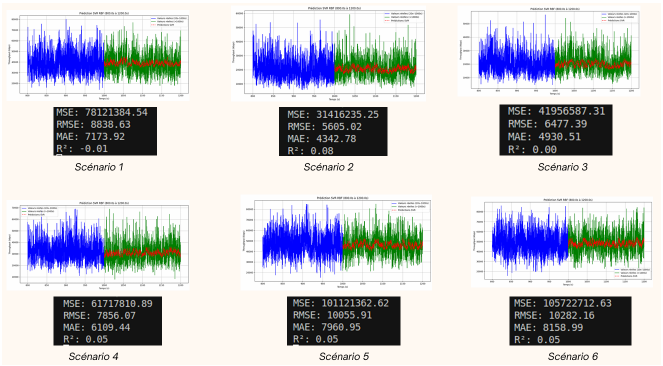

Figure 4: Performances du SVR (noyau RBF) sur les scénarios 1 à 6

Les résultats montrent que le modèle obtient les meilleures performances (plus faible erreur) dans le Scénario 2, où le trafic est stable et peu variable. À l'inverse, les erreurs augmentent sensiblement dans les scénarios à forte charge, notamment le Scénario 6, qui présente une surcharge réseau importante et des pics de trafic élevés. Cela met en évidence la difficulté du SVR à généraliser dans des contextes fortement dynamiques sans réajustement.

II. Méthode 2 : ARIMA

Les modèles auto-régressifs les plus couramment utilisés dans l'analyse de séries temporelles univariées sont les modèles ARMA (Auto-Regressive Moving Average). Ces modèles sont adaptés aux séries temporelles en une seule dimension, généralement étudiées en fonction du temps.

Le modèle ARMA(p, q) combine deux composantes fondamentales :

- Un modèle auto-régressif d'ordre p, noté AR(p),
- Un modèle de moyenne mobile d'ordre q, noté MA(q).

Composante AR(p)

Dans un modèle AR(p), la valeur Y_n de la série à l'instant n est estimée par une combinaison linéaire des p dernières valeurs observées, accompagnée d'un terme d'erreur e_n et d'une constante c :

$$Y_n = c + \sum_{i=1}^{p} \phi_i Y_{n-i} + e_n$$

où ϕ_i sont les paramètres du modèle et p représente l'ordre du modèle.

Composante MA(q)

Le modèle MA(q) utilise quant à lui les erreurs des q instants précédents comme variables explicatives. La formulation générale est :

$$Y_n = \mu + \sum_{i=1}^{q} \theta_i e_{n-i} + e_n$$

où μ est la moyenne de la série, θ_i sont les coefficients de la moyenne mobile, et q est l'ordre du modèle.

Modèle ARMA(p, q)

En combinant les deux composantes précédentes, on obtient le modèle ARMA(p, q) :

$$Y_n = c + \sum_{i=1}^{p} \phi_i Y_{n-i} + \sum_{i=1}^{q} \theta_i e_{n-i} + e_n$$

Ce modèle est applicable uniquement à des séries temporelles stationnaires, c'est-à-dire dont les propriétés statistiques (comme la moyenne et la variance) sont constantes dans le temps.

Extension vers ARIMA(p, d, q)

Cependant, de nombreuses séries réelles présentent une structure non stationnaire, notamment en raison de tendances ou de variations saisonnières. Pour ces cas, on utilise le modèle ARIMA(p, d, q), qui introduit un paramètre d représentant le nombre de différenciations nécessaires pour rendre la série stationnaire.

L'idée fondamentale du modèle ARIMA est d'appliquer d différenciations successives à la série temporelle jusqu'à ce qu'elle devienne stationnaire, puis d'y appliquer un modèle ARMA(p, q).

Remarque

Il est important de noter que, contrairement aux autres méthodes que nous présenterons par la suite, le modèle ARMA(p, q) (et par extension ARIMA) ne fait pas partie de la famille des algorithmes d'apprentissage automatique.

Résultats expérimentaux

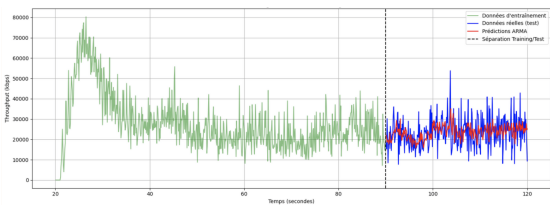

Figure 5: entrainement du modèle Arima et test sur les données du scénario 1

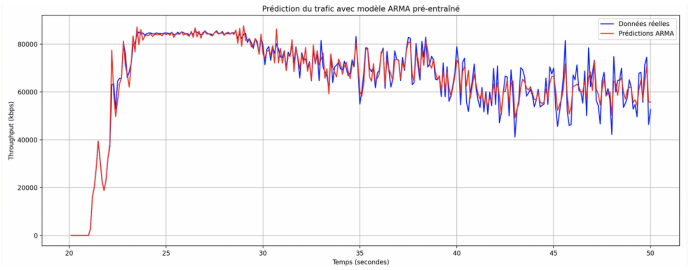

Figure 6: validation du modèle arima sur un nouveau scénario

Les figures ci-dessous illustrent les prédictions du modèle ARIMA comparées aux données réelles de trafic réseau. On peut en tirer plusieurs observations importantes concernant la précision du modèle sur des intervalles de temps courts.

- **Précision à court terme** : Sur des échelles de temps courtes, comme illustré dans la première figure (entre 20 et 50 secondes), le modèle ARIMA pré-entraîné parvient à reproduire de manière fidèle les variations de la série temporelle. Les courbes des prédictions (en rouge) et des données réelles (en bleu) sont très proches, montrant que le modèle est efficace pour capter les dynamiques locales et les fluctuations à court terme.
- **Séparation entraînement/test** : La deuxième figure met en évidence la séparation entre les données d'entraînement et de test. On remarque que dans la zone de test (après 90 secondes), le modèle ARIMA continue à fournir des prédictions relativement cohérentes avec les données réelles, bien qu'on observe un léger décalage et une plus grande variabilité dans les valeurs réelles (en bleu), probablement due à une dynamique plus complexe ou à des événements non captés pendant l'entraînement.
- **Avantages** :
  - Bonne capacité de prédiction à court terme pour des séries stationnaires ou faiblement non-stationnaires.
  - Implémentation simple, peu coûteuse en ressources computationnelles.
  - Ne nécessite pas de jeu de données volumineux pour être efficace.
- **Inconvénients** :
  - Sensibilité aux changements soudains ou non linéaires dans la série temporelle.
  - Moins performant pour des prédictions à long terme, en particulier si la série est influencée par des tendances ou des effets saisonniers complexes.
  - Nécessite un pré-traitement adéquat (stationnarité) et un choix judicieux des paramètres (p, d, q).

On peut conclure donc que le modèle ARIMA montre ici une bonne efficacité pour des prédictions à court terme dans un contexte de trafic réseau. Toutefois, son application reste limitée dès lors que les conditions changent fortement ou que la série présente des comportements non linéaires ou hautement instationnaires.

III. Méthode 3 : RNN - Réseaux de Neurones Récurrents

Les Réseaux de Neurones Récurrents (RNN) sont conçus pour traiter des données séquentielles, en conservant une mémoire des états précédents grâce à des connexions récurrentes. Cette architecture est particulièrement adaptée aux séries temporelles, comme celles liées au trafic réseau. Ils permettent de capturer des dépendances temporelles complexes, souvent non linéaires, que les modèles statistiques classiques peinent à modéliser.

A. Principe des RNN

Un RNN met à jour un état caché à chaque pas de temps t selon l'entrée x_t et l'état précédent h_{t-1} :

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

où σ est une fonction d'activation (tanh, ReLU…), et W_{xh}, W_{hh}, b_h sont les paramètres appris pendant l'entraînement.

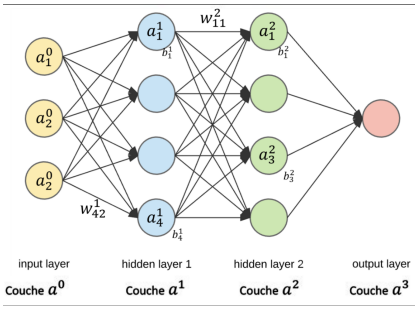

Figure 7: Architecture générale d'un RNN classique : chaque sortie dépend de l'entrée actuelle et de l'état précédent.

B. Résultats expérimentaux

Plusieurs scénarios ont été testés pour analyser l'efficacité des RNN selon deux variables clés :

- **L'horizon de prédiction** : 20s, 60s et 120s.
- **La taille de la fenêtre temporelle** : petite (10s), moyenne (30s), grande (50s).

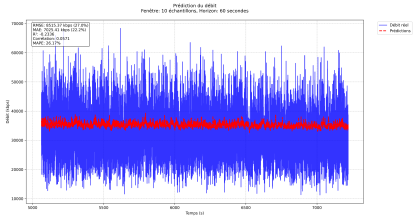

Figure 8: Prédiction d'un horizon de 60 secondes pour une fenêtre temporelle de 10 secondes

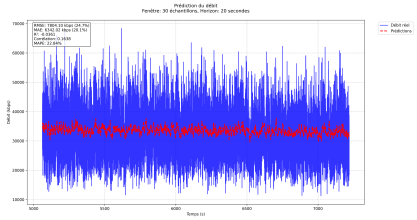

Figure 9: Prédiction d'un horizon de 20 secondes pour une fenêtre temporelle de 30 secondes

C. Avantages et limites

Avantages :

- Très bon comportement pour des séries non linéaires et non stationnaires.
- Capacité à prédire les pics de trafic grâce à la mémoire interne.

Limites :

- Entraînement coûteux en temps et en ressources (surtout sous Python).
- Nécessite un volume important de données représentatives pour généraliser.
- Moins efficace en cas de rupture soudaine du régime si non réentraîné.

D. Temps d'exécution et outils

Les temps de calcul varient en fonction de l'implémentation :

- En Python : entraînement plus long, mais meilleure intégration avec les bibliothèques ML.
- En Matlab : plus rapide pour de petits jeux de données, mais moins flexible.