# Méthodes de type Reservoir Computing pour le traitement des séries temporelles

**Louise Olgiati, Hugo Boulet**

**23/02/2023**

# Table des matières

# 1 Bibliographie


1 **Real-time Reservoir Computing Network-based Systems for Detection Tasks on Visual Contents**,*Azarakhsh Jalalvand, Glenn Van Wallendael, Rik Van de Walle*

2 **Echo State Networks-based Reservoir Computing for MNIST Handwritten Digits Recognition**, Nils Schaetti, Michel Salomon, and Rapha el Couturier

3 **Robust continuous digit recognition using Reservoir Computing**, *Azarakhsh Jalalvand*

4 **Reservoir Computing for Early Stage Alzheimer's Disease Detection**, Frederic Lehmann

# 2 Introduction

Parmi les réseaux de neurones artificielles traitant des informations temporelles, les réseaux de neurones récurrents, tels que les réseaux LSTM, se sont avérés être les plus efficaces. Cependant, entraîner de tels réseaux s'avère aussi extrêmement coûteux.

Alors, de récents travaux se sont penchés sur les réseaux de type **Reservoir Computing (RCN)**, qui possèdent l'avantage d'avoir un entraînement plus robuste (on apprend uniquement la matrice des poids de la couche de sortie), diminuant drastiquement les temps de calculs impliqués.

Un RCN simple comporte deux parties :

- Une couche cachée de neurones, **le reservoir**, possédant des liaisons récurrentes (delayed feedback), caractérisée par les matrices de poids $W_{back}$ et $W_{rec}$.
- Une couche de sortie aux liaisons linéaires caractérisée par sa matrice de poids $W_{out}$.

Le reservoir peut être interprété comme un système dynamique non linéaire qui analyse des flux d'entrées $U_t$. En notant $R_t, U_t$ et $Y_t$ les activations récurrentes, les entrées et les sorties, les équations suivantes résument le fonctionnement du réseau :

$$R_t = (1-\lambda)R_{t-1} + \lambda f_{res}(W_{in}U_t + W_{rec}R_{t-1} + W_{back}Y_{t-1}) \tag{1}$$

$$Y_t = W_{out}R_t \tag{2}$$

Le fonctionnement général est illustré dans la figure ci-dessous :

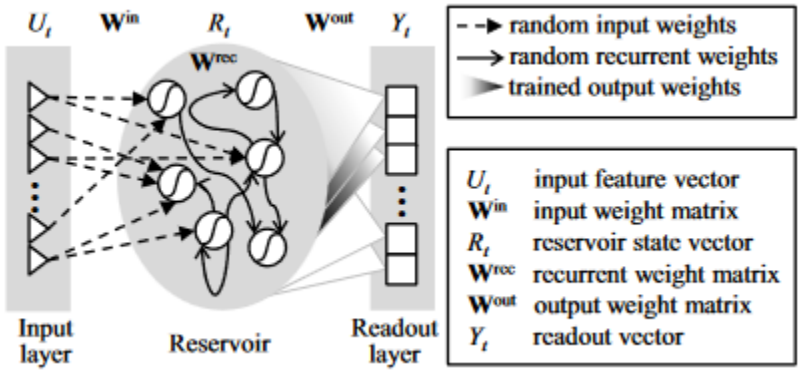


**Figure 1** – Schéma d'un RCN basique

# 3 Etude préliminaire

## 3.1 Objectif

Afin de démontrer l'efficacité des RCN sur la base de données MNIST, nous avons implémenté une méthode de l'état de l'art avec laquelle nous pourrons comparer les résultats. Nous avons donc choisi d'implémenter un réseau de neurones convolutifs, ou CNN (Convolutional Neural Network). Ce type de réseau, très utilisé pour la classification d'images, permet d'extraire les motifs caractéristiques propres à chaque image grâce à une succession de filtres.

## 3.2 Implémentation

Nous avons implémenté ce réseau avec l'API keras de Tensorflow. Pour ce qui est des données MNIST, nous avons utilisé un jeu d'entraînement (50000 données), un jeu de validation (10000 données) et un jeu de test (10000 données). Toutes les images ont été normalisées au préalable.

**Tableau 1** – Architecture du réseau CNN

| Type de couche | Caractéristique | Fonction d'activation |
|---|---|---|
| Conv2D | 32 filtres | reLu |
| MaxPooling2D | Taille (2,2) | |
| Conv2D | 32 filtres | reLu |
| MaxPooling2D | Taille (2,2) | |
| Flatten | | |
| Dense | 100 neurones | reLu |
| Dense | 10 neurones | softmax |

## 3.3 Résultats

Le réseau a été entraîné sur 9 époques (en utilisant EarlyStopping), avec une taille de batch de 32. On obtient une accuracy sur le jeu de test de **98.1%**.

# 4 Objectifs et méthodes utilisées

L'objectif principal de ce travail consiste à démontrer l'efficacité des RCN sur le traitement des séries temporelles par rapport à l'état de l'art (RNN et CNN), sur la base de données classique **MNIST**. Nous étudierons alors le dimensionnement idéal du réseau par rapport au jeu de données et justifierons des meilleurs paramètres et hyperparamètres entrant en jeu.
Ensuite, nous analyserons les différentes manières d'améliorer le calcul des poids de la couche de sortie, $W_{out}$, consistant jusqu'à présent de la pseudo-inversion de l'équation (2) (Moore-Penrose pseudo-inverse).

## 4.1 Présentation du réseau implémenté

**Tableau 2** – Paramètres du réseau

| Parameter | Définition | Valeurs |
|---|---|---|
| $N_{res}$ | Nombre de neurones dans le réservoir | $[0\,;+\infty[$ |
| Spectral radius | Rayon spectral de $W_{rec}$ | $[0\,;1]$ |
| Sparsity | Pourcentage de vide dans $W_{rec}$ | $[0\,;1]$ |
| input shift | Terme additif des entrées | $[0\,;1]$ |
| input scaling | Terme multiplicatif des entrées | $[0\,;1]$ |
| teacher shift | Terme additif des états récurrents | $[0\,;1]$ |
| teacher scaling | Terme multiplicatif des états récurrents | $[0\,;1]$ |
| noise | Bruit additif gaussien | 0,001 |
| leaking rate ($\lambda$) | équation (1) | 0,001 |

Nous implémentons en langage Python le modèle **Echo State Network**, basé sur les équations (1) et (2), en créant une classe **ESN** qui contient en champs les paramètres du tableau 1. Dans ce modèle, la matrice du réservoir $W_{rec}$ est initialisée selon une loi normale centrée, dont on a mis aléatoirement à zéro un nombre (Sparsity) de coefficients.
De plus, les travaux de Jaeger (2001) ont montré que le rayon spectral de $W_{rec}$, noté $s_r$, doit être égal à 1. Les autres matrices du réseau $W_{in}$ et $W_{back}$ sont initialisées aléatoirement. Le

fonctionnement du réseau est résumé dans l'algorithme suivant :

**Algorithme 1 :** Apprentissage de la matrice de sortie

*Entrées*

- $U_t$ = flux d'entrée (colonne d'une image MNIST)
- $R_t$ = flux récurrent
- $Y_t$ = flux de sortie

*Initialisations*

- $W_{rec}(s_r = 1, \beta = 0.5)$
- $W_{in}, W_{back}$ selon loi normale centrée

*Calculs*

**for** $t \leftarrow 0$ **to** $t \leftarrow T$ **do**
  **foreach** $U_t \in images$ **do**
    Calculer chaque état $R_t$ selon (1)
    Mise à jour de $W_{out}$ par régression
  **end**
**end**

*Sorties*

- $W_{out}$ et tous les états récurrents $R_t$

Les deux étapes coûteuses sont le calcul des états récurrents et la mise à jour de $W_{out}$ par régression, et prennent sensiblement le même temps de calcul à nos machines. Nos recherches montrent aussi que plusieurs milliers de neurones dans le réservoir sont nécessaires pour la prédiction, augmentant à la fois le temps de calcul des états et la régression.

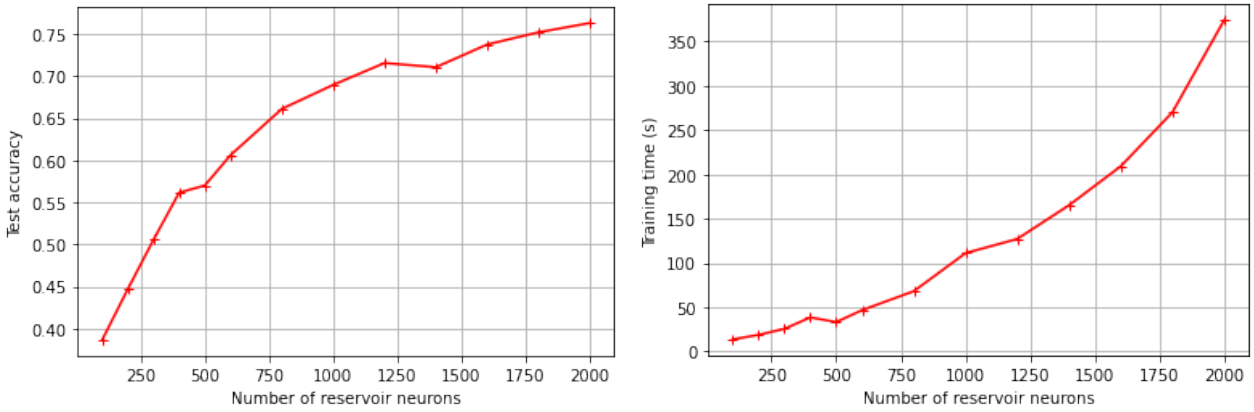


**Figure 2** – Compromis temps de calcul / Taille du réservoir

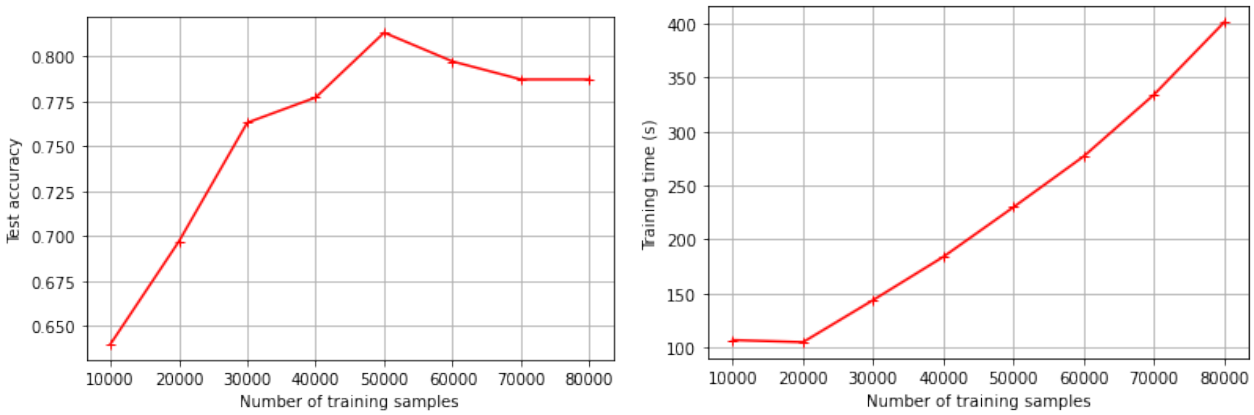


**Figure 3** – Compromis temps de calcul / Taille des échantillons de données

Les entraînements que nous avons effectués montrent que la performance du modèle augmente avec la taille du réservoir, comme attendu. Cependant, le temps d'entraînement augmente aussi exponentiellement avec la taille du réservoir. Pour ce qui est de la taille du jeu d'entraînement, nous obtenons la meilleure performance avec 50 000 données.

## 4.2 Optimisation des paramètres

### 4.2.1 GridSearch

Comme son nom l'indique, le problème d'optimisation considéré se rapporte à un problème de recherche dans une grille.

En fait, pour chaque hyperparamètre, il suffit de définir l'ensemble de valeurs qu'il peut prendre. Nous entraînons ensuite le modèle pour chaque combinaison d'hyperparamètres et mémorisons les résultats de performance. Dans ce cas, il suffit d'obtenir les hyperparamètres les plus performants. Cette méthode s'avère efficace lorsque le nombre de paramètres à optimiser est faible. Dans ce travail, nous avons implémenté plusieurs méthodes GridSearch en parallèle afin de fixer au fur et à mesure des valeurs typiques d'hyperparamètres et aboutir à un réglage fin de l'ESN. Le tableau suivant résume nos différentes phases d'entraînement :

| Paramètres | Tests | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| n_reservoir | 400 | 400 | 400 | 800 | 1500 | 2 000 | 2 000 | 2800 |
| alpha | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | .001 |
| spectral_radius | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| sparsity | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| noise | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| input_scaling | 0.5 | 0.3 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| input_shift | 0.01 | 0.01 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 |
| teacher_scaling | None | None | 0.01 | 0.01 | 0.01 | 0.001 | 0.001 | 0.001 |
| teacher_shift | 0.2 | 0.2 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.2 |
| teacher_forcing | False | False | False | False | False | False | True | True |
| feedback_scaling | 0.6 | 0.6 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.01 |
| random_state | None | None | None | None | None | None | None | Noneoutil |
| out_activation | id | id | id | id | id | identity | id | id |
| inverse_out_activation | id | id | id | id | id | identity | id | id |
| train data | 200 000 | 200 000 | 300000 | 300000 | 80000 | 80 000 | 80 000 | 60 000 |
| test data | 40 000 | 40 000 | 60000 / 2000 | 60000 / 2000 | 60000 / 2000 | 40 000 | 40 000 | 30 000 |
| continuation | True | True | True | True | True | True | True | True |
| **test accuracy** | **0.52** | **0.568** | **0.585** | **0.678** | **0.7335** | **0.778** | **0.748** | **0.810** |

**Figure 4** – GridSearch parallèles

Ces tests nous permettent de fixer les paramètres suivants :

**Tableau 3** – Paramètres fixés du réseau

| Parameter | Définition | valeur |
|---|---|---|
| $N_{res}$ | Nombre de neurones dans le réservoir | $> 2500$ |
| Spectral radius | Rayon spectral de $W_{rec}$ | 1 |
| Sparsity | Pourcentage de vide dans $W_{rec}$ | 0,5 |
| input shift | Terme additif des entrées | ? |
| input scaling | Terme multiplicatif des entrées | ? |
| teacher shift | Terme additif des états récurrents | ? |
| teacher scaling | Terme multiplicatif des états récurrents | ? |
| noise | Bruit additif gaussien | 0,001 |
| leaking rate ($\lambda$) | équation (1) | 0,001 |

Avec cette méthode de recherche par grille, nous n'avons pas pu déterminer les valeurs optimales de shifting et scaling des données. Ces dernières sont toutes situées dans $[0; 1]$ et il semble alors que le GridSearch ne soit pas adapté.

### 4.2.2 RandomSearch

Après recherches des différentes méthodes d'optimisation, nous optons pour la méthode RandomSearch, aussi implémentée dans la bibliothèque **sklearn**, qui a pour différence de considérer des distributions de valeurs au lieu de valeurs fixées. On considère alors les paramètres de shifting et scaling comme des **distributions uniformes sur** $[0; 1]$ et tirons un nombre $N_{iter} = 10$ de réalisations de ces paramètres, pour différentes valeurs de $N_{res}$. Avec des distributions uniformes, nous n'aboutissons jamais à des résultats concluants. En changeant les distributions par des normales, on obtient les résultats suivants :

| Parameter | Valeur RandomSearch | $N_{res}$ |
|---|---|---|
| input shift | 0.05 | 3000 |
| input scaling | 0.1 | 3000 |
| teacher shift | 0.1 | 3000 |
| teacher scaling | 0.001 | 3000 |
| feedback scaling | 0.2 | 3000 |

Ainsi, on obtient sur 60 000 données d'entraînement et un nombre de neurones $N_{res} = 3000$ un score de précision de **83,1%**, améliorable en augmentant le nombre de neurones dans le réservoir. Cependant, la pseudo inversion de $W_{out}$ devient trop coûteuse, pour un faible gain de précision. Il devient primordial d'étudier les méthodes d'optimisation du calcul de cette matrice, une fois tout les hyperparamètres du réseau fixés.

# 5 Optimisation du calcul de $W_{out}$

Le calcul actuel consiste en la pseudo-inversion d'une matrice contenant toutes les données d'apprentissage, d'où la complexité calculatoire élevée, en $O(n^3)$. Nous allons donc explorer deux méthodes de simplification de ce calcul, l'une étant une technique classique de minimisation en machine learning (rétropropagation stochastique du gradient), et l'autre une version récursive de minimisation de moindres carrées (RLS). Finalement, nous étudierons leurs vitesses relatives par rapport à une régression de Ridge dans la dernière partie.

## 5.1 Méthode classique des moindres carrées pour le calcul de $W_{out}$

Nous implémentons dans la méthode **fit** de notre classe **ESN** l'algorithme de descente de gradient stochastique, dont le fonctionnement, adapté au framework du Reservoir Computing, est résumé ci-dessous :
Notons $Y(n) = [y_1(n), y_2(n), ..., y_c(n)]^T$ les outputs et $X(n) = [x_1(n), x_2(n), ..., x_d(n)]^T$ les états du réservoir. Alors le problème linéaire précédent peut être écrit sous la forme

$$Y(n) = W^0 X(n) \tag{3}$$

Ainsi, pour la n-ième donnée d'entraînement, on cherche à minimiser la somme suivante :

$$\epsilon(n) = \frac{1}{2}\sum_{i=1}^{c}(d_i(n) - y_i(n))^2 + \frac{\lambda}{2}\sum_{i=1}^{c}\sum_{j=1}^{d}(W_{i,j}^0)^2 \tag{4}$$

où les $d_i(n)$ sont les vrais labels attribués aux $x_i(n)$ et $\lambda$ un paramètre de régularisation de $||.||_2$.
On peut aussi écrire (4) sous la forme :

$$\epsilon(n) = \frac{1}{2}||d(n) - y(n)||^2 + \frac{\lambda}{2}||W^0||^2 \tag{5}$$

$$\epsilon(n) = \frac{1}{2}[d(n) - W^0 x(n)][d(n) - W^0 x(n)]^T + \frac{\lambda}{2}Tr(W^0(W^0)^T) \tag{6}$$

Finalement, en utilisant les résultats d'algèbre linéaire sur la dérivation de matrices, on obtient :

$$\frac{\partial \epsilon}{\partial W^0}(n) = -[d(n) - W^0 x(n)]x(n)^T + \lambda W^0 \tag{7}$$

Soit l'algorithme de descente stochastique suivant, en notant $\alpha$ le learning rate :

$$W^0(n+1) = W^0(n) - \alpha \frac{\partial \epsilon}{\partial W^0}(n) \Leftrightarrow W^0(n+1) = W^0(n)(1-\alpha\lambda) + \alpha[d(n) - W^0(n)x(n)]x(n)^T \tag{8}$$

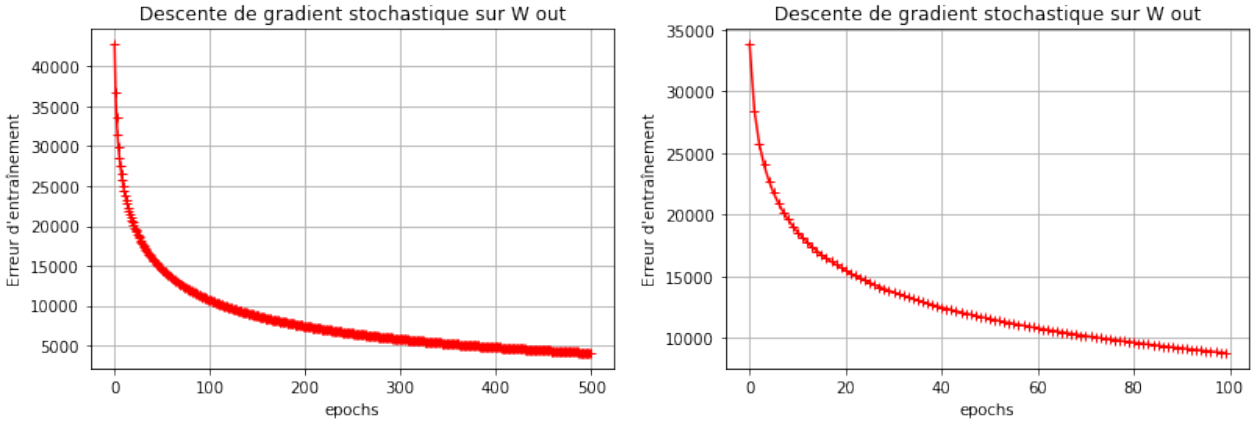


**Figure 5** – $\lambda = 10^{-8}$, $\alpha = 10^{-5}$ à gauche et $\lambda = 10^{-5}$, $\alpha = 10^{-3}$ à droite

On observe bien la convergence du réseau vers une matrice de poids optimale, avec une précision de **84%** sur 60 000 données tests. Alors que la méthode *pseudo-inverse* prenait **11 min** de calculs à nos machines, cette méthode permet d'accélerer d'un **facteur 2** le calcul de la matrice de sortie.

Finalement, elle semble bien adaptée au problème posé. Intéressons-nous alors à sa version récursive.

## 5.2 Méthode récursive des moindres carrées pour le calcul de $W_{out}$

Nous sommes toujours dans le cadre de l'équation (3). Alors, en posant $e(i)$ comme suivant :

$$e(i) = y(i) - W_{out}x(i) \tag{9}$$

On cherche alors à minimiser la fonction de coût suivante :

$$E(n) = \sum_{i=1}^{n} \lambda^{n-i} e(i)^T e(i) + \delta\lambda^n ||W_{out}||_F \tag{10}$$

où $||.||_F$ est défini dans l'équation (6), $\lambda$ est le forgetting factor ($\lambda = 1$ no forgetting) et $\delta$ un paramètre de régularisation. Il vient alors :

$$E(n) = \sum_{i=1}^{n} \lambda^{n-i} Tr([y(i) - W_{out}x(i)][y(i) - W_{out}x(i)]^T) + \delta\lambda^n Tr(W_{out}(W_{out})^T) \tag{11}$$

$$\frac{\partial E}{\partial W_{out}}(n) = \sum_{i=1}^{n} \lambda^{n-i}[-2y(i)x(i)^T + 2W_{out}x(i)x(i)^T] + 2\delta\lambda^n W_{out} \tag{12}$$

Posons :

$$\begin{cases} \Psi(n) = \sum_{i=1}^{n} \lambda^{n-i} x(i)x(i)^T + \delta\lambda^n I_n \\ \Gamma(n) = \sum_{i=1}^{n} \lambda^{n-i} y(i)x(i)^T \end{cases}$$

L'équation (12) s'annule lorsque :

$$W_{out} = \Gamma(n)\Psi(n)^{-1} \tag{13}$$

Le but de cet algorithme de calcul de $W_{out}$ est donc de calculer récursivement $\Gamma$ et $\Psi$. En examinant $\Gamma$ et $\Psi$, on a les relations de récurrence suivantes :

$$\begin{cases} \Psi(n) = \lambda\Psi(n-1) + x(n)x(n)^T \\ \Gamma(n) = \lambda\Gamma(n-1) + y(n)x(n)^T \end{cases}$$

On cherche une forme pratique de calcul de $\Psi^{-1}$. Un lemme d'inversion de matrices donne :

$$\Psi(n)^{-1} = \lambda^{-1}\Psi(n-1)^{-1} - \frac{\lambda^{-2}\Psi(n-1)^{-1}x(n)x(n)^T\Psi(n-1)^{-1}}{1+\lambda^{-1}x(n)^T\Psi(n-1)^{-1}x(n)} \tag{14}$$

On pose alors : $\begin{cases} P(n) = \Psi(n)^{-1} (\Leftrightarrow P^T = P) \\ \kappa(n) = \frac{\lambda^{-1}P(n-1)x(n)}{1+\lambda^{-1}x(n)^T P(n-1)x(n)} \end{cases}$

Ainsi,

$$P(n) = \lambda^{-1}P(n-1) - \lambda^{-1}\kappa(n)x(n)^T P(n-1) \tag{15}$$

Alors, l'équation (15) implique :

$$\kappa(n) = P(n)x(n) \tag{16}$$

Nous pouvons désormais implémenter le calcul récursif de $W_{out}$, grâce à l'équation suivante :

$$\begin{cases} W_{out}(n) = W_{out}(n-1) - \Omega(n)\kappa(n)^T \\ \Omega(n) = y(n) - W_{out}(n-1)x(n) \end{cases}$$

où $\Omega$ est l'erreur de prédiction incrémentale. Finalement, résumons l'algorithme RLS :

**Algorithme 2 :** Recursive Least Square

*Initialisation*

- $W_{out} = O_{n,n}$
- $P(0) = \delta^{-1}I_n$

*Calculs*

**for** $n \leftarrow 1$ **to** $n \leftarrow N_{max}$ **do**

- $\kappa(n) = \frac{P(n-1)x(n)}{\lambda+x(n)^T P(n-1)x(n)}$
- $\Omega(n) = y(n) - W_{out}(n-1)x(n)$
- $W_{out}(n) = W_{out}(n-1) - \Omega(n)\kappa(n)^T$
- $P(n) = \lambda^{-1}P(n-1) - \lambda^{-1}\kappa(n)x(n)^T P(n-1)$

**end**

*Sorties*

- $W_{out}(N_{max})$

## 5.3 Comparaison des méthodes en matière de temps de calcul

Afin de comparer l'efficacité des méthodes proposées, on entraîne le même modèle avec chaque méthode. On utilise les paramètres suivants :

**Tableau 4** – Paramètres du réseau

| Parameter | Valeur |
|---|---|
| $N_{res}$ | 500 |
| Alpha | 0.001 |
| Spectral radius | 1 |
| Sparsity | 0.5 |
| Noise | 0.001 |
| Input shift | 0.2 |
| Input scaling | 0.1 |
| Teacher shift | 0.2 |
| Teacher scaling | 0.001 |
| Teacher forcing | True |
| Feedback scaling | 0.01 |
| Learning rate | 0.0001 |
| Regularization parameter | 0.01 |
| Taille de batch | 32 |
| Forget rate | 0.7 |

On obtient les temps de calcul suivants :

**Tableau 5** – Temps d'entraînement selon la méthode de calcul de $W_{out}$

| Méthode | Nombre d'époques | Temps d'entraînement (s) |
|---|---|---|
| Méthode classique | - | 44.8 |
| Descente de gradient | 10 | 41.4 |
| | 50 | 50.3 |
| | 100 | 61.5 |
| Recursive Least Squares | 10 | 207.3 |
| | 50 | 875.9 |
| | 100 | |

La méthode par la descente de gradient est à peu près équivalente à la méthode classique en matière de temps d'entraînement, sous réserve de limiter le nombre d'époques à environ 100. La méthode récursive, elle, est bien plus longue que la méthode classique.

On observe que pour la descente de gradient le temps d'entraînement n'augmente pas proportionnellement au nombre d'époques : lorsqu'on passe de 10 à 50 époques, on passe de 41 à 50 secondes. Ainsi, on peut en déduire que l'étape la plus longue de l'entraînement de l'ESN n'est pas le calcul de $W_{out}$, mais le calcul des états du réseau.

Nous affinons donc notre analyse pour étudier la différence entre le temps de calcul des états du réseau et le temps de calcul de $W_{out}$. En utilisant la méthode classique de calcul de $W_{out}$ et les mêmes paramètres que dans le tableau 5, et en faisant varier le nombre de neurones dans le réseau, on obtient les résultats suivants.

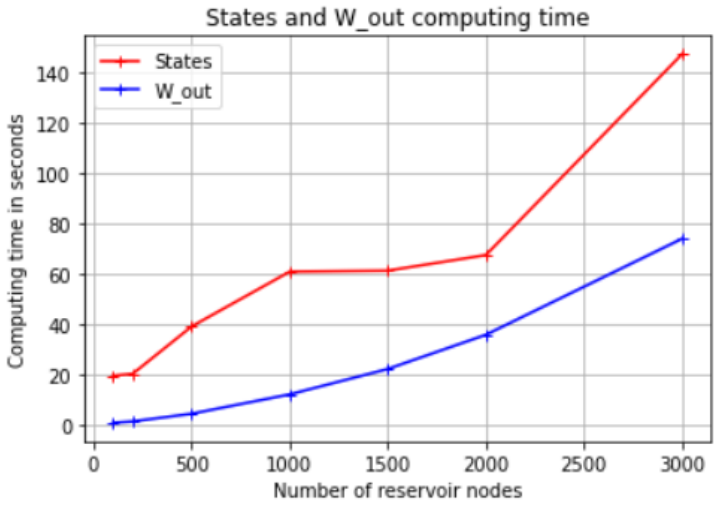


**Figure 6** – Comparaison du temps de calcul des états du réseau et de $W_{out}$ en fonction du nombre de neurones.

Ainsi, nous en déduisons qu'à partir d'un nombre élevé de neurones dans le réservoir, le temps d'entraînement dédié au calcul des états du réseau est deux fois supérieur à celui de la matrice $W_{out}$.

# 6 Conclusion

Pour démontrer l'efficacité des RCN dans le traitement des séries temporelles sur la base de données MNIST, nous avons implémenté un Echo State Network (ESN) et cherché à affiner le choix des paramètres du réseau. Avec les paramètres trouvés, nous avons obtenu une précision sur l'ensemble de test de 83%. Cela est bien inférieur au score de 98% obtenu avec un CNN, mais nous nous approchons tout de même des précisions obtenues dans la littérature. Afin d'améliorer ces résultats, il nous aurait fallu augmenter la taille du réservoir, que nous avons limitée à 2000 neurones durant notre étude en raison des ressources de calcul à notre disposition.

Nous avons ensuite étudié deux manières d'optimiser le calcul de la matrice de poids $W_{out}$ : la descente de gradient et la méthode récursive des moindres carrés. Avec la descente de gradient, on obtient une précision de 84% sur l'ensemble de test, et sous réserve que le

nombre d'époques ne soit pas trop élevé le temps d'entraînement est équivalent à celui de la méthode classique. Ainsi, on en conclut que cette méthode n'apporte pas de réelle amélioration du modèle. Avec la méthode récursive des moindres carrés, on observe que le temps d'entraînement est largement supérieur à celui de la méthode classique (multiplié par 5 pour un entraînement sur 10 époques). Cette méthode n'a donc pas l'effet voulu d'optimisation du temps de calcul.

Finalement, notre analyse de la répartition du temps de calcul entre les états du réseau et la matrice $W_{out}$ nous apprend que le calcul des états du réseau est plus de deux fois supérieur à celui de $W_{out}$. Ainsi, de prochains travaux pourraient être consacrés à l'optimisation du calcul des états du réseau.