Cours : Réseau de neurones – Application aux NARX

L'ensemble de ce cours prend appui sur l'excellent livre *Neural Network Design* de Hagan *et al.* [1].

1 Généralités

1.1 Les problèmes types de l'IA

Les thèmes d'utilisation de l'IA (ou plutôt du *machine learning*) sont :

- la *classification* (idem en anglais) ;
- la *régression* (idem en anglais) ;
- la *segmentation* ou *clustering* ;
- la *prédiction* (idem en anglais).

Pour la classification (fig. 1a), il s'agit d'identifier à quelle catégorie un objet appartient (apprentissage supervisé). Les applications sont la détection de spam, la reconnaissance d'images, etc. Les algorithmes classiques sont : SVM (*Support Vector Machine*), les plus proches voisins (*Nearest Neighbours*) et la forêt d'arbre de décision (*random forest*).
En ce qui concerne la *régressions* (fig. 1b), le but est de prédire un paramètre à valeur continue associé à un objet. Les applications sont : les réponses aux médicaments, le cours de la bourse, etc. Les algorithmes les plus utilisés sont : SVR (*Support Vector Regression*), les plus proches voisins (*Nearest Neighbours*) et la forêt d'arbre de décision (*random forest*).
Pour le *clustering* (fig. 1c), il s'agit de regrouper de manière automatique des objets dans des ensembles (apprentissage non supervisé). Les applications sont : segmentation marketing, regroupement des résultats des expériences. Les algorithmes utilisés sont : k-moyennes (*k-means*), le partitionnement spectral (*spectral clustering*), le « décalage moyen » (*mean shift*).
Enfin, on retrouve la catégorie de la *prédiction* (fig. 1d) pour les données temporelles. Les applications sont : la reconnaissance vocale, la météo ou encore la correction d'un système.
Le site `https://sklearn.org/` propose un organigramme de choix d'un algorithme en fonction de l'objectif, des données et des essais/erreurs (fig. 2). Alors ou placer les fameux réseaux de neurones ? Et bien ils peuvent être utiliser dans tous les types de problèmes ! Nous allons faire quelques rappels dans la sous-section suivante.

1.2 Les réseaux de neurones

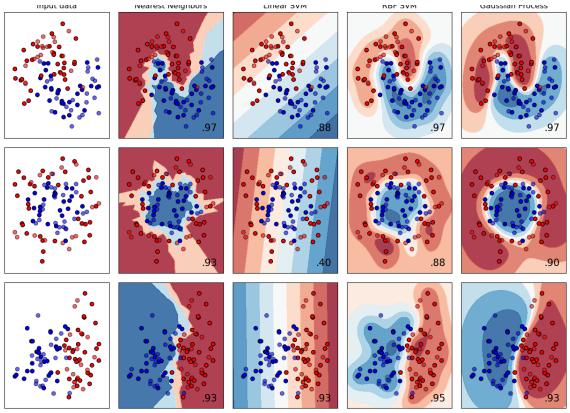
Cette présentation utilise les notations utilisées par Hagan *et al.* [2].

1.2.1 Le modèle du neurone

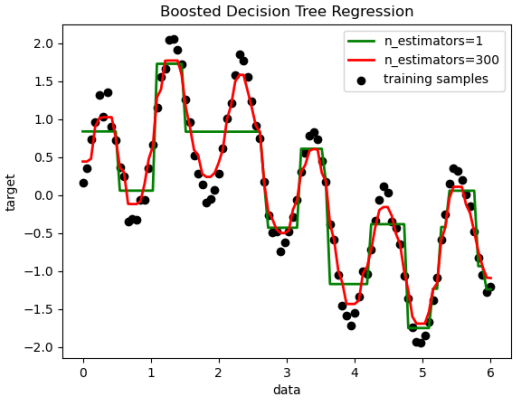
Le réseau neuronal perceptron multicouche est constitué de composants simples. Nous commencerons par un neurone à entrée unique, que nous étendrons ensuite à des entrées multiples, puis nous empilerons ces neurones pour produire des couches. Enfin, nous mettrons les couches en cascade pour former le réseau.
Un neurone à entrée unique est illustré à la figure 3a. L'entrée scalaire p est multipliée par le poids scalaire w pour former wp, un des termes qui est envoyé au sommateur. L'autre entrée, 1, est multipliée par un biais b et est ensuite envoyée au sommateur. La sortie du sommateur entre dans une fonction d'activation f_a, qui produit la sortie scalaire des neurones a.
La sortie du neurone est calculée comme suit :

$$a = f_a\left(wp + b\right) \tag{1}$$

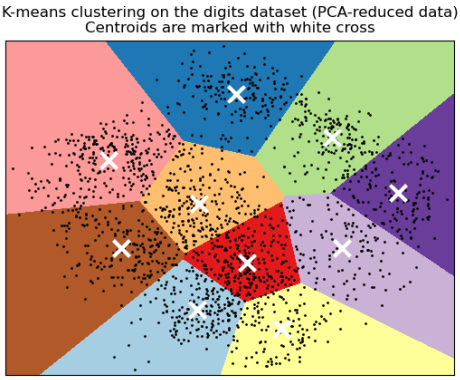
Notez que w et b sont tous deux des paramètres scalaires réglables du neurone. Généralement, la fonction d'activation est choisie par le concepteur, puis les paramètres w et b sont ajustés par une règle d'apprentissage de sorte que la relation entrée/sortie du neurone réponde à un objectif spécifique. La fonction d'activation de la figure 3a peut être une fonction linéaire ou non linéaire. La fonction historique est la fonction sigmoïde, qui est illustrée à la figure 3b.



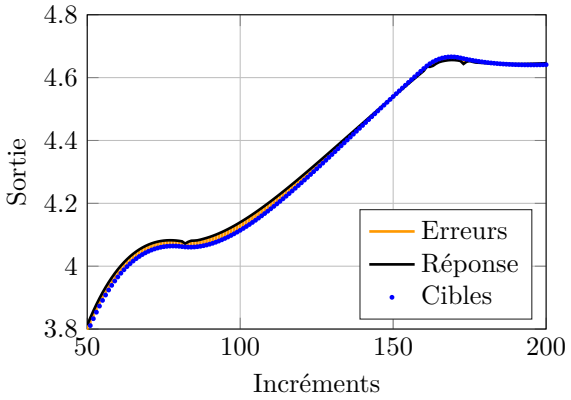
(a) Classification



(b) Régression



(c) Clustering



(d) Prédiction

FIGURE 1 – Illustrations des différentes types de problématique

Cette fonction d'activation prend l'entrée (qui peut avoir n'importe quelle valeur entre plus et moins l'infini) et impose la sortie dans la plage $[0; 1]$, selon l'expression :

$$a = \frac{1}{1 + \mathrm{e}^{-(wp+b)}} \tag{2}$$

En général, un neurone a plus d'une entrée. Un neurone avec R entrées est illustré à la figure 4. Les entrées individuelles $p_1, p_2, \ldots, p_R$ sont chacune pondérées par les éléments correspondants $w_{1,1}, w_{1,2}, \ldots, w_{1,R}$ de la matrice des poids $\mathbf{W}$.
Le neurone a un biais b qui est additionné aux entrées pondérées pour former la sortie :

$$n = \sum_{i=1}^{R} w_{1,i} p_i + b \tag{3}$$

On peut réécrire cette équation sous forme matricielle :

$$n = \mathbf{W} \cdot \underline{p} + b \tag{4}$$

où la matrice $\mathbf{W}$ pour le neurone simple n'a qu'une seule ligne.
Maintenant la sortie du neurone peut s'écrire sous la forme :

$$a = f_a \left(\mathbf{W} \cdot \underline{p} + b \right) \tag{5}$$

La figure 5 représente le neurone sous sa forme matricielle :

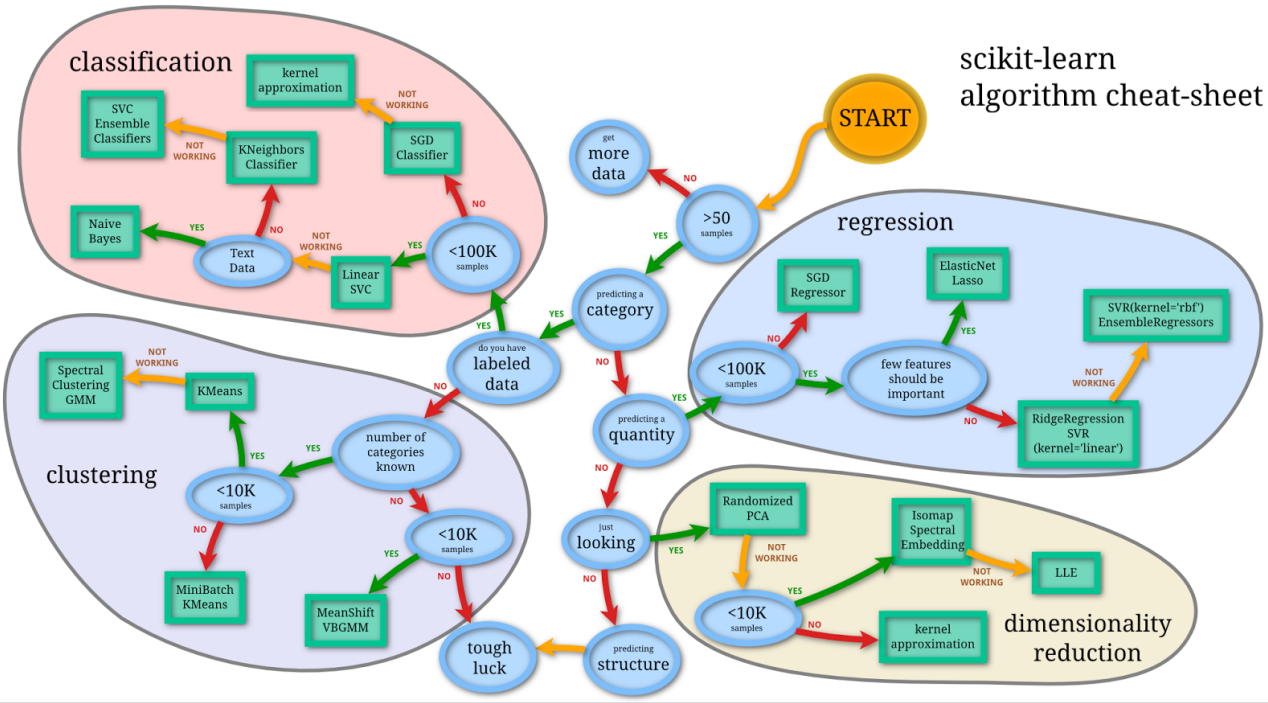
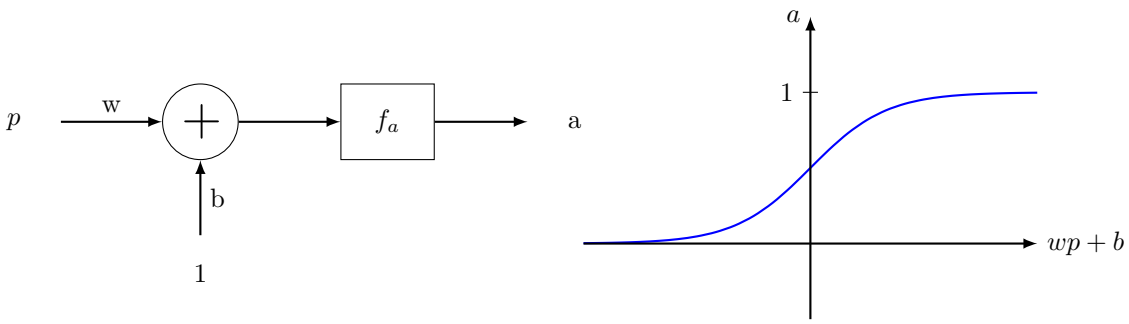


FIGURE 2 – Organigramme de choix d'un algorithme



(a) Un neurone à entrée simple

(b) La fonction d'activation sigmoïde

FIGURE 3 – Illustration d'un neurone simple

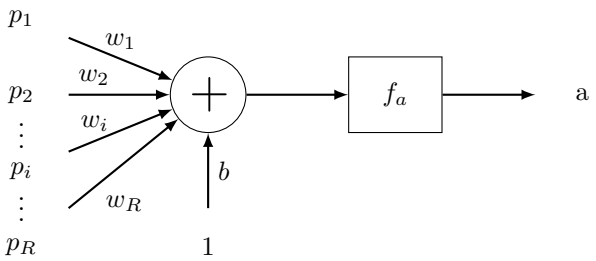


FIGURE 4 – Neurone à entrées multiples

1.2.2 Les architectures des réseaux

Généralement, un seul neurone, même avec de nombreuses entrées, n'est pas suffisant. Il peut en falloir cinq ou dix, fonctionnant en parallèle, dans ce que l'on appelle une couche. Un réseau de S neurones à une seule couche est illustré à la figure 6a. Notez que chacune des R entrées est connectée à chacun des neurones et que la

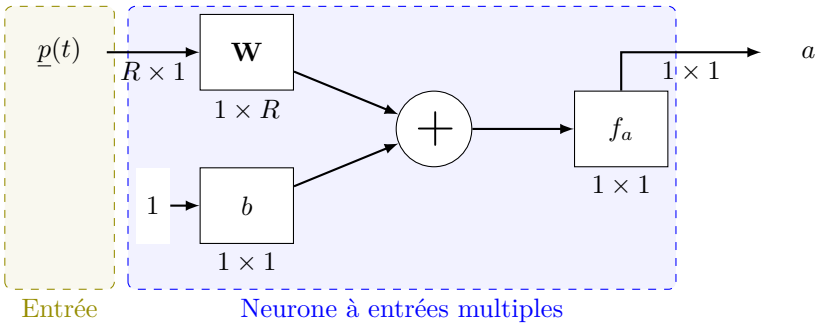
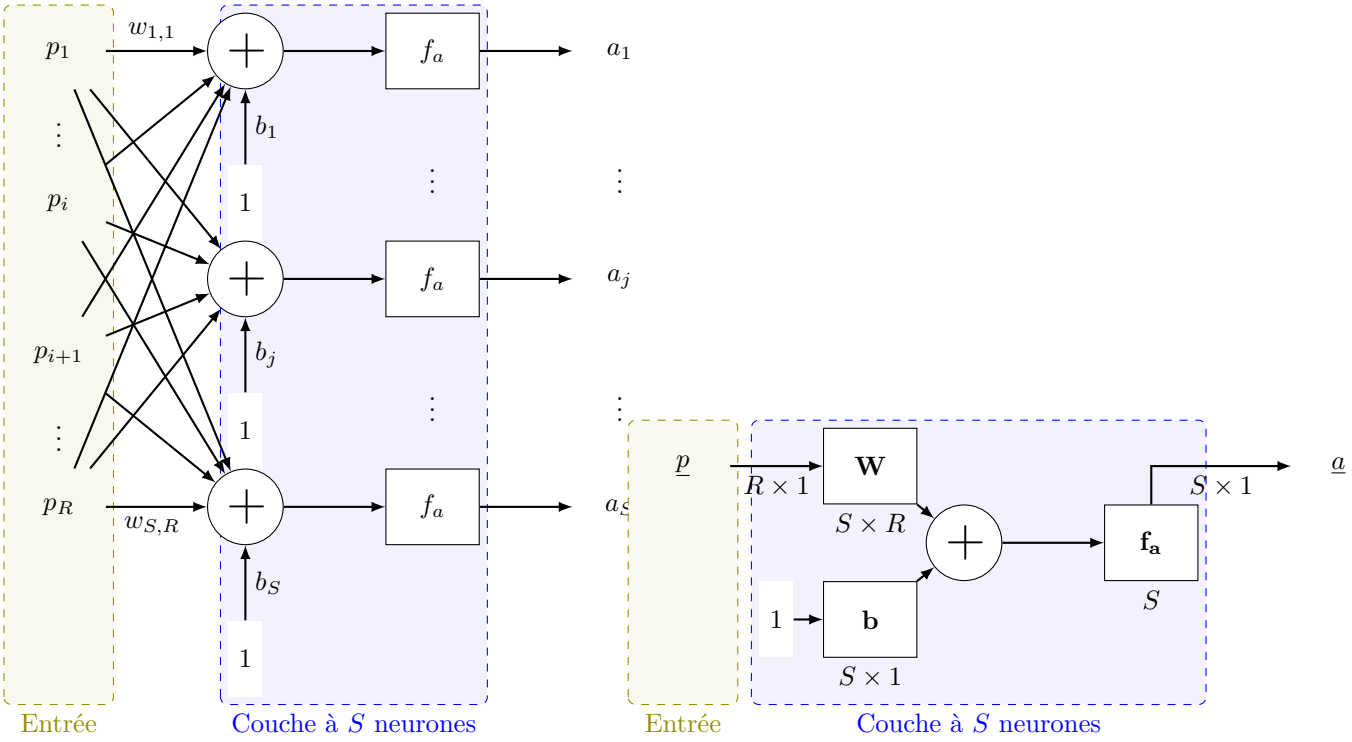


FIGURE 5 – Neurone à R entrées – représentation matricielle

matrice de poids comporte maintenant S lignes. La couche comprend la matrice de poids $\mathbf{W}$, les sommateurs, les biais, les fonctions d'activation et le vecteur de sortie. Il est usuel que le nombre d'entrées dans une couche soit différent du nombre de neurones (c'est-à-dire $R \neq S$).
Le réseau à S neurones, R entrées, une couche peut aussi se représenter sous forme matricielle comme illustré sur la figure 6b.



(a) Couche à S neurones

(b) Couche à S neurones – Représentation matricielle

FIGURE 6 – Couche représentée de deux manière différentes

Considérons maintenant un réseau à plusieurs couches. Chaque couche a sa propre matrice de poids, son propre vecteur de biais, un vecteur d'entrée et un vecteur de sortie. Nous devons introduire une notation supplémentaire pour distinguer ces couches. Nous utiliserons des exposants pour identifier les couches.

2 Étapes préalables à l'entrainement

Un certain nombre d'étapes doivent être réalisées avant l'entrainement du réseau. Elles peuvent être regroupés en trois catégories :

— sélection des donnée ;

— prétraitement des données ;
— choix du type de réseau (imposé ici par le type de problème) et de son architecture.

2.1 Sélection des données

Il est généralement difficile d'intégrer des connaissances préalables dans un réseau de neurones, par conséquent, la qualité du réseau sera fonction de la qualité des données utilisées pour l'entraîner. Les réseaux de neurones représentent une technologie qui est à la merci des données. Les données relatives à l'entrainement doivent couvrir l'ensemble de l'espace des entrées d'utilisation du réseau. Il n'est pas possible de garantir la performance du réseau lorsque les entrées au réseau sont en dehors du champ d'application de l'entrainement. Les réseaux de neurones, comme les autres méthodes non linéaires « black box », ne permettent pas de faire de bonnes extrapolations.
Comment pouvons-nous être sûrs que l'espace d'entrée a été correctement échantillonné par les données d'entrainement ? Il est difficile de le faire avant l'entrainement, et il y a de nombreux cas dans lesquels nous n'avons aucun contrôle sur le processus de collecte des données et on doit alors utiliser toutes les données disponibles. En analysant le réseau entrainé, nous pouvons souvent dire si les données d'entrées étaient suffisantes.
Une dernière question que nous devons nous poser sur la sélection des données est « avons-nous suffisamment de données ? » Il est difficile de répondre à cette question, surtout avant d'entrainer le réseau. La quantité de données requises dépend de la complexité de la fonction que nous essayons d'approximer. C'est pourquoi l'ensemble du processus d'entrainement du réseau neuronal est itératif. À l'issue de l'entrainement, nous analyserons la performance du réseau. Les résultats de cette analyse peuvent nous aider à décider si nous avons suffisamment de données ou non.

2.2 Prétraitement des données

L'objectif principal de la phase de prétraitement des données est de faciliter l'entrainement au réseau. Le prétraitement des données comprend des étapes telles que la normalisation, les transformations non linéaires, l'extraction de caractéristiques, le codage d'entrées/cibles discrètes, le traitement des données manquantes, etc. L'idée est d'effectuer des traitements des données pour faciliter l'entrainement des réseaux de neurones.
En ce qui concerne la normalisation, la méthode la plus répandue est de ramener les données d'entrées dans l'intervalle $[-1; 1]$. Cela facile l'entrainement puisque bien souvent le réseau comporte des fonctions sigmoïdes ou encore tangente hyperbolique dont l'amplitude de sortie est bornée.

2.3 Choix du réseau

L'étape suivante du processus d'entrainement du réseau est le choix de l'architecture du réseau. Le type de base de l'architecture de réseau est déterminé par le type de problème que nous souhaitons résoudre. Une fois que l'architecture de base est choisie, nous devons décider de détails spécifiques tels que le nombre de neurones et de couches que l'on veut utiliser, combien de sorties le réseau devrait avoir et quel type de fonction de performance nous voulons utiliser pour l'entrainement.
Les prévisions relèvent également des catégories de l'analyse des séries chronologiques, de l'identification des systèmes, du filtrage ou de la modélisation dynamique. L'idée est que nous souhaitons prédire la valeur future de certaines séries chronologiques. Un négociant en actions peut vouloir prédire la valeur future d'un titre. Un ingénieur en contrôle peut vouloir prédire la valeur future de la concentration d'un produit chimique, qui est la production d'une usine de traitement. Un ingénieur en systèmes d'énergie peut vouloir prédire les pannes du réseau électrique.
La prédiction nécessite l'utilisation de réseaux neuronaux dynamiques. La forme spécifique du réseau dépendra de l'application. Le réseau le plus simple pour la prédiction non linéaire est le « focused time-delay neural network », qui est illustré à la figure 7. Il fait partie d'une classe générale de réseaux dynamiques, appelés « focused networks », dans lesquels la dynamique n'apparaît qu'à la couche d'entrée d'un réseau statique multicouche à réaction en chaîne. Ce réseau présente l'avantage de pouvoir être entrainé à l'aide de réseaux statiques et des algorithmes de rétropropagation, puisque la ligne « tapped-delay-line » (TDL) à l'entrée du réseau peut être remplacée par un vecteur étendu de valeurs retardées de l'entrée.

Les réseaux à deux couches, avec des fonctions d'activation sigmoïde dans la couche cachée et des fonctions linéaire dans la couche de sortie, sont des équivalents universels comme l'a démontré Hornik *et al.* [3].

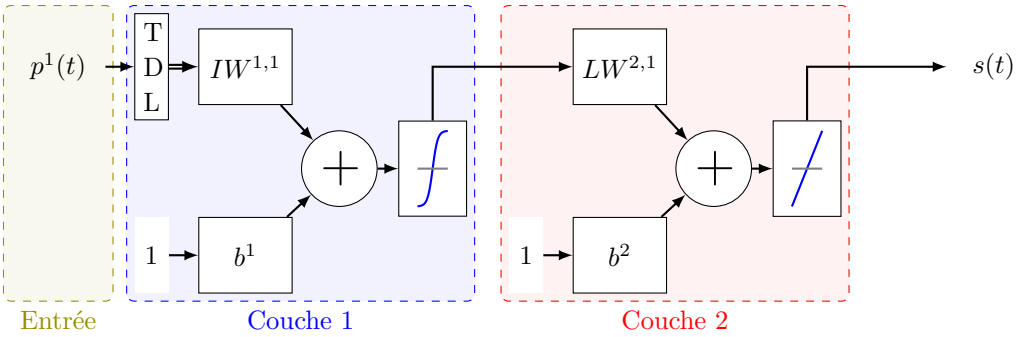


FIGURE 7 – Illustration d'un « Focused Time-Delay Neural Network »

Pour les problèmes de modélisation et de contrôle dynamiques, le réseau NARX (Nonlinear AutoRegressive model with eXogenous input) est très utilisé. Ce réseau est illustré à la figure 8. Le signal d'entrée pourrait représenter, par exemple, la tension appliquée à un moteur, et la sortie pourrait représenter la position angulaire d'un bras de robot. Comme pour le focused time-delay neural network, le réseau NARX peut être entraîné avec une rétropropagation statique. Les deux lignes de retard peuvent être remplacées par des vecteurs étendus d'entrées et des objectifs retardées. Nous pouvons utiliser les objectifs, au lieu de renvoyer les sorties du réseau (ce qui nécessiterait une rétropropagation dynamique pour l'entrainement), car les sorties du réseau devraient correspondre aux objectifs lorsque l'entrainement est terminé.

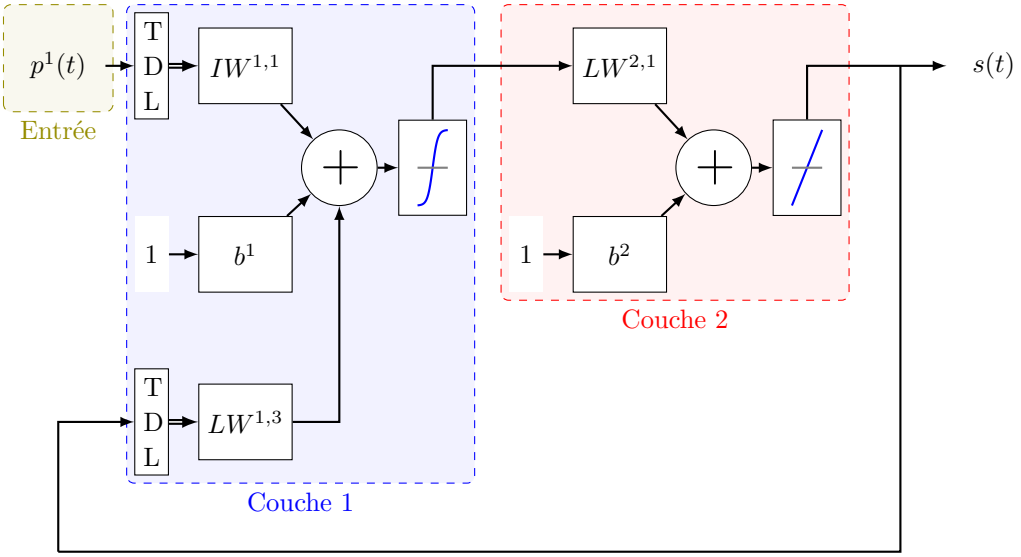


FIGURE 8 – Illustration d'un réseau NARX

Il existe de nombreux autres types de réseaux dynamiques qui pourraient être utilisés pour la prévision, mais le « Focused Time-Delay Neural Network » et le réseau NARX sont les plus simples de leur genre.

3 Entrainement

Après la préparation des données et la mise en place de l'architecture du réseau sélectionnés, nous sommes prêts à entrainer le réseau. Dans cette section, nous aborderons certaines des décisions qui doivent être prises dans le cadre du processus d'entrainement. Cela inclut la méthode d'initialisation des poids, l'algorithme d'entraînement, l'indice de performance et le critère d'arrêt de l'entraînement.

3.1 Initialisation des poids

Avant d'entrainer le réseau, nous devons initialiser les poids et les biais. La méthode que nous utiliserons dépendra du type de réseau. Pour les réseaux multicouches, les poids et les biais sont généralement fixés à de petites valeurs aléatoires (par exemple, répartis uniformément entre -0,5 et 0,5, si les entrées sont normalisées pour se situer entre -1 et 1).
Il existe une autre approche pour fixer les pondérations et les biais initiaux d'un réseau à deux couches. Il a été introduit par Nguyen et Widrow [4]. L'idée est de fixer l'amplitude des poids dans la première couche de manière à ce que la région linéaire de chaque fonction sigmoïde couvre $\frac{1}{S^1}$ de l'amplitude de l'entrée. Les biais sont ensuite fixés de façon aléatoire, de sorte que le centre de chaque fonction sigmoïde tombe de façon aléatoire dans l'espace d'entrée. Les avantages par rapport aux poids et biais purement aléatoires sont les suivants :

— peu de neurones sont gaspillés (car tous les neurones se trouvent dans l'espace d'entrée) ;

— l'entraînement est plus rapide (parce que chaque zone de l'espace d'entrée a des neurones).

3.2 Choix de l'algorithme d'entrainement

3.2.1 Levenberg-Marquardt

Pour les réseaux multicouches ayant jusqu'à quelques centaines de poids et de biais qui sont utilisés pour l'approximation des fonctions, l'algorithme de Levenberg-Marquardt est généralement l'algorithme d'entrainement le plus rapide. Lorsque le nombre de poids atteint un millier ou plus, le Levenberg-Marquardt n'est pas aussi efficace que certains algorithmes à gradient conjugué. Cela est principalement dû au fait que le calcul inverse de la matrice s'échelonne avec le nombre de poids.
La principale application de l'algorithme de Levenberg-Marquardt est le problème d'approximation de fonctions par les moindres carrés : étant donné un ensemble de m paires empiriques (xi, yi) de variables indépendantes et dépendantes, trouver les paramètres β de la fonction modèle $f(x, \beta)$ de sorte que la somme des carrés des déviations $S(\beta)$ est minimisée :

$$\hat{\boldsymbol{\beta}} \in \operatorname{argmin}_{\boldsymbol{\beta}} S\left(\boldsymbol{\beta}\right) \equiv \operatorname{argmin}_{\boldsymbol{\beta}} \sum_{i=1}^{m} \left[y_i - f\left(x_i, \boldsymbol{\beta}\right)\right]^2 \tag{6}$$

3.2.2 Inférence bayésienne

L'inférence bayésienne est une méthode d'inférence par laquelle on calcule les probabilités de diverses causes hypothétiques à partir de l'observation d'événements connus. Elle s'appuie principalement sur le théorème de Bayes :

$$p(A|B) = \frac{p(B|A) \cdot p(A)}{p(B)} \tag{7}$$

où :

— $p(A|B)$ désigne la probabilité de survenue de l'événement A sachant que l'événement B est survenu ;

— $p(A)$ désigne la probabilité de survenue de l'événement A.

Le raisonnement bayésien construit, à partir d'observations, une probabilité de la cause d'un type d'événements. On attribue à toute proposition de cause une valeur la probabilité dans l'intervalle ouvert allant de 0 (faux à coup sûr) à 1 (vrai à coup sûr). L'inférence bayésienne produit une probabilité qui s'interprète comme le degré de confiance à accorder à une cause hypothétique.

3.3 Critères d'arrêt

Pour la plupart des applications des réseaux de neurones, l'erreur d'apprentissage ne converge jamais de manière identique vers zéro. C'est pourquoi nous devons disposer d'autres critères pour décider du moment où il faut arrêter l'entrainement.
Nous pouvons arrêter l'entrainement lorsque l'erreur atteint une certaine limite. Cependant, il est généralement difficile de savoir quel est le niveau d'erreur acceptable. Le critère le plus simple est d'arrêter l'entrainement après

un nombre fixe d'itérations. Comme il est également difficile de savoir combien d'itérations seront nécessaires, le nombre maximum d'itérations est généralement fixé à un niveau raisonnablement élevé. Si les poids n'ont pas convergé après que le nombre maximum d'itérations a été atteint, nous pouvons recommencer l'entraînement, en utilisant les poids finaux du premier passage comme conditions initiales pour le redémarrage.
Un autre critère d'arrêt est la norme du gradient de l'indice de performance. Si cette norme atteint un seuil suffisamment bas, alors l'entrainement peut être arrêtée. Comme le gradient doit être nul au minimum de l'indice de performance, ce critère arrêtera l'algorithme lorsqu'il s'approchera du minimum. Nous pouvons également arrêter l'entraînement lorsque la diminution relative de l'indice de performance entre deux itérations devient faible. Ces deux derniers critères peuvent néanmoins arrêter l'entrainement de manière précoce. Une fois l'entrainement terminé, il est utile de visualiser la courbe de l'indice de performance sur une échelle logarithmique, comme sur la figure 9, pour vérifier la convergence.

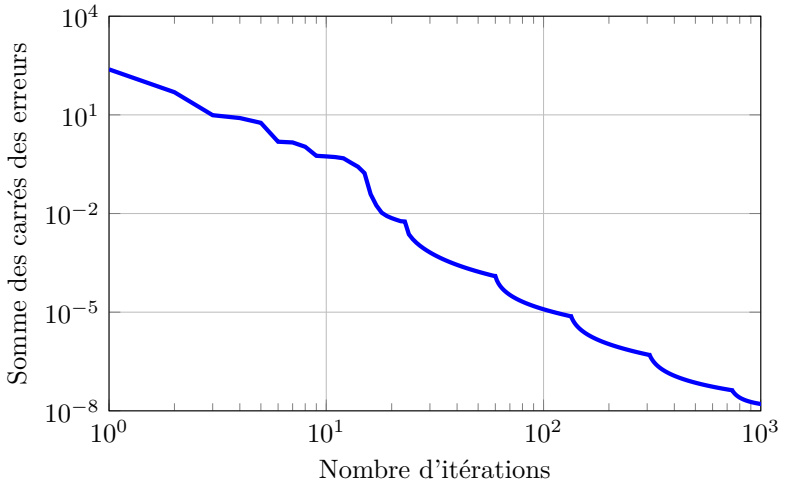


FIGURE 9 – Courbe de performance d'entraînement classique

L'entrainement des réseaux de neurones est un processus itératif. Même après convergence de l'algorithme, l'analyse post-entrainement peut suggérer que le réseau soit modifié et/ou de nouveau entrainé. En outre, plusieurs cycles d'entrainement doivent être effectués pour chaque réseau potentiel afin de s'assurer qu'un minimum global ait été atteint.

3.4 Performance

Pour les réseaux multicouches, l'indice de performance standard est l'erreur quadratique moyenne. D'autres indices peuvent être utilisés, par exemple, l'erreur moyenne absolue.

4 Analyse post-entrainement

Comme nous l'avons vu précédemment, une des applications des réseaux de neurones est la prédiction des valeurs futures de certaines séries temporelles. Pour les problèmes de prédiction, nous utilisons des réseaux dynamiques. Deux concepts importants sont utilisés lors de l'analyse d'un réseau de prédiction formé :

— les erreurs de prédiction ne doivent pas être corrélées dans le temps ;

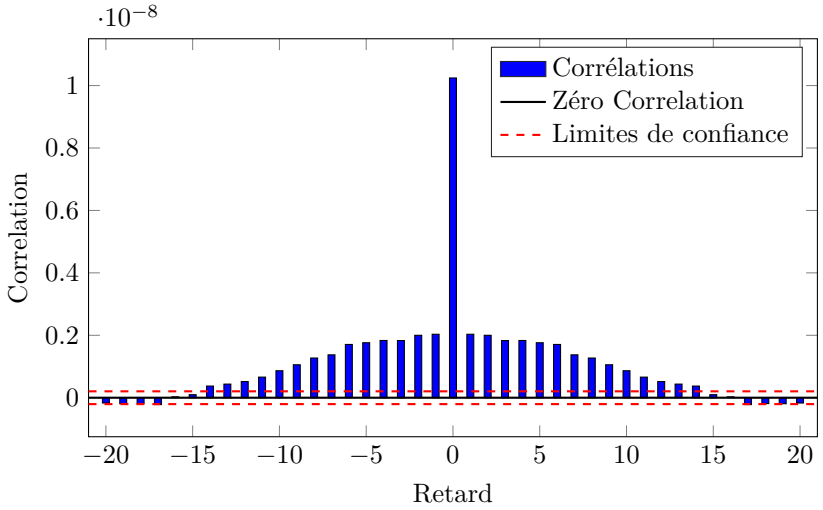
— les erreurs de prédiction ne doivent pas être corrélées avec la séquence d'entrée.

4.1 Autocorrélation dans le temps

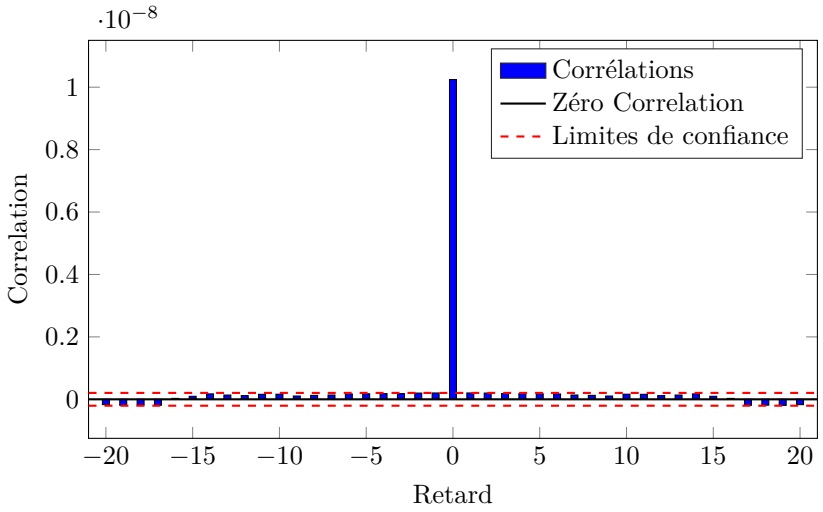
Afin de tester la corrélation des erreurs de prédiction dans le temps, nous pouvons utiliser la fonction d'autocorrélation des échantillons :

$$R_e(\tau) = \frac{1}{Q-\tau} \sum_{t=1}^{Q-\tau} e(t)e(t+\tau) \tag{8}$$

Si les erreurs de prédiction ne sont pas corrélées (bruit blanc - fig. 10), $R_e(\tau)$ devrait être proche de zéro, sauf lorsque $\tau = 0$. Pour déterminer si $R_e(\tau)$ est proche de zéro, nous pouvons fixer un intervalle de confiance d'environ 95 % [5].
La corrélation des erreurs de prédiction peut indiquer que le nombre de retards dans le réseau devrait être augmenté.



(a) $R_e(\tau)$ pour un réseau insuffisamment entrainé



(b) $R_e(\tau)$ pour un réseau correctement entrainé

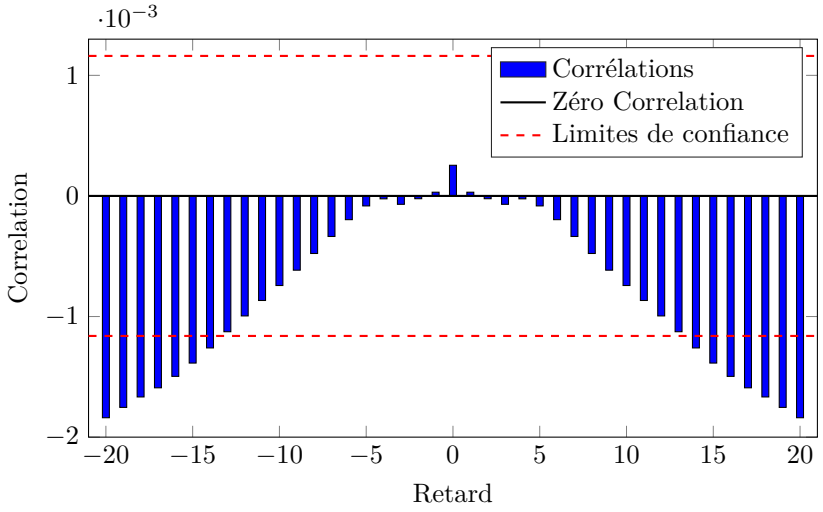
FIGURE 10 – Illustrations de la corrélation des erreurs dans le temps

4.2 Corrélation croisée avec l'entrée

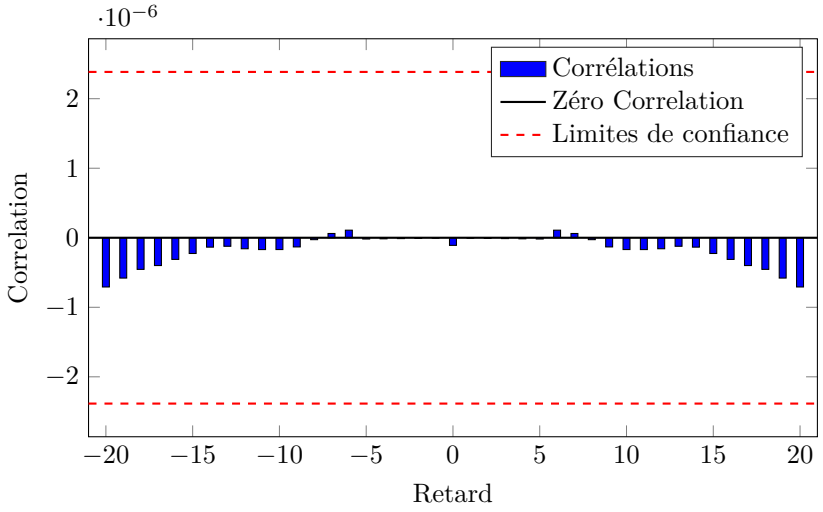
Pour tester la corrélation entre les erreurs de prédiction et la séquence d'entrée, nous pouvons utiliser la fonction de corrélation croisée de l'échantillon :

$$R_{pe}(\tau) = \frac{1}{Q-\tau} \sum_{t=1}^{Q-\tau} p(t)e(t+\tau) \tag{9}$$

S'il n'y a pas de corrélation entre les erreurs prédites et la séquence d'entrée (fig. 11, alors $R_{pe}(\tau)$ devrait être proche de zéro, sauf lorsque $\tau = 0$. Pour déterminer si $R_{pe}(\tau)$ est proche de zéro, nous pouvons fixer un intervalle de confiance d'environ 95 % [5].
Lors de l'utilisation d'un réseau NARX, la corrélation entre l'erreur de prédiction et l'entrée peut suggérer que les retards mis en place dans les chemins d'entrée et de rétroaction devraient être augmentés.



(a) $R_{pe}(\tau)$ pour un réseau insuffisamment entrainé



(b) $R_{pe}(\tau)$ pour un réseau correctement entrainé

FIGURE 11 – Illustrations de la corrélation croisée erreur/entrée

5 Correction de systèmes dynamiques par RNN

Les réseaux neuronaux multicouches ont été appliqués avec succès à l'identification et à la commande de systèmes dynamiques. Plutôt que d'essayer de passer en revue les nombreuses manières dont les réseaux multicouches ont été utilisés dans les systèmes de commande, nous nous concentrerons sur trois contrôleurs de réseau neuronal typiques : *Model Predective Control*, *NARMA-L2 control* et *Model Reference Control*. Ces contrôleurs sont représentatifs de la variété des méthodes courantes d'utilisation des réseaux multicouches dans les systèmes de contrôle. Comme pour la plupart des contrôleurs neuronaux, ils sont basés sur des architectures de contrôle linéaire standard.
Cette partie reprend les grandes lignes de l'article *An introduction to the use of neural networks in control systems* de Hagan *et al.* [2].

5.1 Principe général

L'utilisation de réseaux neuronaux pour le contrôle se fait généralement en deux étapes : l'identification du système et la conception du contrôle. Lors de l'étape d'identification du système, on développe un modèle de réseau neuronal du système à contrôler. Lors de l'étape de conception du contrôle, on utilise le modèle de réseau neuronal du système pour concevoir (ou entrainer) le contrôleur. Dans chacune des trois architectures de contrôle, l'étape d'identification du système est identique. L'étape de conception du contrôle, en revanche, est

différente pour chaque architecture.
La première étape de la commande prédictive consiste donc à entrainer un réseau neuronal pour représenter la boucle ouverte du système. L'erreur de prédiction entre la sortie du système et la sortie du réseau neuronal est utilisée comme signal d'apprentissage du réseau neuronal. Le processus est représenté par la figure 12.

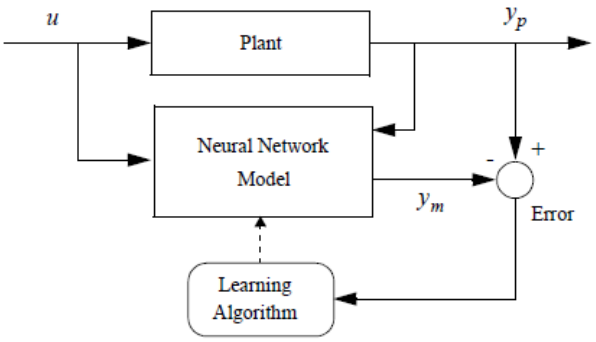


FIGURE 12 – Identification du système par un réseau de neurones

5.2 MRAC

L'architecture de contrôle neuronal dont nous parlerons dans cette sous-section est le *Model Reference Control*. Cette architecture utilise deux réseaux de neurones : un réseau de contrôle et un réseau de modèle du système, comme le montre la figure 13. Le modèle du système est d'abord identifié, puis le contrôleur est entrainé de manière à ce que la sortie du système suive la sortie du modèle de référence.
L'entrainement du contrôleur est coûteux en termes de calcul, car il nécessite l'utilisation de la rétropropagation dynamique. Le côté positif de ce type de contrôle est qu'il s'applique à une classe de systèmes plus importante que les autres types.

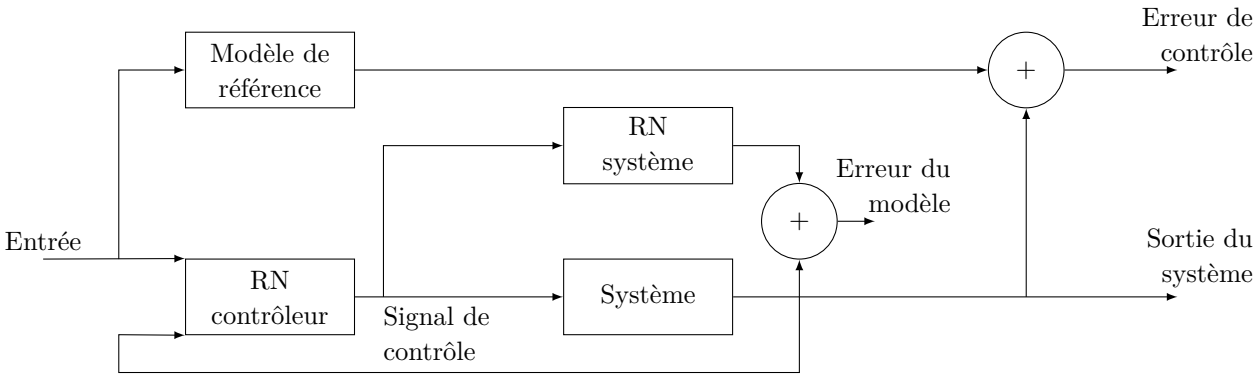


FIGURE 13 – Architecture du *Model Reference Control*

Il existe trois entrées au contrôleur : entrées de référence retardées, sorties de contrôleur retardées (entrées du système) et sorties du système retardées. Pour chacune de ces entrées, nous sélectionnons le nombre de valeurs retardées à utiliser. Généralement, le nombre de retards augmente avec l'ordre du système.
Il y a deux ensembles d'entrées dans le réseau de neurones central : les sorties retardées du contrôleur et les sorties retardées de la centrale.
Il ressort clairement de la figure 14 que la structure est un réseau récurrent. Ce type de réseau est plus difficile à former que les réseaux de rétroaction classiques.
Les données utilisées pour former le contrôleur sont générées en appliquant un signal de référence aléatoire qui consiste en une série d'échelons d'amplitude et de durée aléatoires. Ces données peuvent être générées sans faire fonctionner le système réel, mais en utilisant le réseau neuronal du système.

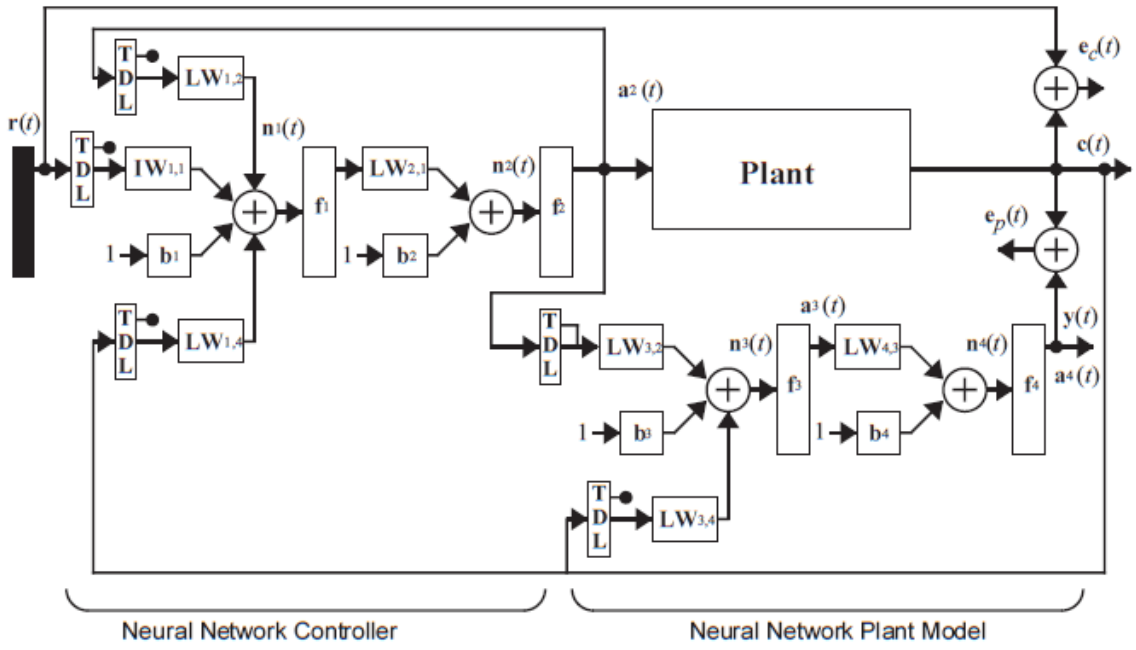


FIGURE 14 – Détails de la structure du MRAC, Hagan *et al.* [2]

Références

[1] M.T. Hagan, H.B. Demuth, M.H. Beale et O. De Jesús : *Neural Network Design*. Martin Hagan, 2014. ISBN 9780971732117.

[2] Martin T Hagan, Howard B Demuth et Orlando De Jesús : An introduction to the use of neural networks in control systems. *International Journal of Robust and Nonlinear Control : IFAC-Affiliated Journal*, 12 (11):959–985, 2002.

[3] Kurt Hornik, Maxwell Stinchcombe et Halbert White : Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[4] Derrick Nguyen et Bernard Widrow : Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. *In 1990 IJCNN International Joint Conference on Neural Networks*, pages 21–26. IEEE, 1990.

[5] George EP Box, Gwilym M Jenkins, Gregory C Reinsel et Greta M Ljung : *Time series analysis : forecasting and control*. John Wiley & Sons, 2015.