ENSMM

Modélisation et discrimination non linéaires à l'aide de réseaux de neurones formels

Guillaume Laurent

2011

Table des matières

Chapitre 1

Avant propos

Ce cours présente l'utilisation de réseaux de neurones formels pour réaliser des modèles de systèmes statiques et dynamiques, ainsi que des classifieurs probabilistes. Seuls les réseaux de neurones à potentiels sont abordés dans ce cours. Il faut savoir qu'il existe de nombreux types de réseaux de neurones comme les réseaux neuro-flou ou les réseaux de fonctions radiales (RFR).

Pour approfondir vos connaissances sur les réseaux de neurones formels, je recommande les ouvrages [10] [4] [9].

Dans ce cours, nous utiliserons les conventions suivantes :
- les caractères minuscules italiques désignent des variables scalaires (par exemple x),
- les caractères majuscules italiques désignent des matrices (par exemple X),
- les caractères petites majuscules italiques désignent des vecteurs (par exemple $\mathbf{x}$),
- les lettres grecques désignent des paramètres réels (par exemple μ),
- les variables soulignées désignent des variables aléatoires (par exemple $\underline{x}$),
- l'opérateur $\mathbb{P}\left[\underline{x} = x\right]$ désigne une probabilité,
- l'opérateur $\mathbb{E}\left[\underline{y}\right]$ désigne l'espérance mathématique de $\underline{y}$.

A la fin du document, un glossaire récapitule toutes les notations.

Chapitre 2

Réseaux de neurones formels

2.1 Neurones formels à potentiels

Un réseau de neurones est composé d'éléments appelés *neurones*. Un neurone à potentiel réalise un fonction non linéaire de ses entrées. Chaque entrées reçoit une information *via* une connexion pondérée (cf. figure 2.1). Le poids d'une connexion est défini par un paramètre w_{ij}. Le calcul de la sortie du neurone s'effectue en deux étapes.

Définition 1. On appelle potentiel (ou activation) du neurone j, la quantité :

$$a_j = w_{0j} + \sum_{i \in \text{pred}(j)} w_{ij} s_i \tag{2.1}$$

w_{0j} est le seuil (ou biais) du neurone j.

La sortie du neurone j est alors :

$$s_j = f_j(a_j) = f_j(w_{j0} + \sum_{i \in \text{pred}(j)} w_{ij} s_i) \tag{2.2}$$

avec f_j une fonction dite *fonction d'activation du neurone* j. Il existe de nombreuses fonctions d'activation :

- la fonction identité (sortie linéaire),
- l'échelon de Heaviside :

$$f(a) = \begin{cases} 0 \text{ si } a < 0 \\ 1 \text{ si } a \geq 0 \end{cases}$$

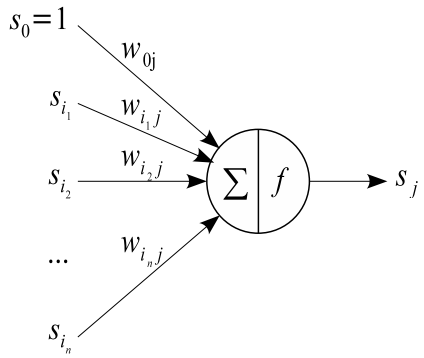


FIGURE 2.1 – Neurone formel à potentiel.

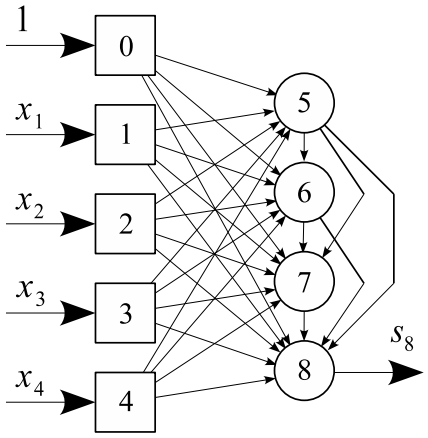


FIGURE 2.2 – Réseau de neurones non bouclé complètement connecté.

– les fonctions « sigmoïdes » :
  – la tangente hyperbolique :

$$f(a) = \mathrm{th}(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$$

  – la fonction logistique :

$$f(a) = \frac{1}{1 + \exp(-a)}$$

  – la fonction arctangente :

$$f(a) = \arctan(a)$$

Les fonctions identités et sigmoïdes sont les plus utilisées car elles sont dérivables.

2.2 Réseaux de neurones non bouclés

Définition 2. Un réseau est dit non bouclé (static/feedforward) ssi son graphe est acyclique.

Un réseau non bouclé réalise une fonction non linéaire de ses entrées. Il existe deux architectures classiques :

– réseaux de neurones complètement connectés,
– réseaux de neurones à couches (RNC) appelés aussi perceptrons multi-couches (PMC).

Dans un réseau de neurones complètement connecté, chaque neurone est commandé par tous les neurones d'indices inférieurs (cf. figure 2.2). Si il y a plusieurs sorties, elles ne sont pas connectées entre elles.

Dans un réseau de neurones à couches, il y a une couche de neurones d'entrée, une ou plusieurs couches de neurones « cachés » et une couche de neurones de sortie (cf. figure 2.3). Chaque neurone est commandé par tous les neurones de la couche précédente.

Dans le cas particulier d'un réseau de neurone à une couche cachée (tanh) et ayant un

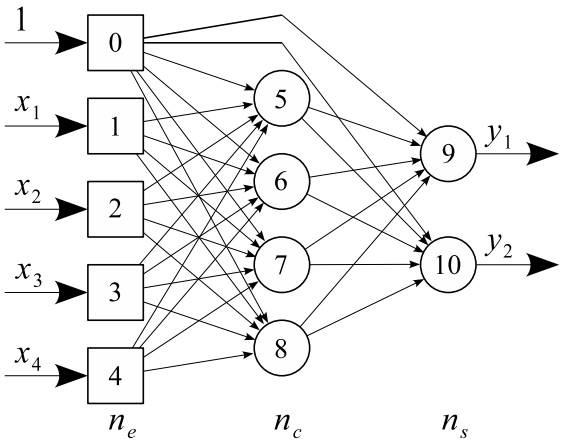


FIGURE 2.3 – Réseau de neurones non bouclé à couches (RNC).

unique neurone de sortie linéaire, si k désigne le neurone de sortie, sa sortie s'écrit :

$$\begin{aligned} s_k &= w_{0k} + \sum_{j \in \text{pred}(k)} w_{jk} s_j \\ &= w_{0k} + \sum_{j \in \text{pred}(k)} w_{jk} \tanh(a_j) \\ &= w_{0k} + \sum_{j \in \text{pred}(k)} w_{jk} \tanh\left(w_{0j} + \sum_{i=1}^{n_e} w_{ij} x_i \right) \end{aligned} \tag{2.3}$$

Propriété 1 (approximation universelle)**.** Toute fonction à valeurs bornées, continue ou non, peut être approchée uniformément dans un intervalle fermé (de $\mathbb{R}^n$) avec une précision arbitraire par un réseau de neurone non bouclé à une couche cachée comportant un nombre fini de neurones à activation sigmoïde et à un neurone de sortie linéaire [6].

Les réseaux de neurones à potentiels sont des approximateurs universels. On a donc une preuve d'existence mais pas de méthode analytique pour trouver un bon réseau !

2.3 Réseaux de neurones bouclés

Définition 3. Un réseau est dit bouclé (dynamic/feedback/recurrent) ssi son graphe est cyclique (chaque cycle comportant au moins un retard unité en temps discret).

Propriété 2. Tout réseau de neurones bouclé peut être mis sous forme canonique, comportant un réseau de neurones non bouclé dont certaines sorties représentent des variables d'état et sont ramenées aux entrées par des bouclages de retard unité [8].

Un réseau bouclé sous forme canonique est défini par un système d'équations récurrentes non linéaires que l'on peut mettre sous la forme d'une représentation d'état :

$$\begin{cases} \widehat{\mathbf{x}}(k+1) = g(\widehat{\mathbf{x}}(k), \boldsymbol{u}(k), \boldsymbol{w}) \\ \widehat{\boldsymbol{y}}(k) = h(\widehat{\mathbf{x}}(k), \boldsymbol{u}(k), \boldsymbol{w}) \end{cases} \tag{2.4}$$

Propriété 3. Tout système dynamique à entrées bornées ayant un état initial fixé peut être approché sur une séquence de taille finie $[0; T]$ avec une précision arbitraire par un réseau de neurone bouclé [16].

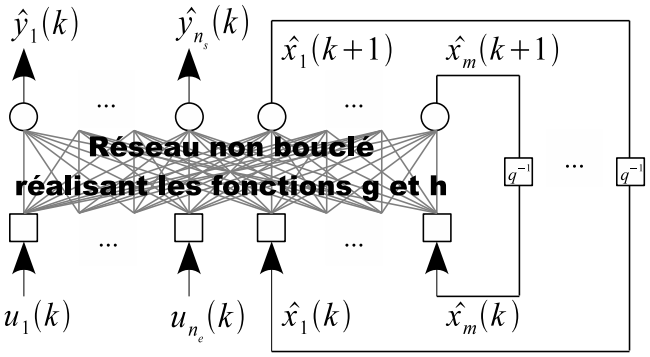


FIGURE 2.4 – Réseau de neurones bouclé sous forme canonique.

2.4 Intérêts des réseaux de neurones pour la modélisation

Propriété 4 (parcimonie)**.** Pour une précision donnée, le nombre de paramètres des approximateurs non linéaires croît linéairement avec le nombre de variables, alors qu'il croît exponentiellement avec ce nombre dans le cas des approximateurs linéaires LP [1].

Les modèles linéaires par rapport à leurs paramètres (LP) se heurtent à l'explosion combinatoire du nombre de leurs paramètres dès que l'on s'intéresse à des espaces de dimensions importantes. Les modèles non linéaires peuvent au contraire être parcimonieux.

Chapitre 3

Modélisation statique

Pour modéliser un système, deux types de modèle existent. Les modèles de connaissance sont issus de l'analyse des observations et mis en équations à l'aide d'une théorie. Ils ont une valeur prédictive et valeur explicative ;

Les modèles « boîte noire » sont réalisés directement à partir de données expérimentales. Ils ont une valeur prédictive mais pas de valeur explicative.

On s'intéresse dans ce cours à la modélisation de type « boîte noire ». Concrètement, on dispose d'un ensemble de N mesures de la grandeur de sortie correspondant à N valeurs de la grandeur d'entrée :

$$\{(\boldsymbol{x}^1, y^1), (\boldsymbol{x}^2, y^2), \quad \dots \quad , (\boldsymbol{x}^N, y^N)\} \tag{3.1}$$

On cherche alors une fonction paramétrée $g(\boldsymbol{x}, \boldsymbol{w})$ qui représente le comportement statique du système « au mieux » (voir plus loin).

On distingue également deux types de modèles « boîte noire ». Les modèles non adaptatifs sont réalisés à partir des mesures réalisées lors d'une campagne initiale (modélisation *hors ligne*). Pendant leur utilisation, les paramètres des modèles non adaptatifs ne sont plus modifiés. Les modèles adaptatifs sont, à l'inverse, corrigés en permanence par rapport aux mesures courantes (modélisation *en ligne*). Pendant leur utilisation, les paramètres de ces modèles évoluent.

3.1 Définitions et objectifs

On considère un système statique entrées/sortie comme représenté en figure 3.1. $\boldsymbol{x}$ est la grandeur d'entrée du système ($\boldsymbol{x} \in \mathbb{R}^n$). y est la grandeur de sortie du système ($y \in \mathbb{R}$). y ne dépend que de $\boldsymbol{x}$ à l'instant présent et pas du passé du système. On suppose que y est mesurable. Nous supposons ici que la sortie est un scalaire pour simplifier les notations et la compréhension du cours, mais tout ce qui suit est valable pour une sortie vectorielle.

Définition 4. La modélisation statique consiste à modéliser le comportement statique d'un système.

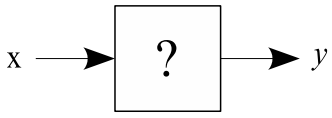


FIGURE 3.1 – Système statique entrées/sortie.

y dépend de $\boldsymbol{x}$ mais cette dépendance n'est pas parfaitement déterministe car des facteurs inconnus ou non mesurables produisent des effets faibles et imprévisibles, de plus il y a souvent un bruit de mesure. On considère alors que y est une réalisation de la variable aléatoire $\underline{y}$. L'objectif est donc d'estimer l'*espérance mathématique* de $\underline{y}$ pour une grandeur d'entrée $\boldsymbol{x}$, notée $\mathbb{E}\left[\underline{y}|\boldsymbol{x}\right]$ et appelée *régression* de $\underline{y}$ par rapport à $\boldsymbol{x}$.

Pour définir analytiquement l'objectif précédent, on utilise un critère. Le critère le plus courant est la somme quadratique des déviations des mesures aux prédictions du modèles (méthode des moindres carrés). On suppose dans ce cas que la mesure de la sortie est entachée d'un bruit gaussien. Il existe d'autres critères qui ne sont pas traités ici mais qui peuvent être plus pertinents dans certains cas.

Définition 5. Soit l'ensemble des N mesures suivant :

$$\{(\boldsymbol{x}^1, y^1), (\boldsymbol{x}^2, y^2), \quad \dots \quad , (\boldsymbol{x}^N, y^N)\} \tag{3.2}$$

On appelle critère (ou coût) quadratique total la fonction :

$$J(\boldsymbol{w}) = \sum_{k=1}^{N} J^k(\boldsymbol{w}) = \sum_{k=1}^{N} \frac{1}{2}\left(y^k - g(\boldsymbol{x}^k, \boldsymbol{w})\right)^2 \tag{3.3}$$

$J^k(\boldsymbol{w})$ est un coût partiel.

g désigne le modèle paramétrique. $g(\boldsymbol{x}, \boldsymbol{w})$ représente donc la valeur de la sortie du réseau de neurones de paramètres $\boldsymbol{w}$ auquel on présente le vecteur $\boldsymbol{x}$ en entrée.

Dans le cas particulier d'un réseau de neurone à une couche cachée (tanh) et ayant un unique neurone de sortie linéaire, si k désigne le neurone de sortie, g s'écrit :

$$g(\boldsymbol{x}, \boldsymbol{w}) = w_{0k} + \sum_{j \in \text{pred}(k)} w_{jk} \tanh\left(w_{0j} + \sum_{i=1}^{n_e} w_{ij} x_i\right) \tag{3.4}$$

L'objectif est donc de trouver les paramètres $\boldsymbol{w}$ tels que le critère quadratique total $J(\boldsymbol{w})$ soit minimum.

3.2 Apprentissage

3.2.1 L'apprentissage : un problème d'optimisation

On cherche à déterminer les poids w_{ij} des connexions qui minimisent le critère $J(\boldsymbol{w})$.

Définition 6. On appelle *apprentissage* la recherche des poids $\boldsymbol{w}$ qui minimisent le critère $J(\boldsymbol{w})$.

On se situe ici dans le cadre de l'apprentissage *supervisé* car on dispose d'un ensemble de couples de mesures de l'entrée et de la sortie correspondante.

Comme il s'agit de modèles non linéaires, il n'y a pas de solutions analytiques. On utilise alors des méthodes d'optimisation numériques.

Les algorithmes du premier ordre modifient le vecteur des paramètres $\boldsymbol{w}$ en fonction du gradient du coût total $\nabla J(\boldsymbol{w})$. Les algorithmes classiques sont :

- plus forte pente (à pas fixe ou optimal),
- gradient conjugué,
- resilient propagation (RPROP) [12].

L'algorithme RPROP est un algorithme de minimisation spécialement conçu pour les réseaux de neurones. Son efficacité est proche de celle des algorithmes du second ordre. L'idée maîtresse est très simple : on ne tient compte que du signe des composantes du gradient de J.

On mémorise un coefficient μ_{ij} d'apprentissage par poids. A chaque itération, chaque μ_{ij} est mis à jour selon la règle suivante :

$$\mu_{ij}(t) = \begin{cases} \eta^+ * \mu_{ij}(t-1) & \text{si } \frac{\partial J}{\partial w_{ij}}(t-1) * \frac{\partial J}{\partial w_{ij}}(t) > 0 \\ \eta^- * \mu_{ij}(t-1) & \text{si } \frac{\partial J}{\partial w_{ij}}(t-1) * \frac{\partial J}{\partial w_{ij}}(t) < 0 \\ \mu_{ij}(t-1) & \text{sinon} \end{cases} \tag{3.5}$$

Les poids sont ensuite mis à jour par :

$$\boldsymbol{w}_{ij}(t) = \boldsymbol{w}_{ij}(t-1) - \mu_{ij}(t) * \text{sign}\left(\frac{\partial J}{\partial w_{ij}}(t)\right) \tag{3.6}$$

Quand la composante du gradient change de signe, on commence par annuler la dernière correction (back-track) avant d'appliquer la nouvelle correction (détails d'implantation dans [12]).

Cependant, on utilisera de préférence un algorithme du second ordre, comme :
- quasi-Newton (BFGS, DFP),
- Gauss-Newton,
- Levenberg-Marquardt.

3.2.2 Calcul du gradient

Tous les algorithmes d'apprentissage nécessitent de calculer le gradient du coût total ou partiel, or :

$$\nabla_{\boldsymbol{w}} J = \nabla_{\boldsymbol{w}} \sum_{k=1}^{N} J^k = \sum_{k=1}^{N} \nabla_{\boldsymbol{w}} J^k \tag{3.7}$$

Il suffit donc de calculer le gradient du coût partiel, deux méthodes de calcul existent :
- calcul du gradient par rétropropagation [15]
- calcul du gradient dans le sens direct [8]

Le calcul dans le sens direct est généralement plus coûteux en temps de calcul, de sorte qu'il n'est utilisé que lorsque le calcul par la rétropropagation est impossible. Nous présentons donc ici le calcul du gradient par rétropropagation.

Rappel :

$$\nabla_{\boldsymbol{w}} J^k = \left(\frac{\partial J^k}{\partial w_{01}}, \frac{\partial J^k}{\partial w_{02}}, \quad \dots \quad\right)^{\mathrm{T}} \tag{3.8}$$

Par définition, w_{ij} n'intervient que dans le calcul du potentiel du neurone j :

$$a_j = w_{0j} + \sum_{p \in \text{pred}(j)}^{n} w_{pj} s_p \tag{3.9}$$

Donc, on peut écrire :

$$\frac{\partial J^k}{\partial w_{ij}} = \frac{\partial J^k}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = \frac{\partial J^k}{\partial a_j} s_i \tag{3.10}$$

Si le neurone j est un neurone de sortie :

$$\begin{aligned}\frac{\partial J^k}{\partial a_j} &= \frac{\partial}{\partial a_j}(\frac{1}{2}(y_j^k - s_j)^2) \\ &= \frac{\partial}{\partial a_j}(\frac{1}{2}(y_j^k - f_j(a_j))^2) \\ &= f_j'(a_j)(f_j(a_j) - y_j^k)\end{aligned} \tag{3.11}$$

De plus, si j est un neurone de sortie linéaire (identité), alors :

$$\frac{\partial J^k}{\partial a_j} = s_j - y_j^k \tag{3.12}$$

Si le neurone j est un neurone d'une couche cachée, J^k ne dépend de a_j que par l'intermédiaire des potentiels des neurones successeurs du neurone j, donc :

$$\frac{\partial J^k}{\partial a_j} = \sum_{m \in \text{succ}(j)} \frac{\partial J^k}{\partial a_m} \frac{\partial a_m}{\partial a_j} \tag{3.13}$$

Or, pour tout neurone m successeur de j, on a :

$$a_m = \sum_{p \in \text{pred}(m)} w_{pm} s_p = \sum_{p \in \text{pred}(m)} w_{pm} f_p(a_p) = w_{jm} f_j(a_j) + \sum_{p \in \text{pred}(m), p \neq j} w_{im} f_p(a_p) \tag{3.14}$$

Donc :

$$\frac{\partial a_m}{\partial a_j} = w_{jm} f_j'(a_j) \tag{3.15}$$

Et finalement :

$$\frac{\partial J^k}{\partial a_j} = f_j'(a_j) \sum_{m \in \text{succ}(j)} \frac{\partial J^k}{\partial a_m} w_{jm} \tag{3.16}$$

Résumé du calcul du gradient par rétropropagation :

1. calcul du potentiel et de la sortie de tous les neurones (sens direct) pour la mesure k,
2. calcul des dérivées partielles relatives à l'activation des neurones de sorties,
3. calcul des dérivées partielles relatives à l'activation des neurones cachés (sens rétrograde si plusieurs couches),
4. calcul du gradient du coût partiel pour la mesure k pour tous les poids.

Pour que les algorithmes d'optimisation fonctionne bien, il est important d'initialiser avec soins les paramètres initiaux du réseaux [3]. On adopte généralement les règles suivantes :

- initialisation des seuils w_{0j} à 0,
- initialisation des autres poids : non nulle et bornée pour s'assurer que les potentiels aient une variance de 1 (il est classique d'effectuer un tirage aléatoire uniforme entre $-w_{max}$ et w_{max} avec $w_{max} < \frac{3}{10\sqrt{n_e}}$).

3.3 Méthodologie de conception d'un modèle statique neuronal

3.3.1 Prétraitements des données

La plupart des données doivent être prétraitées avant d'être utilisées avec un réseau de neurones. Le tableau suivant propose un codage dans $\mathbb{R}^n$ des différents types de valeurs que l'on peut rencontrer.

Type	**Codage**
valeur réelle bornée	une composante centrée et normée
valeur binaire	une composante à valeur dans $\{0;1\}$ ou $\{-1;1\}$
valeur ordinale	une composante réelle centrée et normée
valeur nominale (qualitative)	autant de composantes binaires que de valeurs possibles (codage « un parmi c » ou codage « grand-mère »)

On peut généralement réduire de façon significative la dimension des entrées à l'aide d'une analyse en composantes principales ou curvilignes. L'apprentissage n'en sera que plus rapide et le réseau plus petit.

Enfin, dans le cas d'un système à plus d'une sortie, on procédera de la manière suivante :
- si les sorties sont linéairement dépendantes, on peut utiliser un unique réseau,
- si les sorties sont linéairement indépendantes, on utilisera autant de réseaux que de sorties.

3.3.2 Élaboration du modèle

La démarche d'élaboration d'un modèle suit généralement les étapes suivantes :

1. choix de la structure du réseau,
2. choix du nombre de neurones cachés,
3. choix de l'algorithme d'apprentissage (on choisira un algorithme du second ordre et à défaut l'algorithme RPROP),
4. réalisation de plusieurs apprentissages pour éviter les minima locaux,
5. retour aux étapes 1 ou 2 en fonction des résultats de la validation,
6. élagage.

3.3.3 Validation et sélection de modèles

Le but de la validation est de vérifier la réaction du réseau et d'éviter le sur-apprentissage. De nombreuse méthodes de validation existent : la validation simple, la validation croisée, le bootstrapping, etc.

La validation simple consiste à séparer l'ensemble des mesures en deux sous-ensembles : un ensemble d'apprentissage (généralement 2/3 des données) et un ensemble de validation (1/3 des données). On calcule ensuite le critère total sur les deux ensembles après apprentissage sur l'ensemble d'apprentissage, et ce pour un nombre croissant de neurones cachés (cf. figure 3.2). Le graphe obtenu permet de choisir la complexité évitant le surapprentissage.

3.4 Conclusion

Les modèles neuronaux ne nécessitent pas la recherche d'une solution analytique et peuvent donc modéliser *a priori* tout types de systèmes non linéaires. Il nécessitent en revanche la recherche d'une solution numérique par optimisation (problèmes de minima locaux et du temps

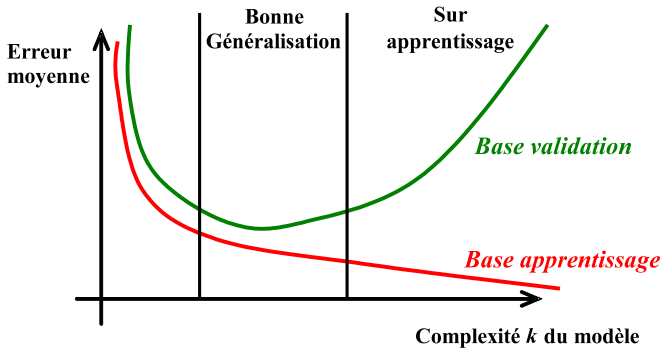


FIGURE 3.2 – Graphique de validation simple

de calcul). Leur mise en oeuvre est relativement indépendante de la complexité du système à modéliser. On remarque aussi que les modèles neuronaux sont très robustes aux bruits. Enfin, un intérêt majeur est leur compacité pour des problèmes de grandes dimensions.

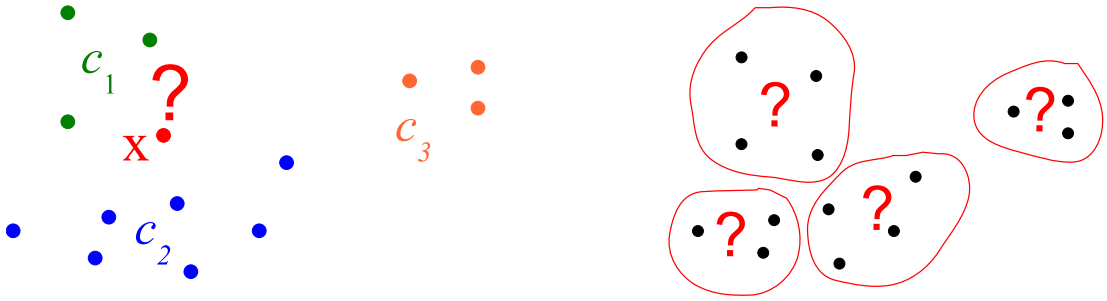
Chapitre 4

Discrimination

Le classement (ou discrimination) affecte un nouvel objet dans une des classes existantes. Le traitement des données est supervisé : on dispose au préalable d'un ensemble de N objets classés (ou étiquetés) en c sous-ensembles (classes).

La classification (ou partitionnement ou encore *clustering*) a pour objectif la constitution de classes d'objets à partir d'un ensemble de N descripteurs non classés. Le traitement est non supervisé (les objets ne sont pas étiquetés).

Ce chapitre traite de l'usage des réseaux de neurones dans le cas du classement.



(a) Classement (b) Classification

FIGURE 4.1 – Classement *versus* classification

4.1 Définitions

4.1.1 Notion de descripteur

Définition 7. On appelle *descripteur* un ensemble de valeurs quantitatives ou qualitatives représentant un objet

Un descripteur peut contenir :
- des valeurs réelles (bornées),
- des valeurs binaires,
- des valeurs nominales (qualitatives),
- des valeurs ordinales.

Exemple { *marque=peugeot, kilométrage=125000, couleur=rouge, cabriolet=faux, état=bon, etc.*}

Dans la plupart des cas, il est nécessaire pour les besoins du classifieur de coder un descripteur dans $\mathbb{R}^n$.

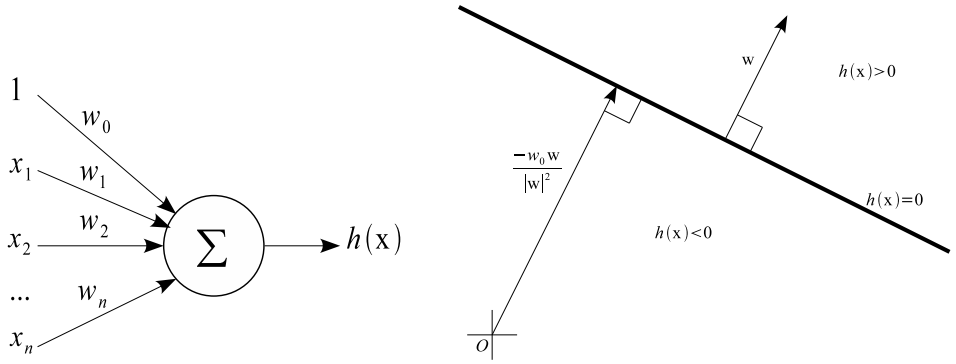


FIGURE 4.2 – Un perceptron et son interprétation géométrique.

4.1.2 Classifieurs séparateurs et probabilistes

Définition 8. Un classifieur séparateur est une fonction qui affecte une classe à un descripteur (prend une décision).

Exemple *On estime qu'un individu mesurant 1,80m est un homme*

Un descripteur est la réalisation d'une variable aléatoire. On peut estimer les probabilités conditionnelles d'appartenance *a posteriori* sachant la réalisation du descripteur : c'est l'objectif d'un classifieur probabiliste.

Définition 9. Un classifieur probabiliste est une fonction qui calcule les probabilités d'appartenance *a posteriori* (sachant le descripteur) d'un objet à chacune des classes (nécessite un étage supérieur pour prendre une décision).

Exemple *On estime qu'un individu mesurant 1,80m a une probabilité de 0,05 d'être une femme et de 0,95 d'être un homme*

4.2 Classifieurs séparateurs définis par un hyperplan pour un problème à deux classes

Dans cette section, on s'intéresse aux problèmes à deux classes.

4.2.1 Définitions

On considère qu'un objet est représenté par un vecteur $\mathbf{x}$. On dispose d'un ensemble d'apprentissage $\{\mathbf{x}^k, y^k\}_{k=1 \text{ à } N}$, où les y^k appartiennent à $\{-1; 1\}$.

Définition 10. On appelle hyperplan séparateur un hyperplan qui sépare parfaitement les deux classes.

Il n'existe pas toujours d'hyperplan séparateur, on se contentera alors de chercher l'hyperplan le plus discriminant.

Les premiers algorithmes de recherche d'un hyperplan séparateur par apprentissage ont été proposés à la fin des années 1950 [13, 17].

4.2.2 Algorithme du perceptron

On définit un classifieur à l'aide d'un neurone linéaire appelé *perceptron* [13].

Son potentiel s'écrit :

$$h(\boldsymbol{x}) = w_0 + \sum_{i=1}^{n} w_i x_i = w_0 + \boldsymbol{x}.\boldsymbol{w} \tag{4.1}$$

L'hyperplan $h(\boldsymbol{x}) = 0$ est l'hyperplan associé au perceptron.
On cherche à minimiser la fonction de coût définie par :

$$J_p(\boldsymbol{w}) = \sum_{k=1}^{N} J_p^k(\boldsymbol{w}) \quad \text{avec} \quad \begin{cases} J_p^k(\boldsymbol{w}) = -y^k h(\boldsymbol{x}^k) & \text{si } h(\boldsymbol{x}^k)y^k \leq 0 \\ J_p^k(\boldsymbol{w}) = 0 & \text{sinon} \end{cases} \tag{4.2}$$

L'algorithme du perceptron consiste à modifier les paramètres à chaque présentation d'un exemple selon la règle suivante :
- si l'exemple est mal classé ($h(\boldsymbol{x}^k)y^k \leq 0$), on applique $\boldsymbol{w}(k) = \boldsymbol{w}(k-1) + y^k\boldsymbol{x}^k$ et $w_0(k) = w_0(k-1) + y^k$,
- si l'exemple est bien classé ($h(\boldsymbol{x}^k)y^k > 0$), les paramètres sont inchangés.

Les paramètres initiaux sont choisis au hasard (ou nuls).

Propriété 5. l'algorithme du perceptron converge vers un hyperplan séparateur en un nombre fini d'itérations [2] ssi les exemples sont linéairement séparables.

Si les exemples ne sont pas linéairement séparables, l'algorithme ne converge pas. Deux cas indistinguables sont donc possibles : les exemples sont linéairement séparables mais on n'a pas encore atteint la convergence, les exemples ne sont pas linéairement séparables.

Pour cette raison, d'un point de vue pratique, on utilise peu cet algorithme qui est surtout intéressant d'un point de vue historique.

4.2.3 Algorithmes des moindres carrés

Algorithmes des moindres carrés avec un neurone linéaire

On cherche à minimiser la fonction de coût quadratique relative au potentiel du neurone :

$$J_q(\boldsymbol{w}) = \sum_{k=1}^{N} (y^k - h(\boldsymbol{x}^k))^2 \tag{4.3}$$

On peut alors trouver une solution en utilisant :
- la méthode des moindres carrés (résolution des équations normales),
- une méthode de minimisation itérative ou récursive (Widrow-Hoff).

L'hyperplan obtenu est nommé hyperplan « MC-potentiel ».

Que les exemples soient linéairement séparables ou non, la fonction de coût sur le potentiel possède un seul minimum et on obtient donc une solution. Malgré tout, même si les exemples sont linéairement séparables, l'hyperplan « MC-potentiel » ne les séparent pas nécessairement : cela dépend de la distribution des exemples.

Algorithmes des moindres carrés avec un neurone à sigmoïde

On cherche à minimiser la fonction de coût quadratique relative à la sortie d'un neurone à activation sigmoïde :

$$J_q(\boldsymbol{w}) = \sum_{k=1}^{N} \Big(y^k - f(h(\boldsymbol{x}^k))\Big)^2 \tag{4.4}$$

On peut alors trouver une solution en utilisant un algorithme de minimisation itératif ou récursif. L'hyperplan obtenu est nommé hyperplan « MC-sigmoïde ».

Si les exemples sont linéairement séparables, le coût peut être annulé en saturant la sigmoïde jusqu'à quasiment obtenir un échelon. L'hyperplan obtenu est généralement positionné de manière équilibré par rapport aux exemples marginaux.

Si les exemples ne sont pas linéairement séparables, l'hyperplan se positionne dans la zone de recouvrement et la sigmoïde n'est pas saturée.

4.2.4 Algorithme de Ho et Kashyap

Si les exemples sont linéairement séparables, il existe un vecteur « positif » $\boldsymbol{b}$ tel que :

$$\boldsymbol{b} = \begin{pmatrix} y^1 h(\boldsymbol{x}^k) \\ \vdots \\ y^N h(\boldsymbol{x}^N) \end{pmatrix} = \begin{pmatrix} y^1 & y^1 x_1^1 & \cdots & y^1 x_n^1 \\ \vdots & & \ddots & \vdots \\ y^N & y^N x_1^N & \cdots & y^N x_n^N \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ w_n \end{pmatrix} = M\boldsymbol{w} \tag{4.5}$$

On cherche donc à annuler le coût suivant :

$$J_s(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{2}\|M\boldsymbol{w} - \boldsymbol{b}\|^2 \tag{4.6}$$

Pour tout $\boldsymbol{b}$ « positif », on peut déterminer un vecteur $\boldsymbol{w}$ optimal, en écrivant :

$$\boldsymbol{w} = (M^{\mathrm{T}}M)^{-1}M^{\mathrm{T}}\boldsymbol{b} \tag{4.7}$$

Il reste à estimer $\boldsymbol{b}$ par descente de gradient sous la contrainte de garder un vecteur « positif », on a :

$$\nabla_{\boldsymbol{b}} J_s = -(M\boldsymbol{w} - \boldsymbol{b}) = -\boldsymbol{e} \tag{4.8}$$

La règle de Ho et Kashyap [5] consiste à modifier itérativement $\boldsymbol{b}$ par :

$$\boldsymbol{b}(k+1) = \boldsymbol{b}(k) + \mu(\boldsymbol{e} + |\boldsymbol{e}|) \tag{4.9}$$

Propriété 6. L'algorithme converge vers une valeur nulle de J_s si les exemples sont séparables et vers une valeur strictement positive sinon.

Cet algorithme est principalement employé pour déterminer si les exemples sont linéairement séparables ou non.

4.3 Classifieurs probabilistes neuronaux

4.3.1 Définitions

La probabilité *a priori* d'appartenance d'un objet à une classe ne tient pas compte de son descripteur.

Exemple *Un individu quelconque a une probabilité de 0,51 d'être une femme et de 0,49 d'être un homme*

La probabilité *a posteriori* d'appartenance d'un objet à une classe tient compte de la valeur de son descripteur.

Exemple *Un individu mesurant 1,80m a une probabilité de 0,05 d'être une femme et de 0,95 d'être un homme*

Un classifieur probabiliste a pour objectif de calculer la probabilité *a posteriori* d'appartenance d'un objet à une classe.

Si l'on considère un problème à deux classes exclusives $\{c_1; c_2\}$ et que l'on note P_1 et P_2, les probabilités d'appartenance *a priori*, c'est-à-dire :

$$P_1 = \mathbb{P}[\underline{c} = c_1]$$
$$P_2 = \mathbb{P}[\underline{c} = c_2]$$

Si l'objet appartient à la classe c_1, le vecteur aléatoire $\underline{\boldsymbol{x}}$ possède la densité de probabilité conditionnelle $f_{\underline{\boldsymbol{x}}|c_1}(\boldsymbol{x})$.

Si l'objet appartient à la classe c_2, le vecteur aléatoire $\underline{\boldsymbol{x}}$ possède la densité de probabilité conditionnelle $f_{\underline{\boldsymbol{x}}|c_2}(\boldsymbol{x})$.

Puisqu'il n'y a que deux classes exclusives, on a la relation :

$$P_1 f_{\underline{\boldsymbol{x}}|c_1}(\boldsymbol{x}) + P_2 f_{\underline{\boldsymbol{x}}|c_2}(\boldsymbol{x}) = f_{\underline{\boldsymbol{x}}}(\boldsymbol{x}) \tag{4.10}$$

$f_{\underline{\boldsymbol{x}}}(\boldsymbol{x})$ est la densité de probabilité inconditionnelle de $\boldsymbol{x}$

La relation de Bayes fournit l'expression des probabilités *a posteriori* :

$$\mathbb{P}[\underline{c} = c_i|\boldsymbol{x}] = \frac{P_i \; f_{\underline{\boldsymbol{x}}|c_i}(\boldsymbol{x})}{f_{\underline{\boldsymbol{x}}}(\boldsymbol{x})} \tag{4.11}$$

Remarque : on a bien $\mathbb{P}[\underline{c} = c_1|\boldsymbol{x}] + \mathbb{P}[\underline{c} = c_2|\boldsymbol{x}] = 1$ et $0 \leq \mathbb{P}[\underline{c} = c_i|\boldsymbol{x}] \leq 1$

Le classifieur séparateur de Bayes attribue $\boldsymbol{x}$ à la classe c_1 si :

$$\mathbb{P}[\underline{c} = c_1|\boldsymbol{x}] > \mathbb{P}[\underline{c} = c_2|\boldsymbol{x}] \tag{4.12}$$

Une propriété du classifieur de Bayes est qu'il minimise l'erreur de classement.

Pour un problème à c classes, si $\{P_i = \mathbb{P}[\underline{c} = c_i]\}_{i=1 \text{ à } c}$ est l'ensemble des probabilités *a priori* des c classes, et si $\{f_{\underline{\boldsymbol{x}}|c_i}(\boldsymbol{x})\}_{i=1 \text{ à } c}$ est l'ensemble des densités conditionnelles relatives à chacune des c classes, alors la densité de probabilité inconditionnelle de $\boldsymbol{x}$ est donnée par :

$$f_{\underline{\boldsymbol{x}}}(\boldsymbol{x}) = \sum_{i=1}^{c} P_i f_{\underline{\boldsymbol{x}}|c_i}(\boldsymbol{x})$$

La relation de Bayes fournit alors les probabilités *a posteriori* :

$$\mathbb{P}[\underline{c} = c_i|\boldsymbol{x}] = \frac{P_i \; f_{\underline{\boldsymbol{x}}|c_i}(\boldsymbol{x})}{f_{\underline{\boldsymbol{x}}}(\boldsymbol{x})}$$

Un objet ayant un descripteur $\boldsymbol{x}$ est attribué à la classe c_j si :

$$\forall i \neq j \quad \mathbb{P}[\underline{c} = c_j|\boldsymbol{x}] > \mathbb{P}[\underline{c} = c_i|\boldsymbol{x}] \tag{4.13}$$

La relation de Bayes permet donc de calculer les probabilités *a posteriori* si l'on connaît :
- les probabilités *a priori*,
- les densités de probabilité conditionnelles.

Malheureusement pour un problème réel on ne les connaît généralement pas ! Deux approches sont néanmoins possibles. Si l'on connaît un ensemble de descripteurs déjà classés, on peut :

- estimer les probabilités *a priori* et les densités de probabilité conditionnelles ; pour estimer les densités de probabilité conditionnelles, il existe deux catégories de méthodes : les méthodes paramétriques font une hypothèse sur la forme des densités (une loi normale par exemple) et les méthodes non paramétriques calculent une estimation des densités sur la base des exemples (méthode des noyaux, k plus proches voisins, etc.) ;
- estimer directement les probabilités *a posteriori* sans utiliser la relation Bayes.

C'est dans cette dernière catégorie que se situent les classifieurs probabilistes neuronaux.

4.3.2 Problèmes à deux classes

On considère un problème à deux classes.

On considère qu'un objet est représenté par un vecteur de la forme :

$$\boldsymbol{x} = (x_1 \; x_2 \; \ldots \; x_n)^{\mathrm{T}} \tag{4.14}$$

On dispose d'un ensemble d'apprentissage $\{\boldsymbol{x}^k, y^k\}_{k=1 \text{ à } N}$, où les y^k appartiennent à $\{0; 1\}$.

Comme il n'y a que deux classes, on a :

$$\mathbb{P}\left[\underline{y} = 1|\boldsymbol{x}\right] = 1 - \mathbb{P}\left[\underline{y} = 0|\boldsymbol{x}\right] \tag{4.15}$$

Il suffit donc d'estimer une des deux probabilités.

On utilise généralement le critère de coût quadratique [1]. :

$$J(\boldsymbol{w}) = \sum_{k=1}^{N} J^k(\boldsymbol{w}) = \sum_{k=1}^{N} \frac{1}{2}\left(y^k - g(\mathbf{x}^k, \boldsymbol{w})\right)^2 \tag{4.16}$$

g est l'estimation de $\mathbb{E}\left[\underline{y}|\boldsymbol{x}\right]$.

Comme l'espérance mathématique d'un variable aléatoire discrète est définie par :

$$\mathbb{E}\left[\underline{y}\right] = \sum_{i=1}^{c} y_i \mathbb{P}\left[\underline{y} = y_i\right] \tag{4.17}$$

Comme $y_1 = 0$ et $y_2 = 1$, on a :

$$\mathbb{E}\left[\underline{y}|\boldsymbol{x}\right] = \mathbb{P}\left[\underline{y} = 1|\boldsymbol{x}\right] \tag{4.18}$$

L'objectif est donc de trouver les paramètres $\boldsymbol{w}$ tels que le critère quadratique total $J(\boldsymbol{w})$ soit minimum.

La fonction g est calculée par un réseau de neurones. On choisit un réseau avec :

- n entrées (n étant la dimension du vecteur $\boldsymbol{x}$),
- une ou plusieurs couches cachées à activation sigmoïde,
- un neurone de sortie à fonction d'activation « logistique » (à valeur dans $[0; 1]$).

4.3.3 Problèmes à plus de deux classes

Pour un problèmes à c classes, trois approches sont envisageables :

- utiliser un seul réseau comportant c sorties,
- utiliser c réseaux « grand-mère » (chaque réseau donnant la probabilité d'appartenance à une classe par rapport à toutes les autres),
- utiliser $c(c-1)/2$ réseaux classifieurs de paires de classes (chaque réseau donnant la probabilité d'appartenance à une classe par rapport à une autre).

1. On utilise aussi couramment le critère de coût de l'entropie croisé.

Mise au point d'un seul classifieur

Dans ce cas, on choisit un réseau avec :
- n entrées (n étant la dimension du vecteur $\boldsymbol{x}$),
- une ou plusieurs couches cachées à activation sigmoïde,
- c neurones de sortie à fonction d'activation « logistique » ou à fonction d'activation de Gibbs (sorties normalisées),

Le codage des sorties est du type « grand-mère » à valeur dans $[0;1]^c$.
Dans ces conditions, le critère quadratique s'écrit :

$$J(\boldsymbol{w}) = \sum_{k=1}^{N} \sum_{j=1}^{c} \frac{1}{2} \left(y_j^k - g_j(\boldsymbol{x}^k, \boldsymbol{w}) \right)^2 \tag{4.19}$$

Mise au point de c réseaux « grand-mère »

On utilise un réseau par classe. La sortie $g_j(\boldsymbol{x}, \boldsymbol{w})$ d'un réseau donne une estimation de la probabilité *a posteriori* que $\boldsymbol{x}$ appartienne à la classe j.
On procède à l'apprentissage de chaque réseau comme pour un problème à deux classes.
Les sorties n'étant pas contraintes, on a bien souvent :

$$\sum_{j=1}^{c} g_j(\boldsymbol{x}, \boldsymbol{w}) \neq 1 \tag{4.20}$$

La meilleure solution consiste à normaliser et écrivant :

$$\mathbb{P}\left[\widehat{\underline{c} = c_i} | \boldsymbol{x}\right] = \frac{g_i(\boldsymbol{x}, \boldsymbol{w})}{\sum_{j=1}^{c} g_j(\boldsymbol{x}, \boldsymbol{w})} \tag{4.21}$$

Mise au point de $c(c-1)/2$ réseaux classifieurs de paires de classes

Chaque réseau calcule une estimation de la probabilité *a posteriori* d'appartenance à une classe par rapport à une autre que l'on note :

$$P_{ij} = \mathbb{P}\left[\underline{c} = c_i | \boldsymbol{x} \in c_i \cup c_j\right] \tag{4.22}$$

On montre [11] que la probabilité *a posteriori* que l'on désire estimer s'écrit :

$$\mathbb{P}\left[\underline{c} = c_i | \boldsymbol{x}\right] = \frac{1}{2 - c + \sum_{i=1, i \neq j}^{c} \frac{1}{P_{ij}}} \tag{4.23}$$

Ici encore, il peut être nécessaire de normaliser ces estimations.
Cette méthode fournit des informations utiles sur la représentation des objets. On peut constater par exemple que deux classes sont linéairement séparables ou non, que certaines entrées sont plus significatives pour distinguer deux classes...

4.4 Méthodologie de conception

1. Traitements des entrées :
   - choix d'un bonne représentation des objets (descripteur),
   - réduction de la dimension des entrées,
   - données suffisantes (Cover).
2. Élaboration du classifieur :

- tester la séparabilité linéaire de chaque paire de classes (avec Ho et Kashyap),
- pour les classes séparables linéairement (ou presque) utiliser les approches « classiques »,
- pour les classes non séparables linéairement utiliser un classifieur neuronal,
- calculer les probabilités d'appartenance *a posteriori* avec la formule de Price,
- fixer des seuils de décision pour définir des classes de rejet.

3. Validation :
   - validation simple, validation croisée, *etc.*
   - sélection des entrées et des classifieurs.

On estime généralement la performance d'un classifieur séparateur par le taux d'exemples mal classés sur un ensemble de validation (ou par validation croisée) qui correspond à une estimation de la probabilité d'erreur de classement :

$$\widehat{p} = \frac{m}{N_v} \tag{4.24}$$

avec m le nombre d'exemples mal classés.

Il faut cependant relativiser l'importance à accorder à cette estimation comme le montre ce calcul des intervalles de confiance (à 95%) sur quelques cas de figure :

	$N_v = 50$	$N_v = 250$
si $\widehat{p} = 0\%$	$[0\%; 8\%]$	$[0\%; 2\%]$
si $\widehat{p} = 5\%$	$[1\%; 15\%]$	$[2,5\%; 9\%]$

Chapitre 5

Modélisation dynamique

Dans ce chapitre, on s'intéresse à la modélisation d'un processus dynamique en temps discret. Pour simplifier les notations, nous supposons ici que l'entrée et la sortie sont des scalaires.

Le processus est soumis à une commande et à des perturbations (cf. figure 5.1). On distingue deux catégories de perturbations, les perturbations mesurables et les perturbations non mesurables (bruits). Les commandes et les perturbations mesurables seront considérés comme des entrées du modèle.

5.1 Définitions et objectifs

5.1.1 Modèles-hypothèses

En ce qui concerne les perturbations non mesurables, on fait généralement une des trois hypothèses suivantes :

- hypothèse « bruit de boucle » (bruit d'état, *equation error*),
- hypothèse « bruit de sortie » (*output error*),
- hypothèse « bruit de sortie et bruit de boucle ».

On fait également des hypothèses sur la structure du modèle. On parle alors de *modèle-hypothèse*. Un modèle-hypothèse traduit le fait que l'on fait une hypothèse sur la structure du système (ordre, retard, etc.) et sur le type de bruit (bruit de sortie, bruit de boucle, etc.).

On considère généralement deux architectures de modèles-hypothèses :

- modèles-hypothèses entrée-sortie (transfer-function model) : la sortie dépend de la commande, des commandes passées et des sorties passées,
- modèles-hypothèses d'état (state-space model) : la sortie dépend de la commande et d'une variable interne appelée *état*.

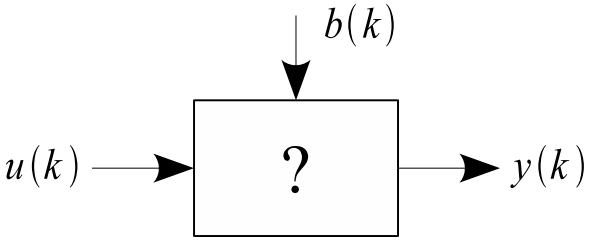


FIGURE 5.1 – Processus soumis à une commande u et à une perturbation b

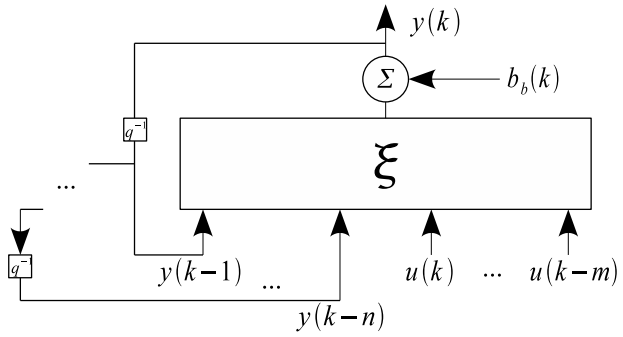


FIGURE 5.2 – Modèle-hypothèse E/S avec bruit de boucle

5.1.2 Prédicteurs

Pour un modèle linéaire les deux représentations sont équivalentes, pour un modèle non linéaire la représentation d'état est plus générale et plus parcimonieuse, mais la représentation entrée-sortie est plus simple à mettre en oeuvre [7].

L'objectif général de la modélisation dynamique est de trouver un modèle-hypothèse adapté au processus et d'identifier les paramètres de ce modèle afin de prédire au mieux la sortie du processus.

Définition 11. Un *prédicteur* fournit une estimation $\widehat{y}(k)$ de l'espérance mathématique de la sortie $y(k)$ du processus à l'instant k en fonction d'informations passées observées (commandes, sorties).

5.1.3 Modélisation dynamique « boîte noire »

La modélisation dynamique « boîte noire » consiste à trouver les paramètres d'un modèle qui minimisent les erreurs de prédictions. Pour définir analytiquement cet objectif, on utilise un critère. Le critère le plus courant est l'erreur quadratique moyenne. Comme pour la modélisation statique, il existe d'autres critères qui ne sont pas traités ici mais qui peuvent être plus pertinents dans certains cas.

Définition 12. On dispose d'un ensemble de N mesures *consécutives* de la grandeur de sortie et de la grandeur de commande du processus :

$$\{(u(1), y(1)), (u(2), y(2)), \quad \dots \quad , (u(N), y(N))\} \tag{5.1}$$

On appelle *erreur quadratique moyenne*, la fonction :

$$J = \sum_{k=1}^{N} \frac{1}{2N} \left(y(k) - \widehat{y}(k)\right)^2 = \sum_{k=1}^{N} J^k \tag{5.2}$$

5.2 Modèles-hypothèses entrée-sortie et prédicteurs associés

5.2.1 Modèle-hypothèse E/S avec bruit de boucle

Ce modèle (cf. figure 5.2) postule l'existence d'une fonction (déterministe) ξ telle que :

$$y(k) = \xi(y(k-1), ..., y(k-n), u(k), ..., u(k-m)) + b_b(k) \tag{5.3}$$

avec :

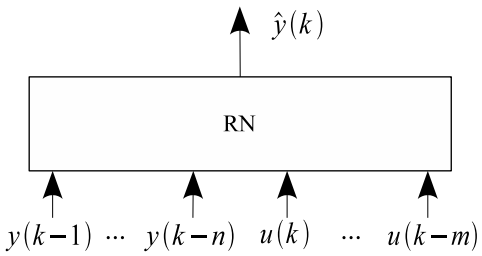


FIGURE 5.3 – Prédicteur neuronal associé au modèle-hypothèse E/S avec bruit de boucle

– $y(k)$ la sortie,
– n l'ordre du modèle,
– $u(k)$ la commande,
– m l'ordre relatif aux commandes du modèle,
– $b_b(k)$ un bruit de boucle additif supposé blanc gaussien.

Le prédicteur théorique associé est :

$$\widehat{y}(k) = g(y(k-1), ..., y(k-n), u(k), ..., u(k-m)) \tag{5.4}$$

Ce prédicteur est optimal car si g correspond exactement à ξ, la variance de l'erreur entre le prédicteur et la sortie est égale au bruit ($\tilde{e}(k) = y(k) - \widehat{y}(k) = b_b(k)$).

On remarque que ce prédicteur est « non bouclé ». Dans le cas linéaire (modèle ARX), g s'écrit :

$$\widehat{y}(k) = -a_1 y(k-1) - ... - a_n y(k-n) + b_0 u(k) + ... + b_m u(k-m) \tag{5.5}$$

Dans le cas d'un prédicteur neuronal, g est calculé par un réseau de neurones non bouclé prenant en entrée le vecteur $\boldsymbol{x} = \begin{pmatrix} y(k-1) & \cdots & y(k-n) & u(k) & \cdots & u(k-m) \end{pmatrix}$ (cf. figure 5.3).

Pour trouver les paramètres minimisant l'erreur quadratique moyenne, on procède à un apprentissage type « modélisation statique » (voir chapitre précédent) avec les données suivantes :

$$X = \begin{pmatrix} y(n) & \cdots & y(1) & u(n+1) & \cdots & u(n-m+1) \\ y(n+1) & \cdots & y(2) & u(n+2) & \cdots & u(n-m+2) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ y(N-1) & \cdots & y(N-n) & u(N) & \cdots & u(N-m) \end{pmatrix} \quad \boldsymbol{y} = \begin{pmatrix} y(n+1) \\ y(n+2) \\ \vdots \\ y(N) \end{pmatrix} \tag{5.6}$$

5.2.2 Modèle-hypothèse E/S avec bruit de sortie

Ce modèle (cf. figure 5.4) postule l'existence d'une fonction ξ telle que :

$$y(k) = \xi(y(k-1) - b_s(k-1), ..., y(k-n) - b_s(k-n), u(k), ..., u(k-m)) + b_s(k) \tag{5.7}$$

avec $b_s(k)$ un bruit de sortie additif supposé blanc gaussien.

Le prédicteur théorique associé est :

$$\widehat{y}(k) = g(\widehat{y}(k-1), ..., \widehat{y}(k-n), u(k), ..., u(k-m)) \tag{5.8}$$

Ce prédicteur est « bouclé ». Dans le cas linéaire (modèle OE), g s'écrit :

$$\widehat{y}(k) = -f_1 \widehat{y}(k-1) - ... - f_n \widehat{y}(k-n) + b_0 u(k) + ... + b_m u(k-m) \tag{5.9}$$

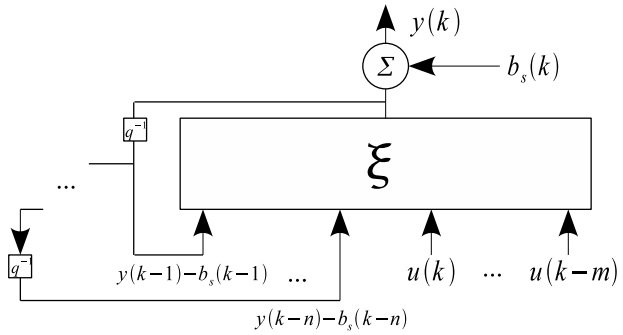


FIGURE 5.4 – Modèle-hypothèse E/S avec bruit de sortie

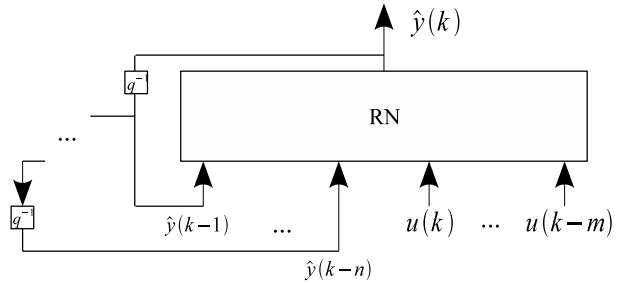


FIGURE 5.5 – Prédicteur neuronal associé au modèle-hypothèse E/S avec bruit de sortie

Pour des conditions initiales différentes, si le prédicteur est stable, l'erreur tend asymptotiquement vers le bruit.

Dans le cas d'un prédicteur neuronal, g est réalisée par un réseau de neurones *bouclé* prenant en entrée le vecteur $\boldsymbol{x} = \big(\widehat{y}(k-1) \quad \cdots \quad \widehat{y}(k-n) \quad u(k) \quad \cdots \quad u(k-m)\big)$ (cf. figure 5.5). Comme le réseau est bouclé, le calcul du gradient est réalisé par rétropropagation à travers le temps (backpropagation through-time) [14]. On dit que l'apprentissage est *non dirigé* par opposition à l'apprentissage d'un prédicteur non bouclé où l'apprentissage est dirigé.

On utilise un réseau d'apprentissage réalisé par N « copies » du réseau initial (cf. figure 5.6). On note w_{ij}^k la copie k du poids w_{ij}. Toutes les répliques w_{ij}^k ont la même valeur w_{ij}, mais pas la même action sur le coût J. On a :

$$\frac{\partial J}{\partial w_{ij}} = \sum_{k=1}^{N} \frac{\partial J}{\partial w_{ij}^k} \tag{5.10}$$

Comme w_{ij} n'intervient que dans le calcul du potentiel du neurone j, on a :

$$\frac{\partial J}{\partial w_{ij}^k} = \frac{\partial J}{\partial a_j(k)} \frac{\partial a_j(k)}{\partial w_{ij}^k} = \frac{\partial J}{\partial a_j(k)} s_i(k) \tag{5.11}$$

où $a_j(k)$ est le potentiel du neurone j à l'instant k et $s_i(k)$ la sortie du neurone i à l'instant k.

Si j est un neurone caché, on utilise l'équation de rétropropagation classique (cf. chapitre 3) :

$$\frac{\partial J}{\partial a_j(k)} = f_j'(a_j(k)) \sum_{m \in \mathrm{succ}(j)} \frac{\partial J}{\partial a_m(k)} w_{jm} \tag{5.12}$$

Si j est un neurone de sortie, $a_j(k)$ influence directement J^k et indirectement les J^l pour

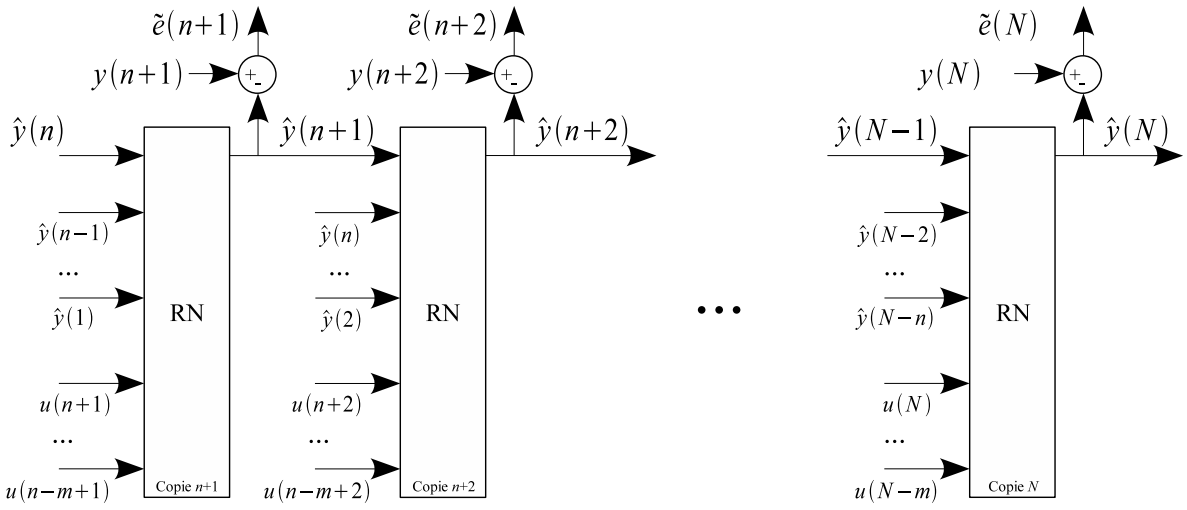


FIGURE 5.6 – Réseau d'apprentissage réalisé par $N-n-1$ « copies » du réseau initial (modèle-hypothèse E/S avec bruit de sortie)

$l > k$, on a donc :

$$\frac{\partial J}{\partial a_j(k)} = \frac{\partial J^k}{\partial a_j(k)} + \sum_{l=k+1}^{N} \frac{\partial J^l}{\partial a_j(k)} \tag{5.13}$$

Le premier terme se calcule directement :

$$\frac{\partial J^k}{\partial a_j(k)} = \frac{1}{N} f_j'(a_j(k))(f_j(a_j(k)) - y_j(k)) \tag{5.14}$$

Pour le deuxième terme, on remarque que tous les J^l pour $l > k$ ne dépendent de $a_j(k)$ que par l'intermédiaire des potentiels des neurones recevant la sortie du neurone j de la copie k, c'est-à-dire tous les neurones de la première couche des copies $k+1$ à $k+n$, on peut donc écrire :

$$\begin{aligned}\sum_{l=k+1}^{N} \frac{\partial J^l}{\partial a_j(k)} &= \sum_{l=k+1}^{N} \sum_{p=1}^{n} \sum_{\substack{m \in \text{succ}(i) \\ i=\text{succ}(j,k+p)}} \frac{\partial J^l}{\partial a_m(k+p)} \frac{\partial a_m(k+p)}{\partial a_j(k)} \\ &= \sum_{l=k+1}^{N} \sum_{p=1}^{n} \sum_{\substack{m \in \text{succ}(i) \\ i=\text{succ}(j,k+p)}} \frac{\partial J^l}{\partial a_m(k+p)} w_{im} f_j'(a_j(k)) \end{aligned} \tag{5.15}$$

avec $\text{succ}(j, k+p)$ l'indice de l'entrée de la copie $k+p$ recevant la sortie du neurone j de la copie k.

Comme $\frac{\partial J^l}{\partial a_m(k+p)} = 0$ pour $l \leq k+p$, l'équation précédente peut s'écrire :

$$\begin{aligned}\sum_{l=k+1}^{N} \frac{\partial J^l}{\partial a_j(k)} &= \sum_{l=n+1}^{N} \sum_{p=1}^{n} \sum_{\substack{m \in \text{succ}(i) \\ i=\text{succ}(j,k+p)}} \frac{\partial J^l}{\partial a_m(k+p)} w_{im} f_j'(a_j(k)) \\ &= f_j'(a_j(k)) \sum_{p=1}^{n} \sum_{\substack{m \in \text{succ}(i) \\ i=\text{succ}(j,k+p)}} \frac{\partial J}{\partial a_m(k+p)} w_{im} \end{aligned} \tag{5.16}$$

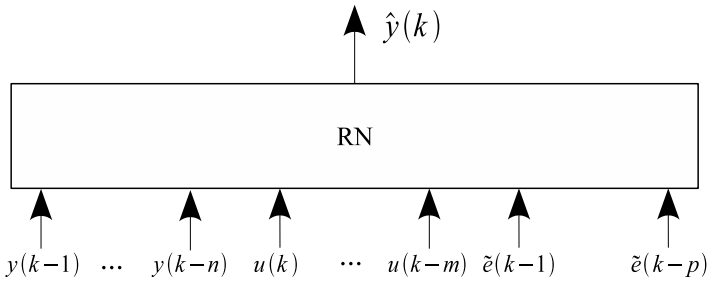


FIGURE 5.7 – Prédicteur neuronal associé au modèle-hypothèse E/S avec bruit de sortie et de boucle

En résumé, si j est un neurone de sortie, on a :

$$\frac{\partial J}{\partial a_j(k)} = \frac{1}{N} f_j'(a_j(k))(f_j(a_j(k)) - y_j(k)) + f_j'(a_j(k)) \sum_{p=1}^{n} \sum_{\substack{m \in \text{succ}(i) \\ i = \text{succ}(j,k+p)}} \frac{\partial J}{\partial a_m(k+p)} w_{im} \quad (5.17)$$

Résumé du calcul du gradient par rétropropagation à travers le temps :

1. calcul du potentiel et de la sortie de tous les neurones pour tous les instants k de 1 à N (sens direct),
2. calcul des dérivées partielles relatives à l'activation des neurones pour tous les instants k de N à 1 (sens rétrograde),
3. calcul du gradient du coût pour toutes les copies de tous les poids,
4. calcul du gradient du coût pour tous les poids.

5.2.3 Modèle-hypothèse E/S avec bruit de sortie et de boucle

Ce modèle postule l'existence d'une fonction ξ telle que :

$$y(k) = \xi(y(k-1), ..., y(k-n), u(k), ..., u(k-m), b_s(k-1), ..., b_s(k-p)) + b_b(k) \quad (5.18)$$

Le prédicteur théorique associé est :

$$\widehat{y}(k) = g(y(k-1), ..., y(k-n), u(k), ..., u(k-m), \tilde{e}(k-1), ..., \tilde{e}(k-p)) \quad (5.19)$$

avec : $\tilde{e}(k) = y(k) - \widehat{y}(k)$.

Ce prédicteur est « bouclé ». Dans le cas linéaire (modèle ARMAX), g s'écrit :

$$\widehat{y}(k) = -a_1 y(k-1) - ... - a_n y(k-n) + b_0 u(k) + ... + b_m u(k-m) + c_1 \tilde{e}(k-1) + ... + c_p \tilde{e}(k-p)) \quad (5.20)$$

Comme pour le prédicteur associé au modèle avec bruit de sortie, pour des conditions initiales différentes, si le prédicteur est stable, l'erreur tend asymptotiquement vers le bruit.

Dans le cas d'un prédicteur neuronal (cf. figure 5.7), g est réalisée par un réseau de neurones *bouclé* prenant en entrée le vecteur :

$$\boldsymbol{x} = \begin{pmatrix} \widehat{y}(k-1) & \cdots & \widehat{y}(k-n) & u(k) & \cdots & u(k-m) & \tilde{e}(k-1) & \cdots & \tilde{e}(k-p) \end{pmatrix} \quad (5.21)$$

Le calcul de gradient suit une procédure analogue à celle décrite à la section précédente.

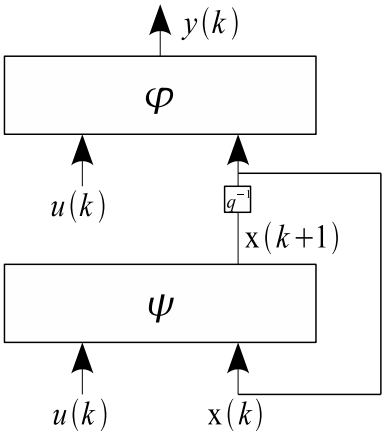


FIGURE 5.8 – Modèle-hypothèse d'état non bruité

5.3 Modèles-hypothèses d'état et prédicteurs associés

5.3.1 Modèle-hypothèse d'état non bruité

Ce modèle hypothèse a peu d'intérêt dans la pratique mais permet de d'expliquer simplement le calcul du gradient par rétropropagation à travers le temps sur un modèle d'état.

Ce modèle postule l'existence des fonctions ψ et φ telles que :

$$\begin{cases} \mathbf{x}(k+1) = \psi(\mathbf{x}(k), u(k)) \\ y(k) = \varphi(\mathbf{x}(k), u(k)) \end{cases} \tag{5.22}$$

avec :

- $y(k)$ la sortie,
- $\boldsymbol{x}(k) \in \mathbb{R}^n$ le vecteur d'état,
- n l'ordre du modèle,
- $u(k)$ la commande.

Le prédicteur théorique associé est :

$$\begin{cases} \widehat{\mathbf{x}}(k+1) = h(\widehat{\mathbf{x}}(k), u(k)) \\ \widehat{y}(k) = g(\widehat{\mathbf{x}}(k), u(k)) \end{cases} \tag{5.23}$$

Ce prédicteur est naturellement « bouclé ». Dans le cas linéaire, on utilise un prédicteur de la forme :

$$\begin{cases} \widehat{x}(k) = A\widehat{\mathbf{x}}(k-1) + bu(k-1) \\ \widehat{y}(k) = C\widehat{\mathbf{x}}(k) \end{cases} \tag{5.24}$$

L'état estimé $\widehat{\mathbf{x}}$ n'a aucune raison d'être une estimation de l'état réel du processus.

Dans le cas d'un prédicteur neuronal, h et g sont réalisés par un seul réseau de neurones bouclé (cf. figure 5.9). Comme le réseau est bouclé, le calcul du gradient est réalisé par rétropropagation à travers le temps (backpropagation through-time) [14].

On utilise un réseau d'apprentissage réalisé par N « copies » du réseau initial (cf. figure 5.10). On note w_{ij}^k la copie k du poids w_{ij}. Toutes les répliques w_{ij}^k ont la même valeur w_{ij}, mais pas la même action sur le coût J. On a :

$$\frac{\partial J}{\partial w_{ij}} = \sum_{k=1}^{N} \frac{\partial J}{\partial w_{ij}^k} \tag{5.25}$$

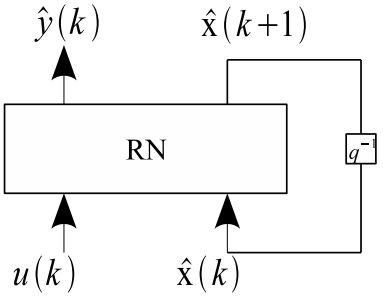


FIGURE 5.9 – Prédicteur neuronal associé au modèle-hypothèse d'état non bruité

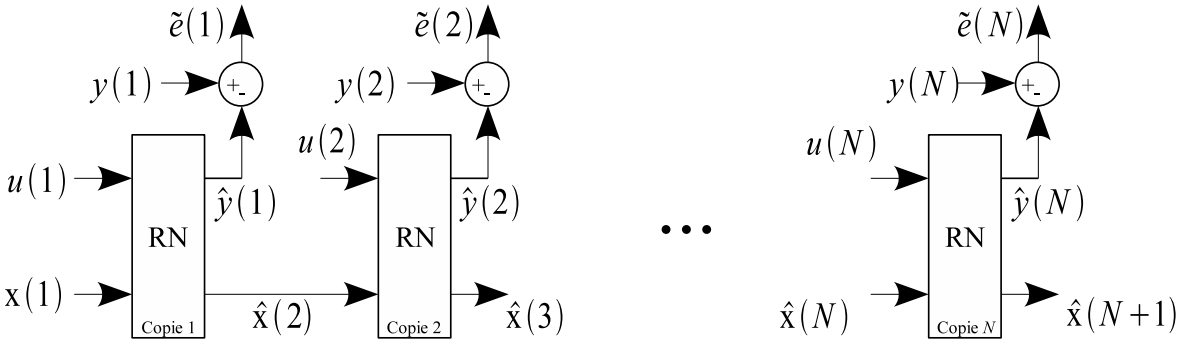


FIGURE 5.10 – Réseau d'apprentissage réalisé par N « copies » du réseau initial (modèle-hypothèse d'état non bruité)

Comme w_{ij} n'intervient que dans le calcul du potentiel du neurone j, on a :

$$\frac{\partial J}{\partial w_{ij}^k} = \frac{\partial J}{\partial a_j(k)} \frac{\partial a_j(k)}{\partial w_{ij}^k} = \frac{\partial J}{\partial a_j(k)} s_i(k) \tag{5.26}$$

où $a_j(k)$ est le potentiel du neurone j à l'instant k et $s_i(k)$ la sortie du neurone i à l'instant k.

Si j est un neurone caché, on utilise l'équation de rétropropagation classique (cf. chapitre 3) :

$$\frac{\partial J}{\partial a_j(k)} = f_j'(a_j(k)) \sum_{m \in \text{succ}(j)} \frac{\partial J}{\partial a_m(k)} w_{jm} \tag{5.27}$$

Si j est un neurone de sortie :

$$\frac{\partial J}{\partial a_j(k)} = \frac{1}{N} f_j'(a_j(k))(f_j(a_j(k)) - y_j(k)) \tag{5.28}$$

Si j est un neurone de sortie d'état et $k = N$:

$$\frac{\partial J}{\partial a_j(k)} = 0 \tag{5.29}$$

Si j est un neurone de sortie d'état et $k < N$, J ne dépend de $a_j(k)$ que par l'intermédiaire des potentiels des neurones recevant la sortie du neurone j de la copie k, c'est-à-dire tous les

neurones de la première couche de la copie $k+1$, on peut donc écrire :

$$\begin{aligned}\frac{\partial J}{\partial a_j(k)} &= \sum_{\substack{m\in\text{succ}(i)\\ i=\text{succ}(j,k+1)}} \frac{\partial J^k}{\partial a_m(k+1)}\frac{\partial a_m(k+1)}{\partial a_j(k)} \\ &= f_j'(a_j(k)) \sum_{\substack{m\in\text{succ}(i)\\ i=\text{succ}(j,k+1)}} \frac{\partial J}{\partial a_m(k+1)} w_{im} \end{aligned} \tag{5.30}$$

avec succ$(j,k+1)$ l'indice de l'entrée de la copie $k+1$ recevant la sortie du neurone j de la copie k.

Résumé du calcul du gradient par rétropropagation à travers le temps :

1. calcul du potentiel et de la sortie de tous les neurones pour tous les instants k de 1 à N (sens direct),
2. calcul des dérivées partielles relatives à l'activation des neurones pour tous les instants k de N à 1 (sens rétrograde),
3. calcul du gradient du coût pour toutes les copies de tous les poids,
4. calcul du gradient du coût pour tous les poids.

5.3.2 Modèle-hypothèse d'état sous la forme d'innovation

Ce modèle postule l'existence des fonctions ψ et φ telles que :

$$\begin{cases}\boldsymbol{x}(k+1) = \psi(\boldsymbol{x}(k), u(k)) + \boldsymbol{b}_b(k) \\ y(k) = \varphi(\boldsymbol{x}(k), u(k)) + b_s(k)\end{cases} \tag{5.31}$$

avec

– $\boldsymbol{b}_b(k) \in \mathbb{R}^n$ un bruit de boucle additif supposé blanc gaussien,
– $b_s(k)$ un bruit de sortie additif supposé blanc gaussien.

Le prédicteur théorique associé est :

$$\begin{cases}\widehat{\boldsymbol{x}}(k+1) = h(\widehat{\boldsymbol{x}}(k), u(k), \tilde{e}(k-1)) \\ \widehat{y}(k) = g(\widehat{\boldsymbol{x}}(k), u(k))\end{cases} \tag{5.32}$$

avec $\tilde{e}(k) = y(k) - \widehat{y}(k)$.

Ce prédicteur est « bouclé ». Dans le cas linéaire, on utilise un prédicteur de la forme :

$$\begin{cases}\widehat{x}(k) = A\widehat{\boldsymbol{x}}(k-1) + Bu(k-1) + K\tilde{e}(k-1) \\ \widehat{y}(k) = C\widehat{\boldsymbol{x}}(k)\end{cases} \tag{5.33}$$

Ce prédicteur est connu sous le nom prédicteur de Kalman.

Dans le cas neuronal, h et g sont réalisés par un réseau bouclé (cf. figure 5.11). Le calcul de gradient suit une procédure analogue à celles décrites précédemment.

5.4 Méthodologie de conception d'un modèle dynamique neuronal

5.4.1 Recueil et traitement des données

– choix une période d'échantillonnage T_e adaptée au système,
– vérification et exploration des non linéarités du système (principe de superposition, principe d'homogénéité, réponse fréquentielle, etc.),

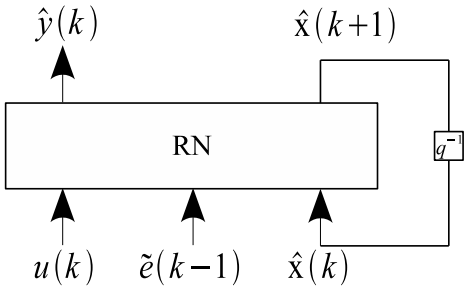


FIGURE 5.11 – Prédicteur neuronal associé au modèle-hypothèse d'état sous la forme d'innovation

- excitation du système dans toutes les combinaisons d'amplitude et de fréquence, avec par exemple :
  - m échelons d'amplitudes aléatoires :

$$u(k) = e\left(\left\lfloor \frac{k-1}{m} \right\rfloor + 1\right) \tag{5.34}$$

    avec $e(k)$ un bruit blanc centré de variance σ_e^2.
  - des échelons d'amplitudes aléatoires à des instants aléatoires :

$$u(k) = \begin{cases} u(k-1) & \text{avec une probabilité } \alpha \\ e(k) & \text{avec une probabilité } 1-\alpha \end{cases} \tag{5.35}$$

  - un vobulateur :

$$u(k) = u_0 + A\sin(\omega(k)kT_e) \tag{5.36}$$

    avec $\omega(k) = \omega_{min} + \frac{k}{N}(\omega_{max} - \omega_{min})$.
- filtrage des signaux inintéressants :
  - pour les hautes fréquences, on ajoute un filtre analogique,
  - pour les basses fréquences, on peux utiliser un filtre numérique.
- suppression des informations redondantes,
- normalisation et centrage des signaux (moyenne de 0 et variance de 1).

5.4.2 Élaboration du modèle

- choix du modèle hypothèse,
- choix des ordres du modèle (sorties et commandes),
- choix du réseau et du nombre de neurones cachés,
- choix de l'algorithme d'apprentissage (second ordre),
- réalisation de plusieurs apprentissages pour éviter les minima locaux.

5.4.3 Validation et sélection de modèles

– recherche de corrélations (linéaires !) :

$$\rho_{\tilde{e}\tilde{e}} = \frac{\sum_{k=1}^{N-\tau} \left(\tilde{e}(k) - \bar{\tilde{e}}\right)\left(\tilde{e}(k+\tau) - \bar{\tilde{e}}\right)}{\sum_{k=1}^{N-\tau} \left(\tilde{e}(k) - \bar{\tilde{e}}\right)^2} \tag{5.37}$$

$$\rho_{u\tilde{e}} = \frac{\sum_{k=1}^{N-\tau} \left(\tilde{u}(k) - \bar{u}\right)\left(\tilde{e}(k+\tau) - \bar{\tilde{e}}\right)}{\sqrt{\sum_{k=1}^{N-\tau} \left(u(k) - \bar{u}\right)^2 \sum_{k=1}^{N-\tau} \left(\tilde{e}(k) - \bar{\tilde{e}}\right)^2}} \tag{5.38}$$

– test classique : on vérifie que 95% des valeurs de ρ sont dans l'intervalle $[-1.96/\sqrt{N}; -1.96/\sqrt{N}]$ pour tout $\tau \in [-20; 20]$.
– calcul de l'erreur quadratique moyenne sur un autre ensemble de mesures entrées/sorties,
– visualisation des prédictions :
  – prédiction à 1 pas,
  – prédiction à n pas,
  – en simulation ($n = \infty$).
– comparaison avec le prédicteur naïf $\widehat{y}(k) = y(k-1)$ (si la période d'échantillonnage est beaucoup plus faible par rapport au temps de réponse du système),
– élagage.

5.4.4 Conclusion

La modélisation de systèmes dynamiques par réseaux de neurones présente les mêmes avantages et inconvénients que la modélisation statique (modélisation de tout types de processus non linéaires, problèmes de minima locaux et du temps de calcul). Un inconvénient supplémentaire réside dans le recueil des données : il est toujours difficile d'exciter un système dynamique non linéaires dans tout ses régimes de fonctionnement.

Annexe A

Glossaire des notations

a_j	potentiel (ou activation) du neurone j
f_j	fonction d'activation du neurone j
g	modèle
h	modèle
$J(\boldsymbol{w})$	fonction de coût totale (critère quadratique)
$J_k(\boldsymbol{w})$	fonction de coût partielle (critère quadratique)
N	nombre de mesures disponibles pour l'apprentissage (hors ligne)
n_c	nombre de neurones cachés d'un réseau de neurone
n_e	nombre de neurones d'entrée d'un réseau de neurone
n_s	nombre de neurones de sortie d'un réseau de neurone
pred(j)	ensemble des indices des neurones dont la sortie est connectée à une entrée du neurone j
succ(j)	ensemble des indices des neurones recevant la sortie du neurone j
succ($j, k+p$)	indice de l'entrée de la copie $k+p$ recevant la sortie du neurone j de la copie k
$\boldsymbol{u}$	commande d'un système (vecteur)
$\boldsymbol{w}$	paramètres d'un modèle (vecteur)
w_{ij}	poids de la connexion du neurone i au neurone j
w_{0i}	seuil (ou biais) du neurone i
$\boldsymbol{x}$	grandeur d'entrée (vecteur) ou état d'un système
y	grandeur de sortie (scalaire)

Annexe B

Sélections de logiciels

Pour Matlab :
- (*neural networks toolbox* de Mathworks)
- *Netlab* du Neural Computing Research Group Information Engineering de l'université d'Aston (bibliothèque libre)
- *NNSYSID toolbox* de Magnus Nørgaard de l'université technique du Danemark (bibliothèque libre)

En langage C/C++ :
- *Fast Artificial Neural Network Library (FANN)* de Steffen Nissen et Evan Nemerson du département informatique de l'université de Copenhague (bibliothèque libre)
- *Neural Network Torch Libary* de Ronan Collobert, Samy Bengio et Johnny Mariéthoz de l'institut de recherche IDIAP (bibliothèque libre)

Bibliographie

[1] A. Barron. Universal approximation bounds for superposition of a sigmoidal function. *IEEE Transactions on Information Theory*, 39 :930–945, 1993. 10

[2] C.M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995. 19

[3] Y. Le Cun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks : Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*. Springer, 1998. this book is an outgrowth of a 1996 NIPS workshop. 14

[4] G. Dreyfus, J.-M. Martinez, M. Samuelides, M. B. Gordon, F. Badran, S. Thiria, and L. Hérault. *Réseaux de neurones : méthodologie et applications*. Eyrolles, 2004. ISBN 2-212-11464-8. 5

[5] E. Ho and R. L. Kashyap. An algorithm for linear inequalities and its applications. *IEE Transactions on Electronic Computers*, 14 :683–688, 1965. 20

[6] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2 :359–366, 1989. 9

[7] A. U. Levin and K. S. Narendra. Control of nonlinear dynamical systems using neural networks : Controllability and stabilization. *IEEE Trans. on Neural Networks*, 4(2) :1011–1020, 1993. 26

[8] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, and S. Marcos. Neural networks and non-linear adaptive filtering : unifying concepts and new algorithms. *Neural Computation*, 5 :165–197, 1993. 9, 13

[9] M. Nørgaard, O. Ravn, N. K. Poulsen, and L. K. Hansen. *Neural Networks for Modelling and Control of Dynamic Systems : A Practitioner's Handbook*. Springer, 2000. 5

[10] Léon Personnaz and Isabelle Rivals. *Réseaux de neurones formels pour la modélisation, la commande et la classification*. CNRS Editions, 2003. ISBN 2-271-06103-2. 5

[11] David Price. *Classification probabiliste par réseaux de neurones ; application à la reconnaissance de l'écriture manuscrite*. PhD thesis, Université Pierre et Marie Curie, Paris VI, 1996. 23

[12] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning : The RPROP algorithm. In *Proc. of the IEEE Int. Conf. on Neural Networks*, pages 586–591, San Francisco, CA, 1993. 12, 13

[13] F. Rosenblatt. *Principles of neurodynamics : Perceptrons and the theory of brain mechanisms*. Spartan, 1962. 18

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error back-propagation. In D. E. Rumelhart, J. L. McLelland, and the PDP Research Group, editors, *Parallel Distributed Processing : explorations in the microstructure of cognition. Vol.1 : Foundations*, pages 318–362. MIT Press, Cambridge MA, 1986. 28, 31

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986. 13

[16] E. D. Sontag. Neural networks for control. In H. L. Trentelman and J. C. Willems, editors, *Essays on control : perspectives in the theory and its applications*, pages 339–380, Boston, 1993. 9

[17] B. Widrow and M. E. Hoff. Adaptive switching circuits. *IRE Wescon Convention Record*, 4 :96–104, 1960. Reprinted in Anderson & Rosenfeld (eds), Neurocomputing : foundations of research, MIT Press, 1988. 18