UNIVERSITÉ DU QUÉBEC À MONTRÉAL

CONTRIBUTION À LA PRÉDICTION DES PERTES DE PUISSANCE SUR UN RÉSEAU ÉLECTRIQUE PAR UN MODÈLE À BASE DE RÉSEAU DE NEURONES

MÉMOIRE PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
CLAUDIA MAGUITO LONTCHI

OCTOBRE 2021

REMERCIEMENTS

Je remercie tous ceux et celles qui de près ou de loin m'ont apporté leurs soutiens matériels et immatériels afin que je puisse acquérir l'aptitude pour ce qui est de la maîtrise en génie électrique. Je voudrais dans ce sens remercier tous mes enseignants du département de génie électrique de l'UQAM.

Je tiens particulièrement à remercier le professeur Mounir Boukadoum qui a bien voulu accepter de diriger ce travail, m'initiant ainsi à ce vaste domaine scientifique en plein expansion qu'est l'IA. Je ne saurais oublier vos précieux conseils dont j'ai bénéficié avant et pendant la rédaction de ce mémoire.

À vous également les membres de ma famille ainsi qu'à mes amis, recevez ici toute ma gratitude vis-à-vis du soutien moral et matériel dont vous avez toujours su faire montre en tout temps et en tous lieux.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

Tableau Page

LISTE DES FIGURES

Figure Page

LISTE DES ABRÉVIATIONS, DES SIGLES ET DES ACRONYMES

ACP :	Auto Correlation Partielle
ANN :	*Artificial Neural Network*
Adam :	*Adaptative Moments*
AdaGrad :	*Adaptative Gradient*
CEI :	Commission Électrotechnique Internationale
CNN:	*Convolutional Neural Network*
CPU:	*Central Processing Unit*
GRU:	*Gated Recurrents Units*
IA :	Intelligence Artificielle
IACM :	Interrupteur Aérien à Commande Manuelle
IHM:	Interface Homme Machine
K-NN:	*K- Nearest Neighbors*
LSTM:	*Long Short Term Memory*
MAE:	*Mean Absolute Error*
MAJ:	Mise à Jour
MAS:	Machine Asynchrone
MLP:	*Multi-Layer Perceptron*
MSE:	*Mean Square Error*
PMC :	Perceptron Multicouches
PP :	Pertes de Puissances
PV:	Photo Voltaïque
RMSE:	*Root Mean Square Error*
RMSProp:	*Root Mean Square Error Propagation*
RNA :	Réseaux de Neurones Artificiels
RNN :	*Recurrent Neural Network*

RPG : Retro Propagation du Gradient

SVM : Support Vecteur Machine

RESUMÉ

La présente recherche a pour objet de proposer des modèles empiriques de prédiction des pertes de puissances dans un réseau électrique. Ces modèles seront obtenus à base d'un RNA de type LSTM. Dans ce sens, 3 algorithmes d'apprentissage dédiés ont été séparément simulés et comparés. Signalons à ce propos que pour chaque algorithme utilisé, plusieurs configurations de ladite architecture ont été testées.

La base de données que nous avons utilisée pour effectuer l'entrainement d'un modèle est issue des rapports dressant les perturbations et les occurrences inhabituelles survenues sur le réseau électrique Nord-Américain. À cet effet, l'on y retrouve l'historique des pertes de puissance relatives à ces perturbations. Cet historique qui couvre une période de 20 ans possède néanmoins des « trous » qui illustrent les difficultés inhérentes à la collecte des données de terrain.

Nous avons structuré notre travail en trois chapitres. Le premier chapitre consacré à l'étude des outils d'analyse de la fiabilité des systèmes permet de mettre en exergue leurs incapacités à prédire temporellement les perturbations. Cette incapacité justifie l'utilisation d'un prédicteur neuronal. Le second chapitre dédié à l'analyse des architectures neuronales nous permet de justifier le choix du réseau LSTM. Le troisième chapitre est axé sur la présentation et l'interprétation des résultats de simulation.

Les résultats obtenus avec ces modèles de prédiction de type '*forecasting*' nous permettent de constater que les LSTM ont une très grande puissance de représentation des séries temporelles lorsqu'ils sont utilisés avec une base de données de très grande taille. Toutefois, nous pensons qu'il est possible d'obtenir des résultats similaires avec d'autres architectures neuronales profondes.

Mots clés : Fiabilité ; Prédiction ; série temporelle ; Architecture neuronale profonde ; Performance

ABSTRACT

The purpose of this research is to propose empirical models for predicting powered losses in an electrical network. These models will be obtained based on an ANN type LSTM. Note that for each algorithm used, several configurations of said architecture have been tested.

The database we used to perform model training is reported from the disruption and unusual occurrences occurring on the North American network. For this purpose, there is a history of the losses of power relating to these disturbances. This historic covers a 20-year, neither holes that illustrate the difficulties inherent in the collection of field data.

We structured our work in three chapters. The first is dedicated to the study of system's reliability analysis tools makes it possible to highlight their disabilities temporarily predict disruption. This inability justifies the use of a neuronal predictor. The second chapter dedicated to the analysis of neural architectures allow us to justify the choice of the LSTM network. The third chapter is focused on the presentation and interpretation of simulation results.

The results obtained with these forecasting prediction models allow us to note that the LSTMs have a very high power of representation of time series when used with a large size database. However, we think it is possible to obtain similar results with other deep neural architectures.

Keywords: Reliability; Prediction; Time series; Deep artificial neural architecture; Performance

INTRODUCTION

Problématique

La disponibilité de l'énergie électrique est une nécessité absolue pour un développement harmonieux des divers secteurs (télécommunications, industries, hôpitaux, réseaux de transport, …) de notre société moderne du 21ème siècle. Les grandes distances qui séparent parfois les centres de productions d'énergie électrique des lieux de consommation de ladite énergie justifient l'existence des réseaux de transport et de distribution de type aérien ou souterrain. Ceci peut s'observer dans la province de Québec où les lignes de transport peuvent s'étendre sur des milliers de kilomètres avant d'atteindre les lieux de consommation dans le Sud. Ces réseaux de transport d'énergie comprennent des composantes (câbles, transformateurs, IACM, parafoudres, chaines d'isolateurs, …) multiformes. Ces composantes peuvent être soumises à des pannes d'origines techniques, humaines et naturelles. Dans les pays nordiques comme le Canada, le dépôt excessif de glace atmosphérique sur les équipements de transport d'énergie électrique constitue une part importante des pannes d'origine naturelle. Quand ce dépôt de glace atmosphérique entraine une baisse considérable de la tension de tenue, les isolateurs de la ligne se comportent comme des conducteurs électriques (Farzaneh et *al.,* 2000). Les évolutions technologiques contemporaines dans le domaine de la sécurité des installations électriques et dans la mise en application des méthodes d'amélioration de la fiabilité des systèmes permettent d'éliminer les pannes d'origines humaines et techniques. Cependant les pannes d'origines naturelles demeurent hors de portée, ce qui nous permet d'affirmer que les pannes et leurs conséquences auront toujours un impact négatif sur le fonctionnement du réseau.

L'existence des pannes d'origines diverses étant donc inévitable, il est utile au gestionnaire ou au propriétaire du réseau d'avoir la maîtrise de leurs occurrences. Cette

maîtrise s'avère toujours utile pour la gestion efficace de la maintenance préventive et corrective du réseau électrique. Dans le contexte du transport et de la distribution d'énergie électrique, cela pose la problématique de prédiction des pannes dans le temps. Ces réseaux de transport et de distribution d'énergie électrique présentent des topologies différentes en raison de la technologie de conception utilisée, de la distance qu'ils couvrent, des normes en vigueur dans une zone géographique, ... L'impact de l'occurrence des pannes sur la disponibilité de l'énergie électrique varie d'un réseau à l'autre en fonction des variables précédentes. Au regard des spécificités locales, cette variation peut relativiser la nécessité d'effectuer la prédiction des pannes. Cette problématique trouve toute sa raison d'être dans un grand réseau tel que le réseau Nord-Américain pour lequel les enjeux liés à la disponibilité de l'énergie électrique sont énormes en tout temps et en tout lieu.

Enjeux de la prédiction des pannes

Les consommateurs et les distributeurs d'énergie électrique éprouvent de plus en plus des difficultés à faire face aux défis liés à l'indisponibilité temporaire de cette énergie du fait des pannes inévitables qui surviennent sur le réseau. Cela étant, la prédiction des pannes devient donc un enjeu majeur pour le distributeur, les consommateurs d'énergie et les scientifiques.

Pour le consommateur

L'incertitude relative aux périodes d'indisponibilité de l'énergie électrique peut contraindre le consommateur à se tourner vers des appareillages électriques possédant une certaine autonomie énergétique pour assurer la continuité de service en l'absence de l'énergie du réseau classique. En effet, dans les zones géographiques où les délestages sont fréquents, le besoin de s'alimenter via des sources d'énergies alternatives est de plus en plus présent chez certains consommateurs. Cette réalité

justifie partiellement la croissance dans les achats d'installations photovoltaïques (Leng et coll., 1996, et Maycock, 2000). Certains consommateurs ou opérateurs économiques choisissent leurs lieux d'établissement en fonction de la disponibilité temporelle de l'énergie électrique. Cette prédiction peut également permettre au consommateur d'ajuster sa facture énergétique dans le cas des options prépayées par exemple.

Pour le gestionnaire du réseau

Le gestionnaire du réseau électrique n'est pas toujours tenu de garantir la fourniture d'énergie sous tension et fréquence nominale ainsi qu'une continuité de service par exemple. Ces faits se justifient en partie par l'existence des pannes sur le réseau. Dès lors, il devient nécessaire d'effectuer une prédiction desdites pannes car elles impactent sur :

- La gestion de la maintenance du réseau ;

En effet, dans le cadre de la maintenance préventive, une bonne prédiction permet de cibler les composantes du réseau qui nécessiteront une observation particulière ainsi que celles qui doivent être remplacées par des équivalents plus performants afin d'éviter la panne ou de minimiser son impact. Selon les résultats de prédiction obtenus, l'on peut se questionner sur la nécessité de poursuivre l'exploitation dudit réseau. Dans le cadre de la maintenance corrective, elle peut contribuer à l'estimation de la charge de travail du personnel durant une période afin d'améliorer la stratégie de déploiement sur le terrain.

- La gestion de ressources financières et matérielles ;

La prédiction des pannes peut impacter positivement sur les allocations financières du service de maintenance, tel que l'approvisionnement en stock pour les équipements de rechange, la simulation des coûts d'intervention, …

- La gestion de la clientèle ou des consommateurs ;

Cette connaissance peut amener le gestionnaire du réseau à faire une communication à l'endroit des consommateurs longtemps avant la survenance de la panne afin que ces derniers prennent les dispositions nécessaires au cas où cette panne engendrerait une coupure d'alimentation.

Pour les scientifiques

Dans le cadre de ce travail, nous disposons d'un tableau récapitulatif des causes et conséquences de l'occurrence des pannes sur le réseau et sur les abonnées. Ces données concernent une période de 14 ans allant de Janvier 2005 à Février 2018. Les informations contenues dans ce tableau peuvent être mises sous forme de données spatiotemporelles. Cette base de données est importante pour les scientifiques car la prédiction des séries temporelles montre un intérêt croissant de nos jours en science des données et notre but est d'y parvenir en utilisant les techniques d'IA.

Quelques approches de prédiction des séries temporelles

En fonction des objectifs et des exigences mathématiques liées à chaque théorie, des chercheurs ont proposés au fil du temps plusieurs modèles de prédiction des séries temporelles.

- Les chaînes de Markov :

Les chaînes de Markov font partie des outils de prédiction des séries temporelles. Une chaîne de Markov se définie comme une suite de variables aléatoires $(X_n, n \in \mathbb{N})$ à valeurs dans un espace d'états noté E qui permet de modéliser l'évolution dynamique d'un système aléatoire. Sa propriété fondamentale encore appelée propriété de Markov stipule que : l'évolution future de la variable aléatoire ne dépend du passé qu'à travers sa valeur actuelle. Il s'agit donc d'analyser la tendance pour un évènement d'être suivi

par d'autres évènements. La probabilité pour un système de passer d'un état actuel *i* vers un état futur *j* est définie par :

$$\mathbb{P}(X_{n+1} = j \mid X_0 = i_0, \dots, X_n = i) = \mathbb{P}(X_{n+1} = j \mid X_n = i) \quad \text{(I)}$$

$$\forall n \geq 1, \forall (i,j) \in \mathrm{E}^2$$

- K plus proches voisins (K-NN) :

Cette méthode issue du domaine de l'IA consiste à repérer dans l'ensemble des données d'apprentissage un groupe formé de K données les plus semblables à celle dont on veut prédire la valeur suivante. La qualité de la prédiction dépend du niveau d'optimisation du paramètre K. (Sharif M et Burn DH, 2006)

- Les modèles autorégressifs

Ils s'appliquent sur des processus invariants à temps discret ce qui n'est pas souvent le cas dans la plupart des séries temporelles ; ce qui impose une transformation de la série initiale en une série stationnaire. (Box et Jenkins, 1976) L'on note également que son utilisation nécessite absolument une maitrise de certains outils statistiques tels que la covariance et l'autocorrélation partielle.

- Les modèles basés sur les réseaux de neurones artificiels

C'est un modèle très intéressant lorsqu'on dispose d'une base de données assez grande sans pour autant savoir quel est le principe physique ou la formulation mathématique qui régit leurs existences. Son principe repose sur l'utilisation des « algorithmes d'apprentissage » dans le but de modifier les paramètres des données présentées au réseau afin d'obtenir en sortie de ce dernier une réponse proche de celle escomptée avec une marge acceptable.

Dans les paragraphes précédents, nous avons présenté une liste non exhaustive des modèles de prédiction des séries temporelles qui existent dans la littérature. À

l'observation des principes de fonctionnement de ces divers modèles, nous avons porté notre choix sur le modèle de prédiction à base de réseau de neurone artificiel pour les raisons suivantes :

La diversité d'outils de développement

Il existe de nos jours plusieurs applications logicielles ou environnement de développement qui facilitent la conception des modèles de prédiction à base de réseaux de neurones artificiels (RNA) tel que : SAS, MATLAB, NeuroOne, Intelligent Miner, Predict, etc.

Niveau de pertinence des données à traiter

Les RNA et plus précisément les RNA profonds font partir des meilleurs outils pour l'analyse des données qu'elles soient bruitées ou incomplètes.

Souplesse

Ils ont la capacité de traiter des problèmes pour lesquels l'on ne dispose d'aucune information au préalable.

Qualité des résultats

Leurs performances ont été démontrées dans l'analyse des processus parfois complexes tel que la classification d'images et ils connaissent de nos jours un essor remarquable au détriment des méthodes traditionnelles telles que les méthodes statistiques ou les arbres de décision. Il est à noter que ces résultats sont obtenus sans que l'on ait besoin de se soucier de la fonction mathématique qui régit la relation entre les sorties et les entrées, car le modèle fonctionne sous forme de boîte noire.

Dans ce mémoire, nous considérons les pertes de puissances sur le réseau comme une forme de manifestation des pannes subies par celui-ci. Nous effectuerons donc des prédictions de perte de puissance sur des horizons à court terme (inférieure ou égale à quatre mois) et long terme (supérieure à quatre mois). Étant donné que l'on fera usage

des méthodes d'apprentissage automatique, nous utiliserons une base de données fournie par Emergency Operations Center Form EIA-417R sur une période allant de Janvier 2005 à Décembre 2018.

Survol du mémoire

Dans ce mémoire, le premier chapitre aborde sommairement la notion de sûreté de fonctionnement des systèmes dans le but de présenter l'approche probabiliste de la fiabilité en précisant son champ d'application. Il présente également quelques outils d'apprentissage automatique dédiés à la prédiction des séries temporelles. Les limites de l'approche probabiliste justifient ainsi l'existence du chapitre 2 qui porte sur une approche de prédiction par réseau de neurones artificiel (RNA) puisqu'étant la mieux adaptée pour la prédiction des cas individuels. Dans ce deuxième chapitre, nous faisons dans un premier temps une présentation des architectures à base de RNA en insistant sur le principe de fonctionnement du Perceptron Multicouches (PMC) dans son architecture classique. À la suite de ce qui précède, nous décrivons le principe de fonctionnement des architectures récentes dites profondes car celles-ci sont plus efficaces que les précédentes. Enfin, nous décrivons l'architecture neuronale choisie pour réaliser notre tâche de prédiction. Nous poursuivons avec un troisième chapitre qui présente la base de données d'entraînement, l'environnement matériel et immatériel de travail ainsi que les résultats obtenus avec les réseaux de neurones profond. Dans ce chapitre, nous présentons également les résultats obtenus avec d'autres outils d'apprentissage automatique dans le but de les confronter avec ceux obtenus par les RNA. À l'issue de cette confrontation, nous retenons un modèle de prédiction sur la base des critères prédéfinis. Nous achevons le mémoire par une conclusion générale qui constitue une synthèse de notre travail tout en proposant quelques pistes d'amélioration.

CHAPITRE I

GÉNÉRALITÉS SUR LA SÛRETÉ DE FONCTIONNEMENT DES SYSTÈMES

De nos jours, la compétitivité des marchés industriels confère aux aspects liés à la fiabilité des produits et des services une grande importance. De même la productivité d'une usine est directement liée au bon fonctionnement de ses machines. On voit aussi apparaitre chez les industriels un besoin grandissant de mesurer ou d'améliorer la fiabilité des produits qu'ils vendent. L'objet du présent chapitre est de présenter le concept de sûreté de fonctionnement des systèmes à travers l'approche probabiliste classique et l'approche empirique basée sur les observations de terrain. Nous présenterons aussi certains outils d'apprentissage automatique dédiés à la prédiction des séries temporelles telles que les arbres de décision, la régression et les supports vecteurs machine (SVM). Cette présentation qui se fera en précisant le contexte d'application de chaque approche a pour but de déduire celle qui est la mieux appropriée pour atteindre notre objectif de prédiction des pannes. Pour réaliser cet objectif, nous présenterons dans la section 1.1 la notion de sûreté de fonctionnement des systèmes, la section 1.2 présentera l'analyse qualitative de la sûreté de fonctionnement, la section 1.3 sera consacrée à l'analyse quantitative de la sûreté de fonctionnement, la section 1.4 présentera brièvement le principe de l'approche empirique, la section 1.5 sera consacrée à la synthèse des deux dernières sections et enfin la section 1.6 sera quant à elle dédiée à la présentation de certains outils d'apprentissage automatique.

1.1 Sûreté de fonctionnement d'un système

Un système est un ensemble d'éléments discrets qui interagissent entre eux en vue de l'accomplissement d'une fonction précise.

1.1.1 Définition du concept de sûreté de fonctionnement

La sûreté de fonctionnement est une science dédiée à l'étude des défaillances en vue de leurs connaissances, leurs évaluations, et leurs prévisions. Ce concept met à disposition des outils visant à analyser et à quantifier l'aptitude d'un système à remplir la fonction pour laquelle il a été conçu.

1.1.2 Les défaillances

Une défaillance est la cessation d'aptitude d'une entité à accomplir une fonction requise. Cette cessation peut être complète (l'entité ne remplit plus sa fonction) ou partielle (la fonction est assurée dans certaines limites).

« Un mode de défaillance est l'effet par laquelle une défaillance est observée » (CEI). Dans le cas d'un disjoncteur magnétothermique par exemple, une défaillance de son mécanisme d'ouverture s'observe par son maintien à l'état fermé tandis qu'une défaillance du capteur de courant s'observe par la non-ouverture du disjoncteur lorsqu'il est sollicité à l'ouverture. Ces diverses défaillances constituent les modes de défaillances du disjoncteur.

1.1.3 Critères d'analyse de la sûreté de fonctionnement

- Fiabilité R(t)

Elle est définie comme la probabilité qu'une entité fonctionne entre les instants 0 et t. Il en découle que la fonction de fiabilité R(t) est décroissante sur [0, 1] tel que $\lim_{t\to\infty} R(t) = 0$.

- Disponibilité A(t)

Elle représente la probabilité qu'une entité puisse fonctionner à un instant t donné.

- Maintenabilité M(t)

Il s'agit de la probabilité pour un composant d'être réparée durant l'instant compris entre 0 et t, contrairement à la fiabilité, sa fonction est croissante sur [0, 1] tel que $\lim_{t\to\infty} M(t) = 1$

- Variables temporelles moyennes

MTTF : durée moyenne de fonctionnement avant la première défaillance

MTTR : durée moyenne de réparation (*Mean Time To Repare*)

MUT : durée moyenne de fonctionnement après réparation (*Mean Up Time*)

MDT : durée moyenne d'indisponibilité (*Mean Down Time*)

MTBF : durée moyenne de bon fonctionnement (*Mean Time Between Failure*)

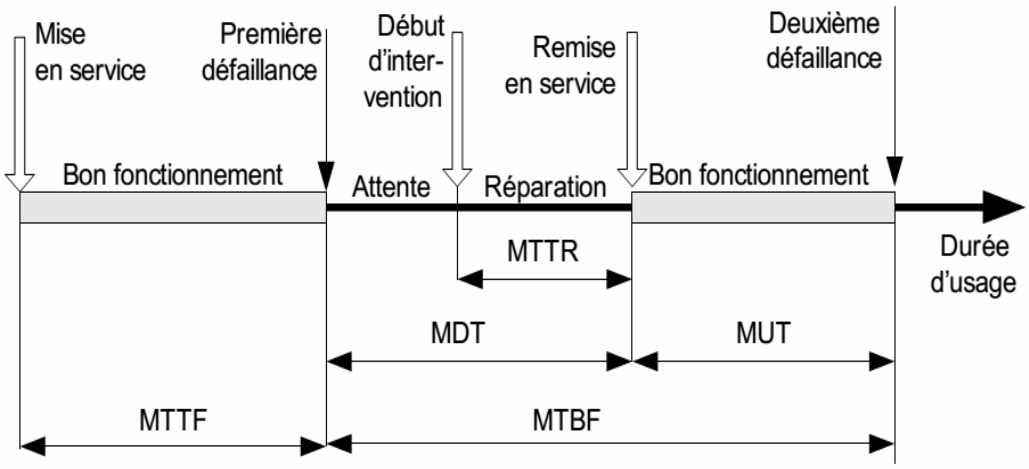

Figure 1.1 : Graphique des temps moyens

- Le taux de défaillance λ(t)

Noté λ(t), est un estimateur de fiabilité. Il représente une proportion de dispositifs survivants à un instant t (Meva'a., 2020). Sa forme générale est Nombre de défaillance/durée d'usage. Il s'exprimera en « pannes/heures » (Meva'a., 2020).

$$\lambda(t) = -\frac{1}{R(t)} . \frac{dR(t)}{dt} \qquad (1.1)$$

Sa représentation graphique pour les composants électriques a l'allure d'une courbe en baignoire.

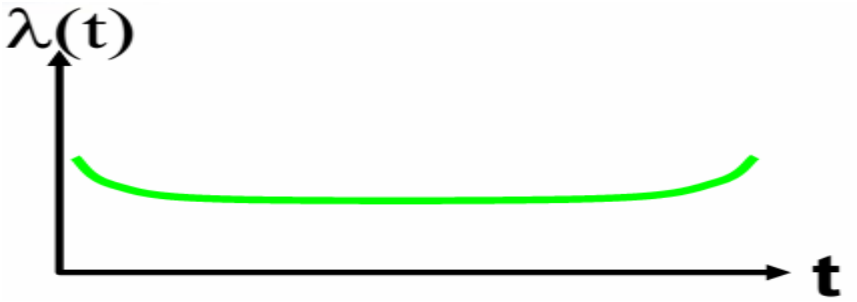

Figure 1.2 : Courbe en baignoire d'un composant électrique

1.2 Analyse qualitative de la sûreté de fonctionnement

Il s'agit ici de l'identification des diverses défaillances d'un système afin d'élaborer une stratégie de modélisation. Cela peut nécessiter quelques prérequis tel qu'une bonne connaissance technique du système mis en étude, la nécessité d'une expérience avérée sur l'exploitation du système en question, une bonne connaissance des outils d'analyse de la sureté de fonctionnement, …

1.2.1 Analyse des défaillances d'un système

L'analyse des défaillances consiste en un recensement des modes de défaillance de chaque composante du système à travers les causes et les effets. L'on élabore les différentes configurations que peut prendre le système vis-à-vis des contraintes imposées par son environnement pouvant induire une interruption des fonctions du composant. Tous les modes de défaillance du composant en question doivent être pris en compte durant cette analyse.

Tableau 1.1 : Analyse des défaillances d'un disjoncteur magnétothermique

MODE DE DÉFAILLANCE	CAUSES	CONSÉQUENCES
Défaut d'isolement	- Mécanisme interne d'isolement des pôles	Court-circuit franc
Détérioration des bornes	- Action humaine - Usure	Circuit ouvert
Déclenchement inopportun	- Disfonctionnement du dispositif de déclenchement - Défaut capteur de courant	Circuit ouvert
Défaut de déclenchement	- Défaut capteur de courant - Disfonctionnement du dispositif de déclenchement	Apparition d'un défaut
Défaut de ré enclenchement	- Défaut du ressort de rappel - Défaut ou disfonctionnement non éliminé - Ligne en court-circuit	Circuit ouvert

1.2.2 Défaillances simultanés de causes communes

L'occurrence simultanée de plusieurs défaillances engendre des conséquences plus néfastes pour le système comparativement à certaines occurrences uniques. Cela peut résulter de la combinaison des facteurs tels que l'environnement (corrosion, mauvaises conditions climatiques, vandalisme, …), les défauts de conception, les défauts de montage et d'exploitation qui ne sont mis en exergue que durant la période d'exploitation.

1.2.3 Défaillance cachée

C'est un type de défaillance qui se détecte à la deuxième apparition au moins. Elle peut persister au-delà de cette deuxième apparition bien que le système soit fonctionnel tout en cumulant des effets. Dans certains cas de figure, son ultime manifestation est catalectique. C'est le cas d'une MAS dont le disjoncteur de protection est surdimensionné. Ce disjoncteur laissera circuler dans la MAS des courants anormalement élevés en cas de surcharge. L'échauffement excessif de la MAS lié au passage de ces courants aura pour effets cumulés, une dégradation progressive de l'isolation du bobinage statorique. L'ultime manifestation de cette défaillance cachée nécessitera une maintenance corrective de plus haut niveau.

1.2.4 Démarche à suivre à la survenance d'une défaillance

A la survenance d'une défaillance, quatre actions sont généralement mises en œuvre pour l'éliminer.

- La détection

Elle peut se faire par une signalisation (IHM ou indicateurs lumineux conçus à cet effet) ou par des techniques appropriées à chaque système. Dans le cas d'un réseau électrique,

cela peut se faire au niveau du poste de contrôle-commande ou par des dispositifs spéciaux de localisation géographique des défauts.

- Le diagnostic

Il vise à déterminer la (les) cause(s) de la panne. Il est effectué par un agent spécialisé en la matière. En fonction du niveau de technologie utilisé dans le système, ce diagnostic peut s'effectuer en s'appuyant sur les indicateurs d'un IHM, sur une simple observation de l'état physique du système (relativement au niveau d'expérience de l'agent) ou sur les prescriptions d'un document constructeur. Dans un réseau électrique, ce diagnostic peut s'effectuer par une équipe pluridisciplinaire (ingénieur électrotechnicien, mécanicien, génie civil, génie thermique, …) selon la nature de la panne tout en faisant usage des techniques et des ressources appropriées.

- La maintenance corrective

Elle intervient après le diagnostic et vise à éliminer la défaillance. Elle s'effectue au moyen d'une technique propre à chaque système.

- La remise en service du système

Elle constitue l'étape de vérité de la maintenance corrective et vise à ramener le système dans son état de fonctionnement d'avant la survenance de la défaillance. Dans le cas de la distribution d'énergie électrique, elle peut se faire globalement ou par zone.

1.2.5 Technique d'amélioration de la fiabilité : la redondance

Un système à structure redondante est celui dans lequel il existe plusieurs moyens techniques indépendants, identiques ou non, qui assurent la même fonction et sont destinés à se substituer les uns aux autres en cas de besoin. Sa configuration est calquée sur le modèle d'un circuit électrique monté en parallèle. L'objectif est d'assurer en permanence la continuité de service en cas de défaillance. Cette technique est très

utilisée dans la réalisation d'une installation photovoltaïque de grande puissance où l'on adopte pour l'onduleur de tension une configuration de type *multy string* afin d'éviter que la défaillance d'un seul onduleur n'entraîne un défaut d'alimentation des charges de type alternative. L'on observe également cette technique dans le réseau électrique pour ce qui concerne les alimentations dites de secours. Il existe deux types de redondance :

- Redondance passive

Seul le premier composant est mis en marche à t = 0. Une fois en panne, le deuxième composant prend le relai et ainsi de suite. Le système au complet tombe en panne quand le n-ième composant tombe en panne (Digabel., 2017). Comme exemple de redondance passive, l'on peut citer le générateur de secours qui est mis en service manuellement ou par inverseur de source automatique en cas de défaillance de la source principale.

- Redondance active

Tous les composants du système fonctionnent dès le temps t = 0. Il suffit qu'au moins un composant fonctionne pour que tout le système que complet fonctionne (S. Diagebel., 2017).

Il ressort de tout ce qui précède que l'analyse qualitative permet de comprendre les causes et les manifestations d'un certain nombre de défaillance. Comme inconvénients, elle est non généralisable et figée sur les idées des observateurs.

1.3 Analyse quantitative de la sûreté de fonctionnement

L'objectif de l'analyse quantitative est de s'appuyer sur les modes de défaillance d'un système afin de modéliser mathématiquement les pannes du système. Cette modélisation permet d'obtenir une valeur quantitative de fiabilité si l'on veut absolument éviter un risque de panne. Elle permet également d'obtenir une valeur

moyenne d'interruptions ainsi que le temps moyen d'indisponibilité si l'on sait d'avance que le système subira des pannes. Lorsque ces valeurs sont obtenues, elles sont comparées à des valeurs dites acceptable. Pour y parvenir, des hypothèses sont émises au départ, des postulats sont parfois utilisés pour adapter les données collectées au modèle mathématique afin d'estimer quantitativement le taux de défaillance ou le taux de réparation. Quand le système comporte des modes de fonctionnement normaux dont les probabilités sont non négligeables, l'on doit déterminer pour chaque mode la probabilité pour le système d'être dans ce mode en utilisant certaines lois classiques de probabilité. Pour un système possédant des modes de fonctionnement plus complexes, l'évaluation de cette probabilité d'état nécessite le recours à une méthode plus appropriée. L'objectif de cette section est de présenter le cadre d'utilisation de ces outils de modélisation.

1.3.1 Quelques lois de probabilité utilisées en analyse de la sûreté de fonctionnement

- La loi de poisson

Elle permet d'estimer « la probabilité qu'un évènement se produise durant un intervalle de temps donné, alors que la probabilité de réalisation de cet évènement est très faible pour un nombre d'essais très grand » (Ajit et Durga., 2010). C'est une loi caractéristique des évènements rares ou très peu probables tel que l'effondrement total d'un réseau électrique, les crashes d'avion, etc. Si l'on note par λ la moyenne d'occurrence d'un évènement rare caractérisé par la variable aléatoire X, la probabilité que celui-ci se produise K fois durant un intervalle de temps donné se traduit mathématiquement par l'expression ci-dessous :

$$P(X = K) = \frac{\lambda^K e^{-\lambda}}{K!} \qquad (1.2)$$

- La loi exponentielle

En théorie de la fiabilité, cette loi est utilisée pour modéliser la durée de vie des systèmes sans usure tel que certains dispositifs électroniques qui ne peuvent être sujet qu'à des défaillances irréversibles. Cette loi étant de type continu, si l'on note par 1/λ la durée de vie moyenne d'une entité, alors sa densité de probabilité pour un temps t s'exprime par :

$$f(t) = \begin{cases} \lambda e^{-\lambda t} & si\ t > 0 \\ 0 & si\ t < 0 \end{cases} \qquad (1.3)$$

- La loi normale

Appelées aussi loi de Gauss ou de Laplace-Gauss, elle est considérée comme l'une des lois les plus appropriées pour modéliser les phénomènes naturels issus d'une multitude d'évènements aléatoires. Concrètement, il s'agit d'une loi de type continu caractérisée par son espérance mathématique μ et son écart type σ dont la densité de probabilité s'écrit :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \qquad (1.4)$$

Lorsque sa moyenne est nul et son écart type unitaire, la loi est dite loi normale standard.

- La loi géométrique

Cette loi est généralement utilisée pour modéliser la durée de vie d'un système qui aurait à tout instant dès sa mise en fonctionnement une probabilité non nulle p de « mourir ». Mathématiquement, il s'agit d'évaluer la probabilité $p'(k)$ tel qu'au cours d'une succession de mise en service du système, l'on ait k fois une issue sans mort du système et 1 fois une issue avec mort du système. Cette loi se traduit par l'équation :

$$p'(k) = q^{1-k}p \qquad (1.5)$$

- La loi de Weibull

C'est la loi la plus utilisée en analyse quantitative de la fiabilité des systèmes du fait de sa flexibilité dans la modélisation de certaines distributions. On l'utilise pour modéliser des temps de défaillance des lampes, moteurs, transistors, pneus…Elle s'emploie également pour la planification de la maintenance préventive et la classification des types de défaillance. La loi de weibull est caractérisée par son facteur de forme β et son facteur d'échelle α. Sa densité de probabilité est donnée par l'expression ci-dessous :

$$f(t) = \begin{cases} \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1} e^{-\left(\frac{t}{\alpha}\right)^{\beta}} & si\ t > 0 \\ 0 & si\ t < 0 \end{cases} \qquad (1.6)$$

1.3.2 Les graphes de Markov

La théorie des graphes de Markov qui est utilisée pour l'analyse quantitative de la sûreté des systèmes consiste en une représentation de tous les états d'un système ainsi que les différentes transitions pouvant exister entre ces états.

Considérons un système constitué de deux composants (C1 et C2) indépendamment du type de liaison. Ces deux composants peuvent amener le système à se retrouver dans quatre configurations possibles selon qu'ils soient en panne ou en bon état. Si l'on note par λ le taux de défaillance et par μ le taux de réparation d'un composant, à travers la figure ci-dessous, nous pouvons illustrer le graphe de Markov dudit système.

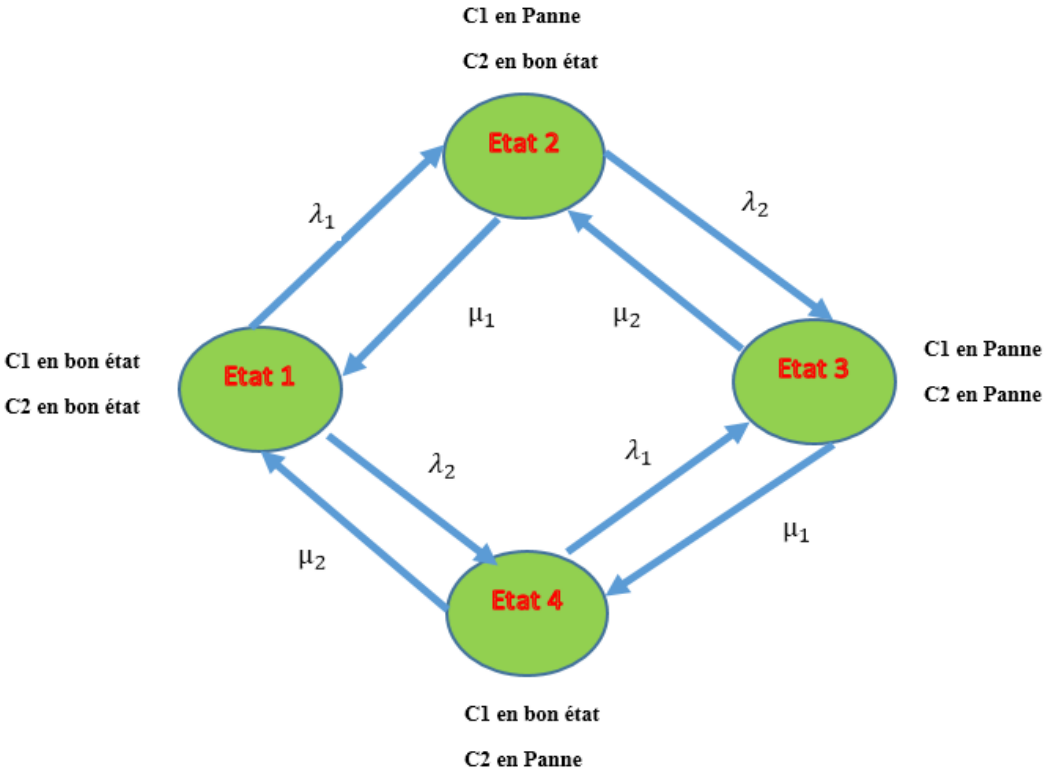

Figure 1.3: Graphe de Markov d'un système à 4 états

D'une façon générale, pour un système possédant p états, si l'on note par P_i la probabilité pour le système d'être dans l'état *i* et par T_{ij} le taux moyen de transition de l'état *i* vers un nouvel état j, alors le calcul de P_i consiste en la résolution de l'équation différentielle ci-dessous en faisant l'approximation que les transitions d'un état vers un autre suivent une loi de type exponentielle avec des taux de défaillance et de réparation constant.

$$\frac{dP_1(t+dt)}{dt}\frac{dP_2(t+dt)}{dt}\dots\frac{dP_P(t+dt)}{dt} = [P_1(t)P_2(t)\dots P_P(t)]\begin{bmatrix} T_{11} & \cdots & T_{1P} \\ \vdots & \ddots & \vdots \\ T_{P1} & \cdots & T_{PP} \end{bmatrix} \qquad (1.7)$$

1.4 Méthode empirique

La méthode empirique s'appuie sur des données facilement accessibles sans s'intéresser au mécanisme qui aurait conduit à leurs existences. La méthode empirique a pour objectif de modéliser une sorte de « boîte noire » établissant le lien statistique pouvant exister entre les données expérimentales pertinentes et les grandeurs à prédire. De nos jours, cette méthode empirique est de plus en plus expérimentée à travers les réseaux de neurones artificiels (RNA). La capacité de représentation non linéaire que possède les RNA leurs permet de simuler les systèmes d'une certaine complexité. L'adéquation entre le résultat prédit par le modèle de type « boîte noire » et la valeur expérimentale observée est obtenue par ajustement via un processus dit d'apprentissage portant sur les valeurs réelles. En plus, les réseaux de neurones nécessitent moins de paramètres ajustables (les poids des connexions) que d'autres outils mathématiques couramment utilisés. En résumé, cette méthode nous permet de modéliser un processus à partir des mesures mises à notre disposition bien que celles-ci soient entachées d'erreurs ou qu'elles ne soient pas toutes représentatives des paramètres qui conditionnent l'existence du processus.

1.5 Synthèse des différentes approches

L'analyse quantitative de la sûreté de fonctionnement a été abordée par l'approche probabiliste. Cette approche permet d'estimer uniquement la probabilité d'occurrence des pannes sans pouvoir les situer dans le temps. Cette estimation se fait en utilisant des hypothèses de simplification qui permettent d'approcher le phénomène étudié par un modèle mathématique. Nous devons souligner que ces hypothèses simplificatrices réduisent la fiabilité du résultat final en nous privant des informations qui peuvent constituer la clé de compréhension de certains phénomènes étudiés. La méthode empirique quant à elle n'exige pas d'hypothèses sur les dynamiques régissant le

processus étudié. Elle se contente du lien inné entre les variables mesurables en faisant abstraction de celles qui sont difficilement ou pas du tout accessibles. Elle produit donc en fin de compte un modèle qui traduit fidèlement le lien existant entre les variables facilement mesurables sans nécessiter une analyse mathématique très poussée.

1.6 Quelques outils d'apprentissage automatique dédiés à la prédiction

1.6.1 Les arbres de décision

Les méthodes en arbre consistent à estimer un ou plusieurs arbres de décision pour représenter les données, pour ensuite faire une prédiction basée sur l'endroit où se trouve chaque observation dans le modèle final (James *et al*., 2013). Un arbre de décision estime un modèle en appliquant une série de divisions binaires et de règles de décisions, à chaque fois en utilisant une variable pour former un nœud interne qui entrainera une séparation en deux branches distinctes.

En régression, le but est de prédire une variable *t* à partir d'un vecteur $\mathrm{X} = (x_1, \dots, x_D)$ de dimension D en entrée. Les données d'apprentissage consistent en un ensemble de vecteurs d'entrée $\{X_1, \dots, X_N\}$ et leurs étiquettes respectives$\{t_1, \dots, t_N\}$. Nous cherchons à construire un arbre de décision pour ces données. La figure 1.4 nous présente cette situation où l'espace de dimension D = 2 a été partitionné en cinq régions dont les frontières sont alignées avec les axes. La première étape divise le plan en deux régions $\{x_1 \leq \theta_1\}$ et $\{x_1 > \theta_1\}$. La région $\{x_1 \leq \theta_1\}$ peut également être subdivisée en deux régions $\{x_2 \leq \theta_2\}$ et $\{x_2 > \theta_2\}$, notées respectivement A et B. Ces régions, tout comme C, D, E constituent les régions terminales de la partition. Dans notre problème de prédiction, il faudra choisir la meilleure valeur possible dans chaque région. L'on choisit donc la valeur constante commune à toutes les régions.

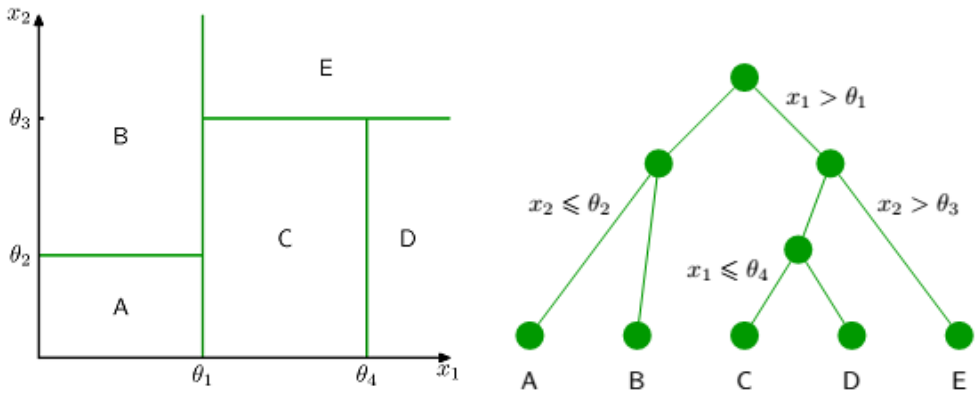

Figure 1.4 : Exemple d'un arbre de décision

1.6.2 Régression linéaire

Considérons un échantillon de n points (x_i, y_i) du plan. Un modèle de régression linéaire simple est défini par une équation de la forme :

$$y_i = \beta_1 + \beta_2 x_i + \varepsilon_i \quad \forall_i \in \{1, \dots, n\} \tag{1.8}$$

Les quantités ε_i qui constituent des erreurs sont supposés aléatoires et viennent du fait que les points ne sont jamais parfaitement alignés sur une droite. Les points (x_i, y_i) étant donnés, l'objectif est de trouver une fonction affine f tel que la quantité

$\sum_{i=1}^{n} \mathcal{L}(y_i - f(x_i))$ Soit minimale. La fonction de coût $\mathcal{L}$ doit être précisée afin de pouvoir déterminer f. Nous optons pour une fonction de coût de type quadratique. Dans ce cas, l'estimateur des moindres carrés ordinaire $\widehat{\beta_1}$ et $\widehat{\beta_2}$ permet de minimiser la quantité

$$S(\beta_1, \beta_2) = \sum_{i=1}^{n} (y_i - \beta_1 + \beta_2 x_i)^2 \tag{1.9}$$

L'équation (1.10) permet d'évaluer ces estimateurs.

$$\begin{cases} \hat{\beta}_1 = \bar{y} - \hat{\beta}_2 \bar{x} \\ \hat{\beta}_2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})y_i}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \end{cases} \quad (1.10)$$

1.6.3 Les machines à vecteur support (Vapnik V., 1998)

À partir d'un ensemble d'apprentissage $X = \{(x_1, y_1), \dots, (x_l, y_l)\} \subset \chi \times \mathbb{R}$, que l'on suppose identiquement distribué suivant une loi inconnue, on cherche à estimer une fonction *f* qui traduit la dépendance entre les variables x et y. Il s'agit donc de trouver la fonction *f* qui minimise le risque fonctionnel empirique définit par :

$$R_{emp}[f] = \frac{1}{l}\sum_{i=1}^{l} c(x_i, y_i, f(x_i)) \quad (1.11)$$

Pour un problème de régression, l'objectif est de trouver une fonction f présentant au plus une déviation maximale ε vis-à-vis des données d'apprentissage y_i et qui soit la plus régulière possible. La fonction *f* pouvant être définie par :

$$f(x) = < \omega, x > + \ b \quad \omega \in \chi \ , \ b \in \mathbb{R} \quad (1.12)$$

Finalement, le risque empirique régularisé devient donc :

$$R_{reg}[f] = \frac{1}{l}\sum_{i=1}^{l} \left| f(x_i) - y_i \right|_{\varepsilon} + \frac{\lambda}{2}\|\omega\|^2 \quad (1.13)$$

λ est une constante de régularisation permettant de contrôler l'apport du risque empirique.

Conclusion

Le chapitre qui s'achève nous a permis de présenter la notion de sûreté de fonctionnement des systèmes sur l'aspect qualitatif et quantitatif. L'aspect quantitatif a présenté certaines limites liées à la modélisation des phénomènes par des lois de probabilité. Nous avons également vu que l'aspect quantitatif peut être abordé par une méthode empirique basée sur les RNA. Au regard des atouts de la méthode empirique comparativement à l'approche probabiliste, nous l'avons choisie pour réaliser la prédiction des pannes. Le chapitre suivant sera consacré à la présentation des RNA.

CHAPITRE II

PRÉSENTATION DES RÉSEAUX DE NEURONES ARTIFICIELS

Les réseaux de neurones artificiels (RNA) sont des modèles mathématiques et informatiques, des assemblages d'unités de calculs appelés neurones formels, qui s'inspirent du modèle de la cellule nerveuse humaine. Cet héritage de la biologie humaine est une source de motivation chez certains chercheurs qui n'ont cessé au fil du temps dans de créer une sorte d'équivalence sur le plan fonctionnel entre le neurone biologique et le neurone formel. La matérialisation de cette idée a connu depuis près de trois décennies des avancés très significatives. Ces avancés sont déjà appréciables dans divers domaines tels que les milieux financiers (pour l'évaluation du risque financier), en médecine (pour le diagnostic médical), dans le domaine bancaire (pour la détection de fraude sur des cartes de crédit), en aéronautique (pour la programmation des pilotes automatiques), dans le domaine de la prévision météorologique, l'imagerie, reconnaissance de formes, traitement du signal,... Dans ce chapitre, nous nous proposons de présenter le principe de fonctionnement des architectures classiques et profondes de RNA afin de justifier le choix de celle qui sera utilisée pour réaliser la prédiction des pannes.

2.1 Présentation sommaire du neurone biologique

Le cerveau humain contient près de 10^{11} neurones et environ 10^{4} connexions entre neurones. Dans un neurone nous pouvons distinguer trois régions principales.

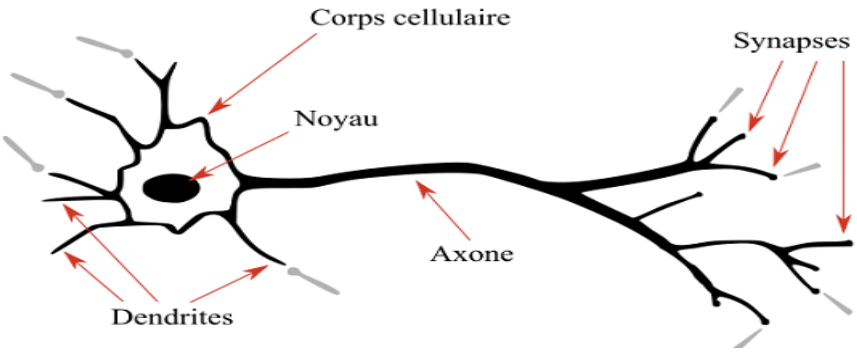

Figure 2.1: Neurone biologique

- Le corps cellulaire

Il contient le noyau du neurone et effectue les transformations biochimiques nécessaires à la synthèse des enzymes et des autres molécules qui assurent la vie du neurone. Il a une forme pyramidale ou sphérique dans la plupart des cas.

- Les dendrites

Chaque neurone possède une « chevelure » de dendrites. Celles-ci sont de fines extensions tubulaires, de quelques dixièmes de microns de diamètre et d'une longueur de quelques dizaines de microns. Ce sont les récepteurs principaux du neurone chargés de capter les signaux qui lui parvienne.

- L'axone

L'axone, qui est à proprement parler la fibre nerveuse, sert de moyen de transport. Les connexions entre deux neurones se font en des endroits appelés synapses où ils sont séparés par un petit espace synaptique de l'ordre d'un centième de micron. L'on peut tout simplement dire que le soma du neurone traite les signaux qui lui proviennent de ses dendrites, transmet le résultat de ce traitement aux neurones auxquels il est connecté par l'intermédiaire de son axone. Le schéma classique présenté par les biologistes est celui d'un soma effectuant une sommation des influx nerveux transmis par ses dendrites. Si la sommation dépasse un certain seuil, le neurone répond par un influx

nerveux au potentiel d'action qui se propage le long de son axone, dans le cas contraire, le neurone reste inactif.

- Les synapses

Ce sont des zones de contact incomplètes spécialisées dans la transmission de l'influx nerveux d'un neurone à l'autre ou d'un neurone à une fibre musculaire.

2.2 Réseaux de neurones formels ou artificiels

2.2.1 Définition

Un RNA est un processus massivement distribué en parallèle, qui présente une propension naturelle à stocker de la connaissance empirique et la rendre disponible à l'usage (Haykin, 1994).

Il est comparable au cerveau humain sur deux aspects :

- La connaissance est acquise par le réseau à travers un processus d'apprentissage ;
- Les connexions entre les neurones (poids synaptiques) servent à stocker la connaissance.

Historiquement, l'inspiration pour les RNA provient de la volonté de créer des systèmes artificiels intelligents, capables d'effectuer des opérations semblables à celles que le cerveau humain effectue de manière routinière. Le tableau 2.1 montre l'analogie existante entre le neurone formel et le neurone biologique.

Tableau 2.1 : Analogie entre neurones biologiques et artificiels

Neurones biologiques	**Neurones artificiels**
Synapses	Connections pondérés
Axons	Sorties
Dendrites	Entrées
Sommateur	Fonction d'activation

2.2.2 Modèle mathématique d'un neurone artificiel

Un neurone artificiel est essentiellement constitué d'un intégrateur qui effectue la somme pondérée de ses entrées et transforme le résultat obtenu au moyen d'une fonction de transfert. La figure 2.2 est la représentation classique d'un RNA.

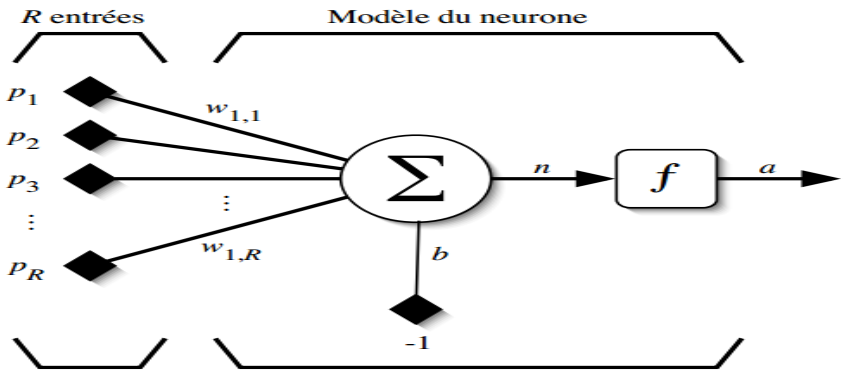

Figure 2.2 : Modèle d'un neurone artificiel (Marc Parizeau., 2004)

Les R entrées du neurone correspondent au vecteur P= $[p_1\, p_2 \ldots . p_R]^T$, alors que w= $\left[w_{1,1} w_{1,2} \ldots w_{1,R}\right]^T$ représente le vecteur des poids du neurone. La sortie n de l'intégrateur est donnée par l'équation suivante :

$$n = \sum_{j=1}^{R} w_{1,j}\, p_j - b$$

$$= w_{1,1}p_1 + w_{1,2}p_2 + \cdots + w_{1,R}p_R - b \qquad (2.1)$$

L'on peut aussi écrire (2.1) sous la forme matricielle par :

$$n = W^T P - b \qquad (2.2)$$

Cette sortie correspond à une somme pondérée des poids et des entrées moins ce qu'on nomme le biais *b* du neurone. Le résultat *n* de la somme pondérée s'appelle le niveau d'activation du neurone. Lorsque le niveau d'activation atteint ou dépasse le seuil *b*, alors l'argument de *f* devient positif ou nul sinon, il est négatif. Un autre facteur limitatif dans ce modèle concerne son caractère discret. En effet, pour pouvoir simuler un réseau de neurones, il faut rendre le temps discret dans les équations. Autrement dit, nous allons supposer que tous les neurones sont synchrones, c'est à dire qu'à chaque temps *t*, ils vont simultanément calculer leur somme pondérée et produire une sortie $a(t) = f(n(t))$. Dans les réseaux biologiques, tous les neurones sont en fait asynchrones. Revenons donc à notre modèle tel que formulé par l'équation (2.2) et ajoutons la fonction d'activation *f* pour obtenir la sortie d'un neurone :

$$a = f(n) = f(W^T P - b) \qquad (2.3)$$

En posant W= W^T, on obtient la formule générale qui est :

$$a = f(WP - b) \qquad (2.4)$$

L'équation précédente permet de créer le modèle matriciel compact de la figure 2.3.

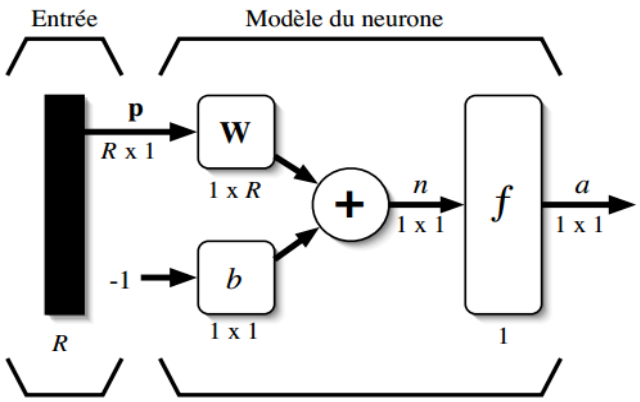

Figure 2.3: Modèle matricielle d'un neurone artificiel (Marc Parizeau., 2004)

On y représente les *R* entrées comme un rectangle noir. De ce rectangle sort le vecteur *P* dont la dimension matricielle est *R*×1. Ce vecteur est multiplié par une matrice *W* qui contient les poids synaptiques du neurone. Dans le cas d'un neurone simple, cette matrice possède la dimension 1 × R. Le résultat de la multiplication correspond au niveau d'activation qui est ensuite comparé au seuil *b* (un scalaire) par soustraction. Finalement, la sortie du neurone est calculée par la fonction d'activation f. La sortie d'un neurone est toujours un scalaire.

2.2.3 Définitions de quelques termes propres aux RNA

Poids d'un neurone : le poids $w_{ij}^{(k)}$ de connexion d'un neurone a_j^{k-1} de la couche *(k-1)* à un neurone a_i^k de la couche *k* correspond à l'importance du neurone *j* de valeur a_j^{k-1} dans l'émission du signal du neurone *i* qui prendra la valeur a_i^k. Le poids est un réel qui peut être positif ou négatif.

Biais d'un neurone : le biais b_i^k est un poids constant (nombre réel positif ou négatif) applicable lors du calcul d'un signal a_i^k du neurone *i* appartenant à la couche *k*.

L'intégrateur : il consiste à regrouper l'ensemble des neurones a^{k-1} de la couche *(k-1)* afin de calculer le terme d'intégration permettant de déterminer la valeur d'un neurone a_i^k de la couche *k*. L'intégration par somme pondérée est cependant la plus utilisée.

Couche d'entrée : couche contenant des neurones factices transmettant les variables d'entrée au réseau de neurone.

Sortie du neurone : c'est la valeur distribuée vers d'autres neurones ou simplement la valeur de sortie du réseau de neurone.

Fonction d'activation : c'est l'action de transformer le résultat de l'intégration d'un neurone *i* appartenant à la couche *k* en une valeur comprise entre -1 et +1.

La fonction d'activation doit être non linéaire (Gybenco., 1989), différentiable en tout point, étendue (Wu., 2009), monotone et posséder une identité en 0. Il existe plusieurs types de fonctions d'activation.

- La fonction seuil

Elle applique un seuil à son entrée tel que pour une valeur négative, la fonction retourne la valeur 0 et pour une valeur positive ou nulle, la fonction retourne la valeur 1. Cette fonction est très utilisée pour la prise des décisions de type 'TOUT ou RIEN'. Le biais *b* qui figure dans l'équation (2.4) permet de fixer sur l'axe des abscisses la position du seuil où la fonction passe de 0 à 1. Son expression mathématique est donnée par (2.5)

$$\begin{cases} a = 0 \;\; si\; n < 0 \\ a = 1 \;\; si\; n > 0 \end{cases} \qquad (2.5)$$

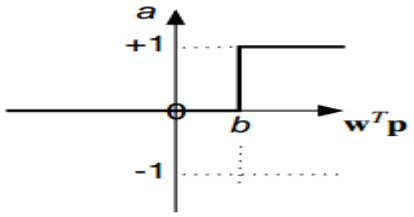

Figure 2.4: Fonction de transfert de type seuil

- La fonction sigmoïde

Elle sature à une valeur élevée lorsque son argument est grand et inversement dans le cas contraire. Cela a pour conséquence d'annuler facilement sa dérivée et de mettre ainsi fin au processus d'apprentissage. Cette fonction est par contre sensible lorsque la valeur de l'argument d'entrée est proche de 0. L'expression mathématique et la représentation géométrique sont les suivantes :

$$a = \frac{1}{1+e^{-n}} \tag{2.6}$$

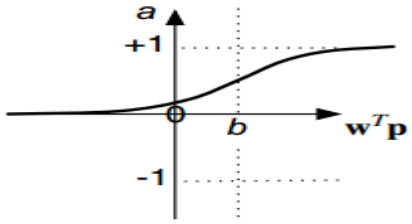

Figure 2.5: Fonction sigmoïde

- La fonction tangente hyperbolique

Cette fonction est utilisée comme fonction d'activation à l'intérieur du réseau parce qu'elle admet des valeurs positives et négatives ce qui augmente la capacité d'apprentissage du réseau. Cependant, elle est déconseillée pour la dernière couche si le résultat doit être le fruit d'une classification (valeur de 0 ou 1). Son expression mathématique est donnée par (2.7) et sa représentation graphique par la figure (2.6) :

$$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}} \tag{2.7}$$

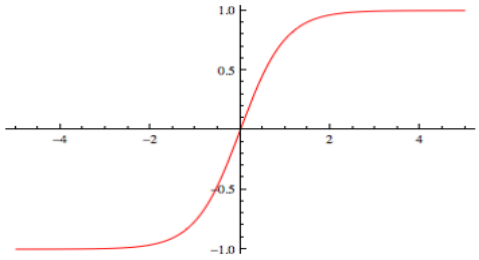

Figure 2.6 : Fonction tangente hyperbolique

- La fonction *Rectified Linear Unit* : *ReLu*

Elle est graphiquement similaire à une fonction linéaire à la seule différence qu'elle vaut zéro sur tout $\mathbb{R}^{-}$. « Des recherches montrent que *ReLu* est un meilleur modèle car il réduit la probabilité du '*vanishing gradient'* » (Glorot et al., 2011). En effet, La fonction *ReLu* est réputée pour bien propager l'erreur en raison de sa pente qui vaut 1. Toutefois, si son activation est toujours négative peu importe le vecteur d'entrée, le neurone sera alors 'désactivé' ou 'mort'. Ainsi, il y aura moins de neurone pour mettre à jour le réseau. Il est donc souhaitable d'utiliser la fonction d'activation *ReLu* lorsqu'on dispose d'un grand nombre de neurones dans une couche. Son expression mathématique et sa représentation graphique sont données par l'équation (2.8) et la figure 2.7.

$$ReLu = \begin{cases} 0 & si \;\; x < 0 \\ y & si \;\; x \geq 0 \end{cases} \tag{2.8}$$

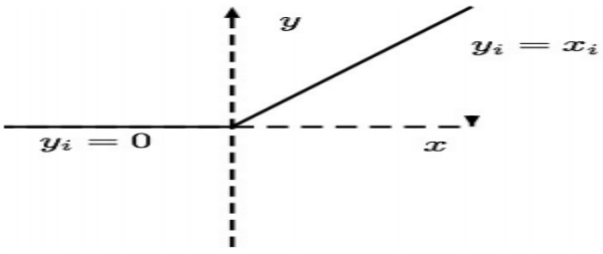

Figure 2.7: fonction *ReLu*

2.3 Architectures classiques de RNA

Dans un réseau, chaque sous-groupe fait un traitement indépendant des autres et transmet le résultat de son analyse au sous-groupe suivant. L'information donnée au réseau va donc se propager couche par couche, de la couche d'entrée à la couche de sortie, en passant par aucune ou plusieurs couches intermédiaires (dites couches cachées). A l'exception des couches d'entrée et de sortie, chaque neurone dans une couche est connecté à tous les neurones de la couche précédente et de la couche suivante. Les RNA ont la capacité de stocker de la connaissance empirique et de la rendre disponible à l'usage. Ces habiletés de traitement de la connaissance par le réseau vont être stockées dans les poids synaptiques, obtenus par le processus d'apprentissage. Les RNA ressemblent donc au cerveau car non seulement, la connaissance est acquise à travers un apprentissage mais est également stockée dans les connexions entre les entités. « Plusieurs architectures de réseaux de neurones aux propriétés très diverses ont été développés depuis l'apparition des neurones formels dans les années 1940 » (Mc Cullot et Pitts., 1943). Globalement, l'on distingue les architectures cycliques et acycliques. Ces deux familles possèdent chacune plusieurs variantes issues d'une structure basique qui est le perceptron.

2.3.1 Le perceptron monocouche

Le perceptron qui a été mis sur pied par Frank Rosenblatt en 1958 peut être considéré comme un réseau de neurones ne possédant qu'une seule sortie sans couche cachée. Il s'agit d'un classificateur linéaire qui a pour objectif d'effectuer la séparation linéaire de deux classes. En effet, considérons le modèle de la figure 2.3 pour deux entrées $x_1 \; et \; x_2$ ainsi que la valeur du biais b fixée à 1. Le problème ici consiste à trouver la bonne combinaison des valeurs des poids $w_1 , w_2 \; et \; w_b$ afin d'obtenir en sortie des valeurs désirées illustrant la réalisation d'une fonction de classification quels que soient

les valeurs prises par x_1 et x_2. Pour $w_1 = 1, w_2 = 1$ et $w_b = 0$, la sortie du perceptron réalise la fonction OU logique. Quand on fait une représentation graphique illustrant les valeurs de sortie du perceptron en fonction des combinaisons possibles des entrées, l'on constate que les valeurs prises par la sortie sont linéairement séparées en deux groupes de « 1 » et « 0 » tel qu'illustré par la figure 2.8. Notons également que la sortie du perceptron réalise aussi la fonction ET logique pour une autre combinaison des valeurs des poids indépendamment de celles prises par les entrées). Cependant, en 1969, M. Minsky et S. Papert ont publié un livre dans lequel ils mettaient en exergue les limites du perceptron. En effet, d'après la figure 2.8, quand $x_1 = 0$, la sortie du perceptron doit augmenter, si x_2 augmente et $x_1 = 1$, la sortie du perceptron doit décroître. Le premier cas exige que $w_2 > 0$ et le second exige quant à lui que $w_2 < 0$. Cette incapacité d'apprentissage illustre la limite du perceptron et justifie donc l'exploration d'une autre variante du perceptron dite perceptron multicouches (PMC) aussi appelée *multy layer perceptron* (MLP).

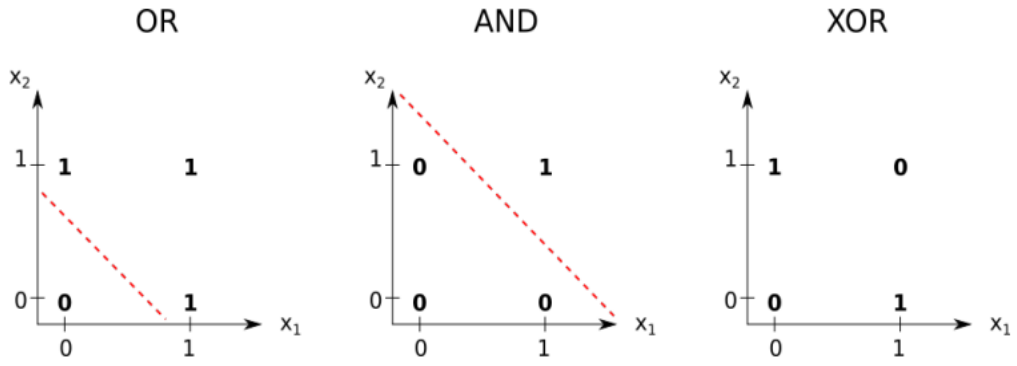

Figure 2.8: Fonctions OU et ET linéairement séparables (trait rouge) et XOR non séparable

2.3.2. Le perceptron multicouche

Le MLP tire ses origines des travaux de F. Rosenblatt. C'est un RNA acyclique composé d'une couche d'entrée, d'une ou plusieurs couches intermédiaires dites cachées et d'une couche de sortie. Le MLP est capable de réaliser plusieurs types de fonction à condition que ses paramètres d'apprentissage (poids) ainsi que ses fonctions d'activation à utiliser pour les couches cachées et de sorties soient bien choisies. Nous pouvons illustrer cela en réalisant la fonction logique XOR qui a mis en exergue la limite du perceptron dans le paragraphe précédent. En effet, considérons une configuration avec 3 entrées comme précédemment mais possédant deux neurones dans sa couche cachée et 1 neurone dans la couche de sortie. Si l'on s'inspire du modèle de représentation matricielle d'un RNA, alors cette configuration peut s'illustrer graphiquement à travers la figure 2.9 dans laquelle une combinaison des différentes valeurs des poids du réseau est présentée par les deux matrices des poids que voici :

$$\begin{cases} W^{(1)} = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} \\ W^{(2)} = [0 \quad 1 \quad -2] \end{cases}$$

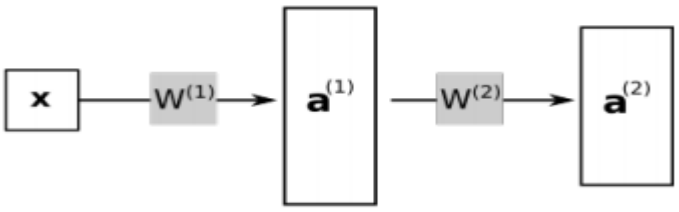

Figure 2.9: Représentation matricielle d'un MLP

Ces valeurs attribuées aux matrices des poids permettent de modifier l'espace de représentation des données d'entrées de la figure 2.9 en un autre espace présenté par la figure 2.10 et permettent ainsi de séparer linéairement les sorties de la fonction XOR. L'on peut donc dire que les couches cachées constituent des représentations au plus haut niveau des entrées.

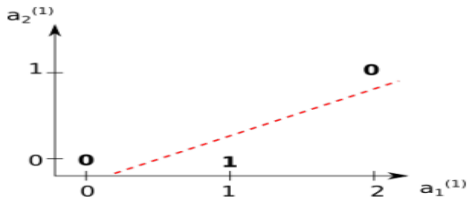

Figure 2.10: Fonction XOR réalisée par un MLP à deux couches cachées

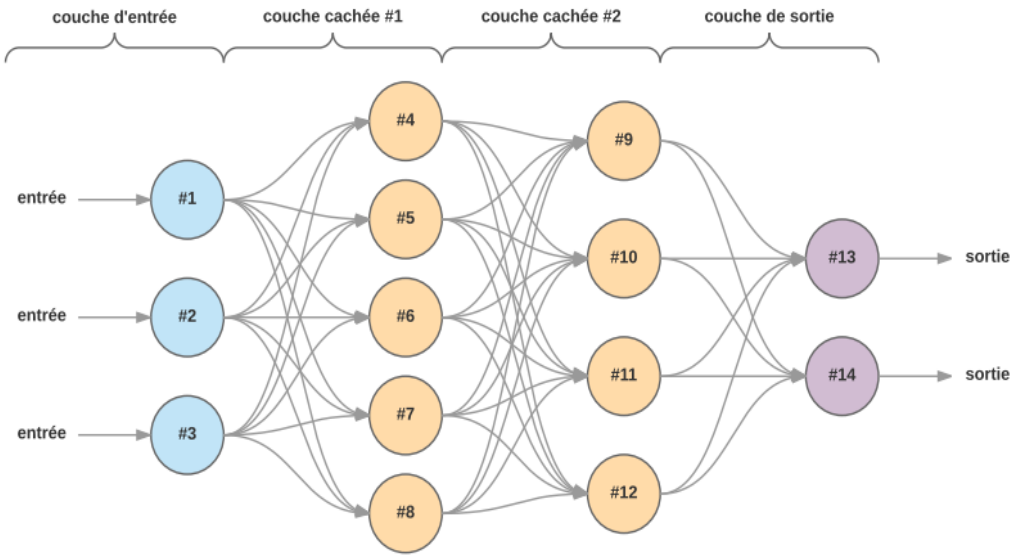

Figure 2.11: MLP à deux couches cachées

2.3.3 Les réseaux de neurones à fonction à base radiale

Ces réseaux ont été introduit par HARDY en 1971 mais la théorie y afférente fût développée par POWEL en 1985. Ils ont connu l'appellation réseau de neurone grâce aux travaux de Lowe et Broomhead. Les réseaux de neurones à fonction de base radiale (RBF) sont constitués d'une couche d'entrée et de sortie, une couche cachée comportant *n* unités qui réagissent significativement à une partie de l'espace d'entrée (échantillon) suivant une fonction d'activation de type gaussienne. Dans la pratique, les RBF sont beaucoup plus utilisés dans des réseaux à une seule couche cachée. Ils trouvent leurs applications dans le traitement de la parole, de l'image et du signal. La fonction

d'activation de ce réseau est de type radial. Sa réponse croît ou décroît de façon monotone par rapport à un point central. Typiquement, l'expression mathématique de la fonction d'activation est donnée ci-dessous.

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2}\right) \qquad (2.9)$$

r et *c* sont respectivement le rayon et le centre de la fonction. La figure 2.12 représente la fonction d'activation ainsi que la structure d'un réseau de type RBF.

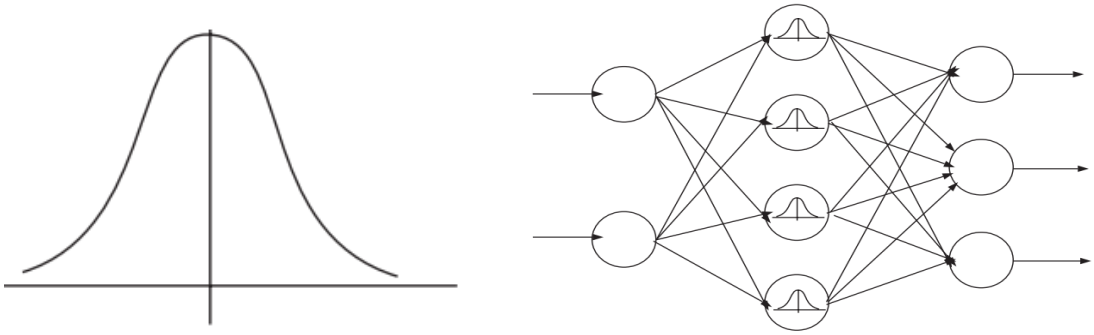

Figure 2.12: Fonction d'activation à gauche et topologie à droite du RNA de type RBF

2.3.4 Les réseaux cycliques

Appelés aussi "réseaux récurrents", ce sont des réseaux dans lesquels il y a retour en arrière de l'information. Pour qu'un tel système soit causal, il faut évidemment qu'à toute boucle soit associé un retard. Puisque l'immense majorité des applications sont réalisées par des programmes d'ordinateurs, on se place dans le cadre des systèmes à temps discret où les équations différentielles sont remplacées par des équations aux différences.

2.3.4.1 Les réseaux de Kohonen

Ce sont des réseaux qui établissent une carte discrète, ordonnée topologiquement, en fonction des patterns d'entrée. La figure 2.13 présente ce réseau constitué d'une sorte de treillis dont chaque nœud est un neurone associé à un vecteur de poids. La correspondance entre chaque vecteur de poids est calculée pour chaque entrée. Il s'agit donc d'un réseau de neurone à compétition. Ces réseaux trouvent leurs applications dans le domaine de la robotique, la compression des données et la statistique (méthode ACP).

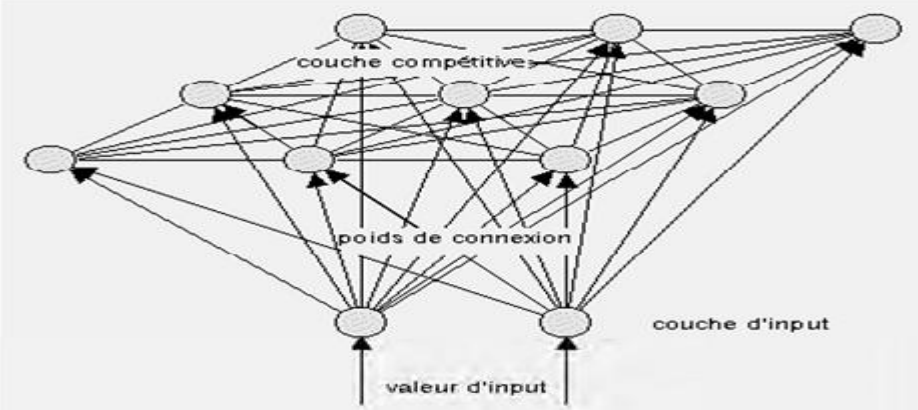

Figure 2.13: Forme basique d'une carte topologique de Kohonen

2.3.4.2 Les réseaux de Hopfield

Ils ont proposé en 1982 par le physicien John Hopfield. Ce sont des réseaux récurrents et entièrement connectés (Figure 2.14). Dans ce type de réseau, l'interconnexion entre neurones ne permet pas de distinguer les neurones d'entrée et de sortie. Ils fonctionnent comme une mémoire associative non-linéaire et sont capables de trouver un objet stocké en fonction des représentations partielles ou bruitées. L'application principale des réseaux de Hopfield est l'entrepôt de connaissances mais aussi la résolution de problèmes d'optimisation. Il s'agit d'une

structure dite complètement connectée puisque chaque neurone est connecté à tous les autres. Les neurones ont un état binaire (classiquement, 1 ou -1, même si certains modèles utilisent 1 et 0).

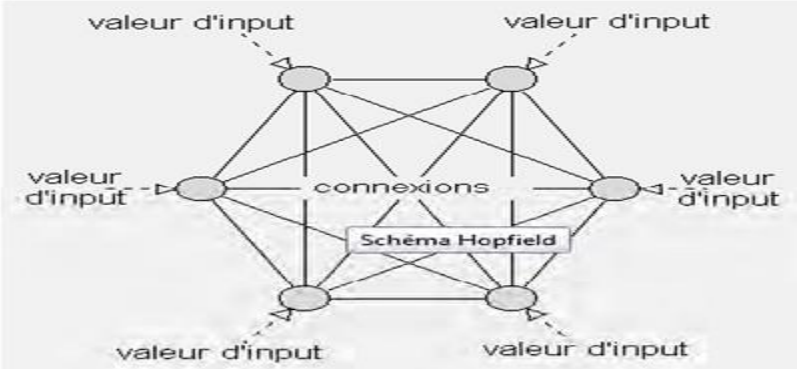

Figure 2.14: Modèle de Hopfield

2.4. Limites des architectures de RNA classique

Nous avons constaté plus haut que le perceptron était incapable d'apprendre la fonction XOR à cause de sa structure monocouche mettant ainsi en exergue son manque de profondeur. Le MLP quant à lui a bel et bien réussi cet exercice grâce à sa couche cachée. Cela nous a permis de dire que la couche cachée assurait une représentation au plus haut niveau des données d'entrées. Les architectures de RN présentées ci-dessus ont en général très peu de couches cachées dans leurs configurations ce qui leurs confère l'appellation de réseaux peu profond. Comme conséquence de ce manque de profondeur, l'on peut citer :

- L'obligation pour l'utilisateur d'effectuer un prétraitement sur les données d'entrées avant de les présenter au RN ;
- L'incapacité du réseau à extraire les traits caractéristiques extrêmement fins d'une image lors d'un processus de classification d'images ;

- L'inaptitude du réseau à traiter correctement des grandes quantités de données en entrées.

2.5 Architectures de RNA profondes

De nos jours, les réseaux profonds peuvent posséder plus de 100 couches cachées. L'objectif principal de cette architecture est de confier l'extraction des caractéristiques des variables d'entrées à un processus d'apprentissage effectué par les premières couches du réseau. Plusieurs architectures de RNA profonds ont vu le jour durant ces dernières années. Dans ce paragraphe, nous n'en présenterons que quelques-unes tout en précisant leurs champs d'application. Il n'est pas toujours aisé de comparer les performances de toutes ces architectures. Cette difficulté est liée d'une part au fait qu'elles ne traitent pas les mêmes types de données et d'autre part, l'on observe quasiment tous les mois l'apparition des nouvelles variantes d'architectures de RNA profondes.

2.5.1 Les réseaux de type CNN

Les réseaux CNN ont été développés par Yann LeCun en 1990 à travers le réseau LeNet. Son graphe fonctionnel est donné par la figure (2.15). Ce réseau utilise l'opération mathématique de convolution en lieu et place de la multiplication matricielle dans au moins l'une des couches constitutives. L'on peut définir les CNNs comme une classe de RNA dans laquelle les groupes de neurones artificiels identifient les modèles invariants. « Les CNNs sont largement utilisés dans le domaine de la classification d'images, reconnaissance de caractères, classification de texte, robotique, du traitement audio et vidéo. (Dario A Modei et Al., 2016).

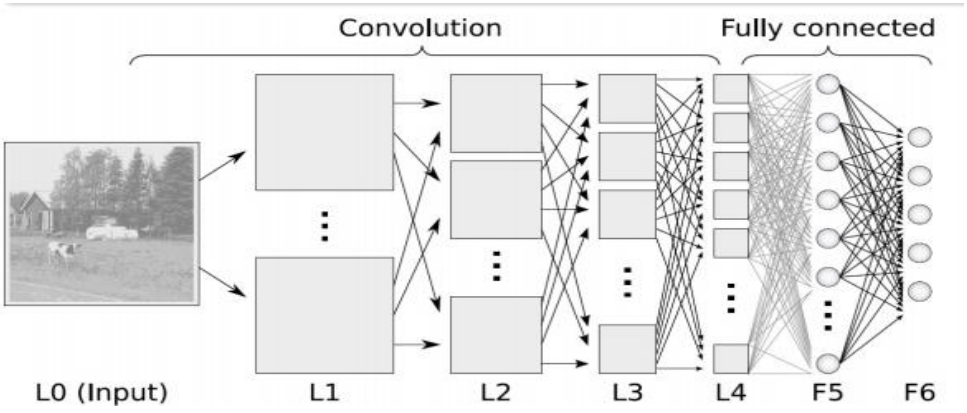

Figure 2.15: Représentation synoptique d'un CNN

Un CNN est globalement constitué de trois types de couche : entrée, convolution et classification.

2.5.1.1 Couches de convolution

Elles réalisent les opérations de convolution, de *pooling* et de rectification d'unité linéaire(*ReLu*). L'opération de convolution fait passer les images d'entrées à travers un ensemble de filtres convolutifs afin que ces derniers activent ou ressortent certaines caractéristiques particulières de ces images. L'opération de *pooling* ou de mise en commun simplifie la sortie en effectuant un échantillonnage non linéaire qui permet de réduire le nombre de paramètre que le réseau doit apprendre. *ReLu* favorise un entraînement plus rapide et plus efficace en transformant les valeurs négatives en zéro et en maintenant les valeurs positives.

2.5.1.2 Couches de classification

Elles permettent de ressortir un vecteur de dimension k (k représente le nombre de classe dont le réseau est capable de prédire). Ce vecteur contient la probabilité pour chaque classe d'image d'être classifiée. Ici, la dernière couche utilise la fonction d'activation *softmax* pour fournir la classification.

Les CNNs apprennent moins de paramètres que les autres RNA mais réalisent en contrepartie plusieurs opérations et exigent donc des calculateurs plus puissants pour leurs mis en œuvre. Parmi les variantes les plus connu de CNN, l'on peut citer :

- LeNet

Développée par Yann LeCun dans les années 1990, l'architecture LeNet utilisée pour lire les codes postaux, les chiffres…

- Alex Net

Développé par Alex Krizhevsky, Ilya Sutskever et Geoff Hinton, Alex Net a été soumis au défi Image Net ILSVRC en 2012 et a nettement surpassé ses concurrents. Le réseau avait une architecture très similaire à LeNet, mais était plus profond, plus grand et comportait 35 couches convolutives empilées les unes sur les autres (auparavant, il était commun de ne disposer que d'une seule couche convolutive toujours immédiatement suivie d'une couche de pooling).

- Google Net

Le vainqueur de ILSVRC challenge 2014 était un réseau convolutif de Szegedy et al. De Google. Sa principale contribution a été le développement d'un *module inception* qui a considérablement réduit le nombre de paramètres dans le réseau.

2.5.2 Les réseaux de neurones récurrents : RNN

Un réseau de neurone récurent est un réseau de neurone artificiel constitué d'unités interconnectées interagissant non linéairement et pour lequel il existe au moins un cycle dans la structure.

Ils sont adaptés au traitement des données séquentielles (ou des séries temporelles). Ils calculent au temps *t* leur sortie en fonction de l'entrée correspondante x_t et de l'état de

la couche cachée au temps précédent. Ils font ainsi évoluer un état interne qui joue le rôle de mémoire à court terme permettant ainsi de tenir compte des dépendances temporelles des variables d'entrées.

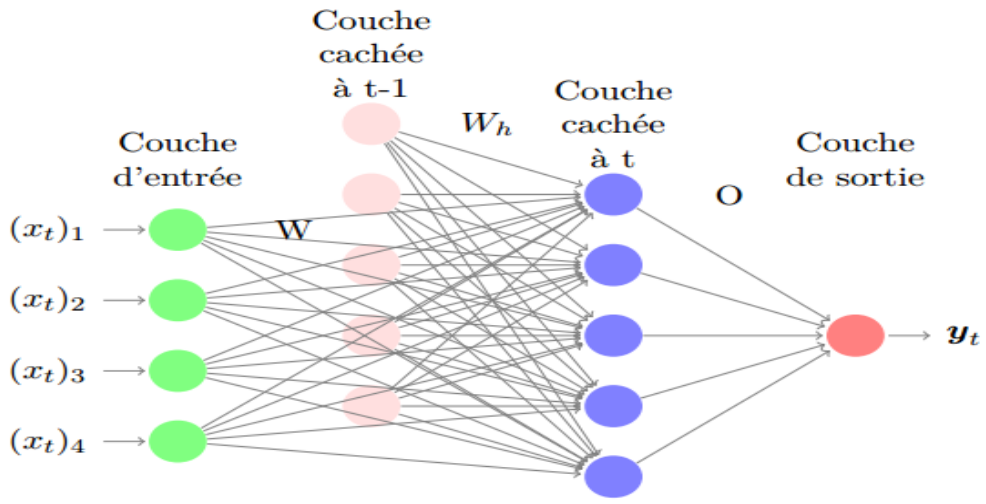

Figure 2.16: Schéma d'un RNN

Il n'est pas toujours aisé d'entraîner les RNNs afin qu'ils puissent réaliser leurs meilleures performances. L'utilisation des méthodes d'apprentissage sur ce réseau fait face au problème de « *Vanishing and exploding gradient* » (Bengio et *al*., 1994). Ces modèles ne prennent en compte que des dépendances à très court terme, c'est la raison pour laquelle d'autres architectures tel que les *LSTM* et les *GRU* ont vues le jour afin de remédier à ce problème.

2.5.3 Réseau de type *LSTM*

Un *LSTM* qui signifie mémoire à court et long terme peut être définit comme un modèle de cellule spécial capable d'apprendre des dépendances à long terme et de se souvenir d'une information pendant de longues périodes.

L'idée derrière les *LSTM* est de diviser le signal entre ce qui est important à court terme à travers le *hidden state* et ce qui le sera à long terme via la *cell state*.

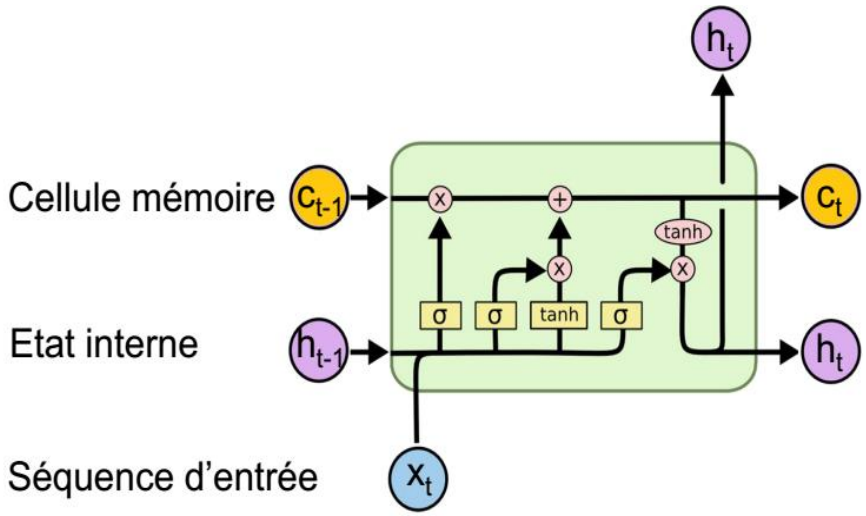

Figure 2.17: Forme compact d'un LSTM. (http://colah.github.io).

Le fonctionnement d'un LSTM peut ainsi se résumer aux trois étapes ci-dessous :

- Détecter les informations pertinentes venant du passé stocké dans la *cell state* au moyen du *forget gate ;*
- Choisir au niveau de l'entrée courante au moyen de l'*input gate* les informations qui seront pertinentes à long terme et rajoutées à la *cell state* qui est la mémoire longue ;
- Récupérer les nouvelles informations importantes à court terme de la *cell state* pour produire le *hidden state* au moyen de l'*output gate*.

Il ressort de ce qui précède que le *LSTM* est d'une grande utilité pour la prédiction des série temporelles car en mémorisant ce qui s'est déroulé dans le passé, elle peut prédire une action à l'instant *t.*

2.5.4 *Deep generative model*

« Tandis que des modèles dis discriminatifs tel que *CNN*, *RNN*, *MLP* essayent de prédire p(y/x) avec *y* étant le label et *x* l'entrée, un modèle génératif décrit comment

les données sont générées. Il apprend ainsi *p(x, y)* et fait des prédictions en utilisant la loi de Bayes pour calculer *p(y/x) »*(Ng et Jordan., 2002). Si les architectures citées plus haut permettent de réaliser la classification d'images ou la prédiction, le *deep generative model* est une architecture qui peut aller au-delà en générant des nouvelles observations. C'est ainsi que plusieurs variantes de ce modèle ont vu le jour tel que la « machine de Boltzmann » (Ackley et *al*., 1985), *Deep Belief Networks*, *Deep Boltzmann Machines*, « *Generative Stochastic Networks* » (Y. Bengio et *al*., 2013), etc.

2.6 Apprentissage des RNA

L'apprentissage des RNA peut se définir comme « une tentative de comprendre et reproduire la faculté d'apprentissage humaine dans des systèmes artificiels qui s'implémentent au moyen d'algorithmes capables, à partir d'un nombre important d'exemples (les données correspondant à l'expérience passée), d'en assimiler la nature afin de pouvoir appliquer ce qu'ils ont ainsi appris aux cas futurs » (Ange Tato., 2018).

2.6.1 Apprentissage supervisé

Il s'agit de contraindre le réseau de converger vers un état bien déterminé tout en lui présentant des motifs. L'on crée donc un modèle qui peut être de prédiction, de classification, … En se servant d'une base de données dite d'apprentissage constituée des paramètres d'entrées et de sortie du processus à modéliser jusqu'à ce que le modèle s'adapte par comparaison des sorties désirées avec celles obtenues. Une fois le modèle obtenu, celui-ci est soumis à un test à travers une autre base de données différente de la base d'apprentissage. La validité du modèle sera établie après l'analyse du résultat de la métrique d'évaluation du réseau.

2.6.2 Apprentissage non supervisé

Dans cet apprentissage, un algorithme explore les données d'entrées sans avoir une variable de sortie explicite. Il est utilisé quant on ne sait pas comment classer les données et souhaitons que l'algorithme y trouve des modèles. Le *clustering* est un algorithme utilisé en apprentissage non supervisé puisqu'il permet de réaliser une construction automatique des classes en se basant sur un critère de similarité entre les données (Martins, Nobre, Cardoso, Delbem, & Marques, 2016).

2.6.3 Apprentissage par renforcement

L'apprentissage renforcé est une technique similaire à l'apprentissage supervisé à la différence qu'au lieu de fournir des résultats désirés au réseau, on lui accorde plutôt un grade (ou score) qui est une mesure du degré de performance du réseau après quelques itérations.

L'apprentissage a ainsi pour objectif de mieux entraîner le réseau afin qu'il puisse réduire la différence entre ses sorties réels et les sorties désirées. Il s'agit donc de minimiser l'erreur du réseau. L'une des méthodes d'apprentissage les plus utilisées est celle de la rétro-propagation du gradient de l'erreur qui consiste à diffuser l'erreur générer par le RNA en son sein mais dans le sens contraire de la propagation de ses activations.

2.7. Méthode de la rétro propagation du gradient (RPG)

Considérons le réseau de la figure 2.18 dont la représentation matricielle concorde avec les notations du paragraphe 2.2. Nous présentons ici la méthode de la RPG (Marc Parizeau., 2004).

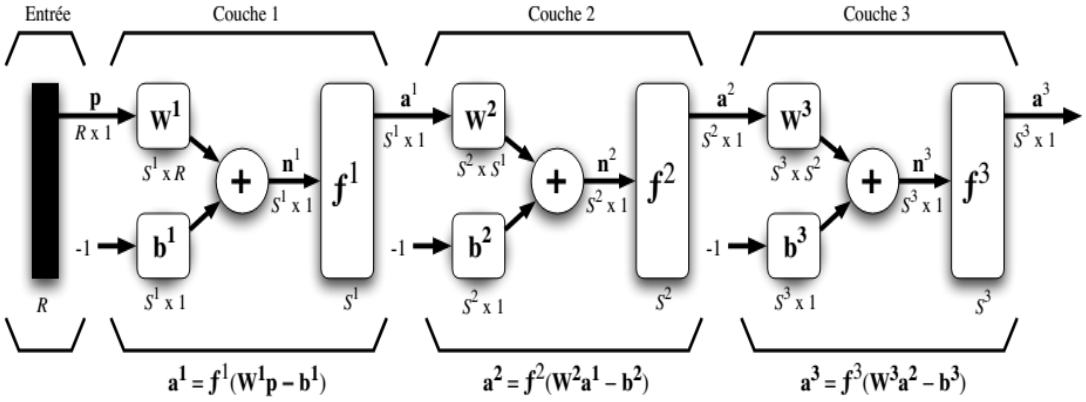

Figure 2.18: Forme matricielle d'un réseau MLP à 3 couches (Marc Parizeau., 2004)

À partir de l'équation (2.4) régissant la sortie d'un réseau à une seule couche illustrée par la figure 2.3, l'on peut déduire celle de la figure ci-dessus en disant que la sortie d'une couche *k* dans un MLP est donnée par **:**

$$a^k = f^k(W^k a^{k-1} - b^k), \text{ avec } k = 1, \ldots, M, \qquad (2.12)$$

Chaque sortie du réseau correspond dès lors à a^M sachant que *M* représente le nombre total de couche et $a^0 = P$ constitue le cas basique de l'expression de récurrence. L'erreur *e(t)* entre la cible *d(t)* et la sortie du réelle du réseau vaut :

$$e(t) = d(t) - a(t) \qquad (2.13)$$

Afin de minimise l'erreur quadratique moyenne, l'on définit un indice de performance F. On regroupe l'ensemble des poids et biais du réseau dans un vecteur X tel que :

$$F(X) = E[e^T(t)e(t)] \qquad (2.14)$$

L'écriture E[.] est celle l'espérance mathématique. L'on approxime l'expression (2.14) par :

$$\hat{F}(X) = e^T(t)e(t) \tag{2.15}$$

Par définition, la descente de gradient s'évalue par les formules ci-dessous :

$$\Delta w_{i,j}^k(t) = -\eta \frac{\partial \hat{F}}{\partial w_{i,j}^k} \tag{2.16}$$

$$\Delta b_i^k(t) = -\eta \frac{\partial \hat{F}}{\partial b_i^k} \tag{2.17}$$

η représente le taux d'apprentissage, $w_{i,j}^k$(t) est le poids qui relie le neurone *i* à son entrée *j* dans la couche *k* à l'instant t. En appliquant la technique de chaînage des dérivés aux termes de droite des deux équations précédentes, l'on obtient les 2 expressions ci-dessous :

$$\frac{\partial \hat{F}}{\partial w_{i,j}^k} = \frac{\partial \hat{F}}{\partial n_i^k} \times \frac{\partial n_i^k}{\partial w_{i,j}^k} \tag{2.18}$$

$$\frac{\partial \hat{F}}{\partial b_i^k} = \frac{\partial \hat{F}}{\partial n_i^k} \times \frac{\partial n_i^k}{\partial b_i^k} \tag{2.19}$$

L'on peut généraliser (2.2) sachant que les niveaux d'activation n_i^k d'une couche *k* sont liés aux poids et aux biais de ladite couche en réécrivant l'un des termes communs aux deux équations précédentes comme suit :

$$n_i^k = \sum_{j=1}^{s=k-1} w_{i,j}^k a_j^{k-1} - b_i^k \tag{2.20}$$

On peut ainsi déduire que :

$$\begin{cases} \frac{\partial n_i^k}{\partial w_{i,j}^k} = a_j^{k-1} \\ \frac{\partial n_i^k}{\partial b_i^k} = -1 \end{cases} \quad (2.21)$$

A partir de (2.18) et (2.19) l'on définit la sensibilité du neurone *i* de la couche *k* par rapport au niveau d'activation de ladite couche par :

$$s_i^k = \frac{\partial \hat{F}}{\partial n_i^k} \quad (2.22)$$

En insèrent (2.22) dans (2.18) et (2.19) on obtient :

$$\frac{\partial \hat{F}}{\partial w_{i,j}^k} = s_i^k a_j^{k-1} \quad (2.23)$$

$$\frac{\partial \hat{F}}{\partial b_i^k} = - s_i^k \quad (2.24)$$

(2.16) et (2.17) deviennent donc :

$$\Delta w_{i,j}^k(t) = -\eta s_i^k(t) a_j^{k-1}(t) \quad (2.25)$$

$$\Delta b_i^k(t) = \eta s_i^k(t) \quad (2.26)$$

Les équations (2.25) et (2.26) s'écrivent en notation matricielle comme suit :

$$\Delta W^k(t) = -\eta s^k(t)(a^{k-1})^T(t) \quad (2.27)$$

$$\Delta b^k(t) = \eta s^k(t) \quad (2.28)$$

Le terme commun aux deux expressions précédentes est une matrice de *k* lignes et une colonne qui se note comme suit :

$$s^k \equiv \frac{\partial \hat{F}}{\partial n^k} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n_1^k} \\ \frac{\partial \hat{F}}{\partial n_2^k} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n_i^k} \end{bmatrix} \tag{2.29}$$

Nous pouvons à présent évaluer les sensibilités s^k en faisant une fois de plus recours à la notion de chaînage des dérivés ce qui nous permettra d'obtenir une formule de récurrence dans laquelle les sensibilités des couches d'entrées sont directement liées à celles des couches de sorties contrairement à ce qui est matérialisé par (2.4) où l'information circule des entrées vers la sortie : d'où l'origine même du vocable « rétro propagation » utilisé ici. Pour y parvenir, évaluons la matrice des dérivés des niveaux d'activation des couches.

$$\frac{\partial n^{k+1}}{\partial n^k} = \begin{bmatrix} \frac{\partial n_1^{k+1}}{\partial n_1^k} & \frac{\partial n_1^{k+1}}{\partial n_2^k} & \cdots & \frac{\partial n_1^{k+1}}{\partial n_{s^k}^k} \\ \frac{\partial n_2^{k+1}}{\partial n_1^k} & \frac{\partial n_2^{k+1}}{\partial n_2^k} & \cdots & \frac{\partial n_2^{k+1}}{\partial n_{s^k}^k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial n_{s^{k+1}}^{k+1}}{\partial n_1^k} & \frac{\partial n_{s^{k+1}}^{k+1}}{\partial n_2^k} & \cdots & \frac{\partial n_{s^{k+1}}^{k+1}}{\partial n_{s^k}^k} \end{bmatrix} \tag{2.30}$$

La matrice ci-dessus liste l'entièreté des sensibilités des niveaux d'activation d'une couche par rapport à celle qui la précède. Prenons en compte à présent chaque neurone *i* et une entrée *j* tel que le couple (*i,j*) puisse permettre de parcourir la matrice précédente. L'on peut donc écrire pour chaque élément de ce couple que :

$$\frac{\partial n_i^{k+1}}{\partial n_j^k} = \frac{\partial}{\partial n_1^k}\left(\sum_{l=1}^{S^K} w_{i,l}^{k+1} a_l^k - b_i^{k+1}\right) = w_{i,j}^{k+1}\frac{\partial a_j^k}{\partial n_j^k}$$

$$= w_{i,j}^{k+1}\frac{\partial f^k\left(n_j^k\right)}{\partial n_j^k} = w_{i,j}^{k+1}\dot{f}^k\left(n_j^k\right) \qquad (2.31)$$

En transposant (2.31) dans (2.30), l'on peut se rendre compte à l'évidence que les expressions des éléments non diagonaux seront tous nuls. Notons que (2.31) sous sa forme matricielle se note comme suit :

$$\frac{\partial n^{k+1}}{\partial n^k} = W^{k+1}\dot{F}^{k+1}(n^k) \qquad (2.32)$$

Au vu donc de ce qui précède, (2.30) s'écrira donc de la façon suivante :

$$\dot{F}^k(n^k) = \begin{bmatrix} \dot{f}^k\left(n_1^k\right) & 0 & \cdots & 0 \\ 0 & \dot{f}^k\left(n_2^k\right) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{f}^k\left(n_{s^k}^k\right) \end{bmatrix} \qquad (2.33)$$

Nous pouvons déjà au regard de ce qui précède et une fois de plus en utilisant le chaînage des dérivés extraire l'expression de récurrence des sensibilités comme suit :

$$s^k = \frac{\partial \hat{F}}{\partial n^k} = \left(\frac{\partial n^{k+1}}{\partial n^k}\right)^T \frac{\partial \hat{F}}{\partial n^{k+1}} = \dot{F}^k(n^k)(W^{k+1})^T \frac{\partial \hat{F}}{\partial n^{k+1}}$$

$$= \dot{F}^k(n^k)\left(W^{k+1}\right)^T S^{k+1} \qquad (2.34)$$

(2. 34) illustre bien le fait que les sensibilités sont propagées des sorties vers les entrées tel que :

$$s^M \rightarrow s^{M-1} \rightarrow \cdots\cdots\cdots \rightarrow s^2 \rightarrow s^1$$

Evaluons donc à présent s^M qui est l'expression de base pour un neurone *i* afin d'achever cette formule de récurrence.

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (d-a^M)^T (d-a^M)}{\partial n_i^M}$$

$$= \frac{\partial}{\partial n_i^M}\left(\sum_{l=1}^{s^M} (d_l - a_l^M)^2\right)$$

$$= -2(d_i - a^M)\frac{\partial a_i^M}{\partial n_i^M}$$

$$s_i^M = -2(d_i - a^M)\dot{f}^M\left(n_i^M\right) \qquad (2.35)$$

L'équation (2.35) est en définitive celle qui permet d'évaluer les sensibilités de chaque couche en parcourant le réseau de la sortie vers les entrées. Elle constitue la substance de la méthode d'apprentissage par rétro propagation qui sous sa forme matricielle se note comme suit :

$$s^M = -2\dot{F}^M(n^M)(d - a^M) \qquad (2.36)$$

2.7.1. Algorithme de la méthode de rétro-propagation du gradient

L'on peut illustrer l'algorithme de la RPG par les cinq étapes ci-dessous :

- Étape 1

Initialiser tous les poids du réseau à de petites valeurs aléatoires

- Étape 2

Pour chaque association d'entrée/sortie (p_q, d_q) de la base d'apprentissage ;

- Propager les entrées vers l'avant à travers les couches du réseau tel que

$$a^0 = p_q$$

$$a^k = f^k(W^k a^{k-1} - b^k) \qquad \text{Pour k = } 1,.....M$$

- Rétro propager les sensibilités vers l'arrière via les couches du réseau par :

$$s^M = -2\dot{F}^M(n^M)\left(d_q - a^M\right)$$

$$s^k = \dot{F}^k\left(n^k\right)\left(W^{k+1}\right)^T S^{k+1} \qquad \text{Pour } k = M\text{-}1, \ldots, 1$$

- Mise à jour des poids et des biais

$$\Delta W^k = -\eta s^k(t)(a^{k-1})^T \qquad \text{Pour k = } 1,.....M$$

$$\Delta b^k = \eta s^k \qquad \text{Pour k = } 1,.....M$$

- Étape 3

Si le critère d'arrêt fixé a été atteint, l'on stoppe l'apprentissage

- Étape 4

Si le critère d'arrêt fixé n'a pas encore été atteint, on permute l'ordre de présentation du couple d'entrée/sortie de la base d'apprentissage

- Étape 5

Retour à l'étape 2 pour la prochaine itération

2.7.2 Limite de la méthode de RPG

La formule 2.35 permet d'évaluer la sensibilité des couches à la propagation du gradient. Dans le cas d'un réseau profond possédant un nombre très élevé de couches dans son architecture (valeur de *M* très grande), la sensibilité des couches diminue et limite ainsi la capacité d'apprentissage du réseau. Ce phénomène est connu dans la littérature sous le vocable anglais *Vanishing gradient.* Pour résoudre ce problème, un autre type d'apprentissage adapté aux architectures profondes appelé *Deep Learning* a vu le jour.

2.8 *Deep Learning*

C'est une technique d'apprentissage basée sur plusieurs couches de représentation de très grands jeux de données avec des transformations successives qui amplifient les caractéristiques des données en entrée, les discriminent tout en atténuant leurs variations.

Le *deep learning* est basé sur l'idée des réseaux de neurones profonds et il est taillé pour gérer des données massives en ajoutant des couches au réseau. Un modèle de *deep learning* a la capacité d'extraire des caractéristiques à partir des données brutes grâce aux multiples couches de traitement. Ces couches effectuent plusieurs transformations

Linéaires et non linéaires afin d'apprendre progressivement sur les caractéristiques des données avec une intervention humaine minime.

Les algorithmes de *deep learning* sont des algorithmes d'optimisation. Ils ont pour objectif de minimiser au moyen des méthodes d'optimisation non linéaire une fonction de coût (fonction d'erreur) qui constitue une mesure de l'écart entre les sorties réelles du RNA profond et les sorties désirées. L'optimisation est réalisée au moyen des méthodes itératives, par modification des poids en fonction de l'évolution du gradient de la fonction d'erreur évalué par la méthode de RPG. Le résultat est par la suite exploité par l'algorithme d'optimisation choisi. Ce procédé permet d'obtenir un compromis satisfaisant entre la précision de l'approximation sur l'ensemble d'apprentissage et la précision de l'approximation sur l'ensemble de validation distinct du précédent. Puisque nous travaillons dans le cadre de l'apprentissage supervisé, notons que l'objectif ici est d'éviter un sur-apprentissage ou un sous-apprentissage du réseau car dans les deux cas de figure, les sorties du réseau sont très éloignées de la marge fixée. Il existe des méthodes d'optimisation avec un taux d'apprentissage constant et des méthodes à taux d'apprentissage adaptatif. Nous choisissons donc les méthodes avec un taux d'apprentissage adaptatif car elles convergent assez rapidement et fonctionnent aussi bien avec des bases d'apprentissage constituées des données éparses.

2.8.1 Algorithmes d'optimisation de *Deep Learning*

Nous présentons ici quelques algorithmes d'optimisation de la descente du gradient utilisant un taux d'apprentissage adaptatif tel qu'*Adam, RMSprop* et *AdaGrad.*

2.8.1.1 *AdaGrad*

« C'est un algorithme basé sur la descente du gradient qui adapte automatiquement le taux d'apprentissage aux paramètres du réseau en effectuant plus de mises à jour (MAJ) sur les caractéristiques moins fréquentes et moins de mise à jour sur les caractéristiques plus fréquentes » (Duchi et *al*., 2011). Par la suite, il effectue une mise à échelle de la MAJ au moyen de l'inverse de la racine carré de la somme des gradients, puis une accumulation du carré des gradients dans une variable *r*. Il s'en suit une stabilisation de la valeur numérique de mise à l'échelle par une constante δ et enfin une MAJ du paramètre caractéristique de la fonction de coût (Ludovic Trottier., 2020). Les étapes ci-dessous résument son fonctionnement d'après (Ludovic Trottier., 2020).

- Étape 1

Présentation de l'échantillon d'apprentissage $(x^{(i)}, y^{(i)})$ de taille *m*

- Étape 2

Evaluation de la somme des gradients pour chaque paramètre de la fonction de coût

$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\theta} L(f(x^{(i)}, \theta), y^{(i)})$$

- Étape 3

Accumulation du carré des gradients dans la variable *r*

$$r \leftarrow r + \hat{g} \odot \hat{g}$$

- Étape 4

Mise à l'échelle et stabilisation du vecteur directionnel *d* du gradient

$$d = \frac{\eta}{\delta + \sqrt{r}} \odot \hat{g}$$

- Étape 5

Mise à jour du paramètre de la fonction de coût

$$\theta \leftarrow \theta - d$$

Adagrad est utilisé pour l'apprentissage des grands réseaux de neurones chez Google qui ont appris à reconnaître les chats dans les vidéos YouTube. Il présente néanmoins un inconvénient lié à la croissance excessive de *r* qui peut être résolu par *RMSprop*. Le symbole $\odot$ est celui de la multiplication terme à terme (produit matriciel de Hadamard).

2.8.1.2 *RMSprop*

Cet algorithme d'optimisation est basé sur l'idée selon laquelle « au lieu d'accumuler tous les carrés des gradients précédents, on restreint la fenêtre des gradients accumulés à une taille fixée » (Hinton et *al.,* 2012). Les étapes suivantes décrivent son fonctionnement (Ludovic Trottier., 2020).

- Étape 1

Présentation de l'échantillon d'apprentissage $(x^{(i)}, y^{(i)})$ de taille *m*

- Étape 2

Evaluation de la somme des gradients pour chaque paramètre de la fonction de coût

$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(f(x^{(i)}, \theta), y^{(i)})$$

- Étape 3

Accumulation du carré des gradients récents en y insérant une moyenne mobile exponentielle ρ dont le taux de croissance est compris dans l'intervalle [0,1]

$$r \leftarrow \rho * r + (1 - \rho) * \hat{g} \odot \hat{g}$$

- Étape 4

Mise à l'échelle et stabilisation du vecteur directionnel *d* du gradient

$$d = \frac{\eta}{\delta + \sqrt{r}} \odot \hat{g}$$

- Étape 5

Mise à jour du paramètre de la fonction de coût

$$\theta \leftarrow \theta - d$$

La valeur suggérée pour ρ est de 0,99. L'inconvénient de cet algorithme est lié au fait que r soit une estimation du second moment (variance décentrée) du gradient $\hat{g}$ et possède un large biais positif au début de la descente du gradient. L'algorithme d'optimisation Adam vise dont à corriger cela.

2.8.1.3 *ADAM*

L'algorithme d'optimisation *ADAM* (*Adaptative Moments*) fonde son principe sur l'idée selon laquelle « en plus de stocker une moyenne décroissante exponentielle des précédents carrés des gradients *r* comme *RMSprop,* l'on doit conserver aussi la moyenne décroissante exponentielle des précédents gradients » (Kingma et Ba., 2014). Sa méthode consiste donc à calculer une estimation du premier et second moment avec une moyenne mobile exponentielle à taux $\rho_1 \, et \, \rho_2$ ($\rho_1 = 0{,}9 \, et \, \rho_2 = 0{,}99$) puis de corriger les biais des moments sachant que le premier moment normalisé par le second moment donne la direction de la MAJ (Ludovic Trottier., 2020).

- Étape 1

Présentation de l'échantillon d'apprentissage $(x^{(i)}, y^{(i)})$ de taille *m*

- Étape 2

Evaluation de la somme des gradients pour chaque paramètre de la fonction de coût

$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\theta} L(f(x^{(i)}, \theta), y^{(i)})$$

- Étape 3

Evaluation du premier moment

$$s = \rho_1 * s + (1 - \rho_1) * \hat{g}$$

- Étape 4

Evaluation du second moment

$$r \leftarrow \rho_2 * r + (1 - \rho_2) * \hat{g} \odot \hat{g}$$

- Étape 5

Correction biais du premier moment

$$\hat{s} \leftarrow \frac{s}{1-(\rho_1)^t}$$

- Étape 6

Correction biais 2e moment

$$\hat{r} \leftarrow \frac{r}{1-(\rho_2)^t}$$

- Étape 7

Mise à l'échelle et stabilisation du vecteur directionnel *d* du gradient

$$d = \frac{s}{\delta + \sqrt{\hat{r}}} * \eta$$

➢ Étape 8

Mise à jour du paramètre de la fonction de coût

$$\theta \leftarrow \theta - d$$

L'utilisation des algorithmes d'optimisation ci-dessus permet d'obtenir un meilleur apprentissage du RNA profond au prix d'un bon compromis temps/efficacité. L'appréciation de la performance d'un RNA à l'issue d'une phase d'apprentissage se fait au moyen de certaines métriques bien précises qui seront présentées dans le prochain paragraphe.

2.9 Métrique d'évaluation des RNA

Plusieurs métriques d'évaluation des RNA existent selon que le RNA soit dédié à une tâche de classification ou de régression. Nous nous contenterons dans ce paragraphe de présenter celles qui sont utilisées pour les tâches de régression.

2.9.1 *MAE*

Elle permet d'évaluer l'écart entre ce que le modèle de RNA prédit et ce qui est attendu. Si Y et $\bar{Y}$ sont respectivement les valeurs prédites par le réseau et réelles pour N échantillons testés, elle cette métrique se calcule par la formule :

$$MAE = \frac{1}{N}\sum_{j=1}^{N}\left|Y_j - \bar{Y}\right| \quad (2.37)$$

2.9.2 *RMSE*

A la différence de la *MAE*, elle évalue la moyenne des carrés de l'écart entre ce que le modèle de RNA prédit et ce qui est attendu.

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(Y_j - \bar{Y})^2} \tag{2.38}$$

2.10 Synthèse des architectures

Nous avons présenté quelques architectures de RNA les plus utilisées. Il ressort de cette présentation que les architectures profondes sont les mieux adaptées pour la manipulation des données massives du fait des unités de calcul de plus en plus puissantes contrairement aux architectures classiques dont les limites ont été présentées dans le paragraphe 2.4. Parmi ces architectures profondes, le *CNN* s'est avéré être très utilisé dans le domaine du traitement ou classification des images tandis que les architectures tel que *RNN*, *LSTM* proposées par « Hochreiter et Schmidhuber » en 1997 ainsi que le *GRU* à travers leurs principes de fonctionnement se positionnent comme les meilleures architectures de prédiction des séries temporelles.

Tableau 2.2 : Caractéristiques de 3 catégories de RNA

Type de RNA	Caractéristiques fondamentale	Données en entrée
MLP	1- Comprend plusieurs couches constituées d'un nombre variable de neurone. 2- propagation directe de l'information à travers le réseau. 3- les couches intermédiaires relient les entrées aux sorties.	-Sous forme de tableau. -Taille limitée.
RNN	1- propagation avant et arrière de l'information 2- l'interconnexion entre les neurones du réseau présente au moins un cycle	-Taille variable. -Séquentielle. -Langage naturel.
CNN	Combinaison de plusieurs couches où chacune représente une des fonctionnalités du réseau	-Image et audio. -Matricielles. -Possédant des liens spatiaux.

Dans les paragraphes suivants, nous essaierons d'analyser profondément l'architecture de type *RNN* car elle est destinée au traitement des données de type séquentielles.

2.11 Réseaux de neurones récurrents ou *RNN*

L'on sait que le cerveau humain ne commence pas sa pensée à partir de zéro à chaque instant. Lorsqu'on lit un ouvrage, on comprend chaque phrase ou paragraphe en fonction des phrases ou des paragraphes précédents parce que nos pensées ont une certaine persistance. Les architectures classiques de RNA ne peuvent pas le faire et cela

est un inconvénient majeur. Les *RNNs* traitent ce problème. Ce sont des réseaux avec des boucles, permettant aux informations de persister. Les *RNNs* utilisent des informations séquentielles. Dans une architecture neuronale classique, nous supposons que toutes les entrées (et les sorties) sont indépendantes les unes des autres. Cette supposition n'est pas toujours vérifiée pour de nombreuses tâches. Si on veut par exemple prédire le prochain mot dans une phrase, il faut absolument connaître les mots qui sont venus avant. Cette réalité met en évidence la dépendance qui existe entre les entrées (ou les sorties) pour une telle tâche. Elle s'apparente donc à une relation de récurrence entre les entrées (ou les sorties). Les *RNNs* sont appelés récurrents car ils exécutent la même tâche pour chaque élément d'une séquence, la sortie étant dépendante des calculs précédents. Une autre façon de penser les *RNNs* est de dire qu'ils ont une « mémoire » qui capture l'information sur ce qui a été calculé jusqu'ici. En théorie, les *RNNs* peuvent utiliser des informations dans des séquences arbitrairement longues, mais dans la pratique, on les limite à regarder seulement quelques étapes en arrière.

2.11.1 Principe de fonctionnement d'un RNN

Un RNN est comme un MLP dans lequel on rajoute des liens pour donner en entrée d'une couche du réseau sa propre sortie au pas de temps précédent en plus de la sortie courante de la couche précédente. Dans le cas où les vecteurs d'entrées sont indépendants les uns des autres, cela n'a pas beaucoup d'intérêt mais s'ils sont sous forme de séquences temporelles ou spatiales, cette modification aura un très grand impact. En général, une seule couche ne suffit pas pour capter toute l'information contenue dans les séquences. Dans ce cas, l'on empile plusieurs couches de type *RNN* afin d'avoir un réseau plus profond. Ce procédé favorise l'extraction des informations plus complexes en entrée, garantissant ainsi une meilleure modélisation des données. Dans la plupart des cas, deux à cinq couches sont nécessaires puisqu'au-delà, il devient

difficile d'obtenir la convergence du réseau. Un *RNN* applique donc une fonction à une séquence d'entrée$[X_1, X_2, \dots, X_T]$, pour produire une séquence de sortie $[Y_1, Y_2, \dots, Y_T]$ en maintenant un état interne$[H_1, H_2, \dots, H_T]$.

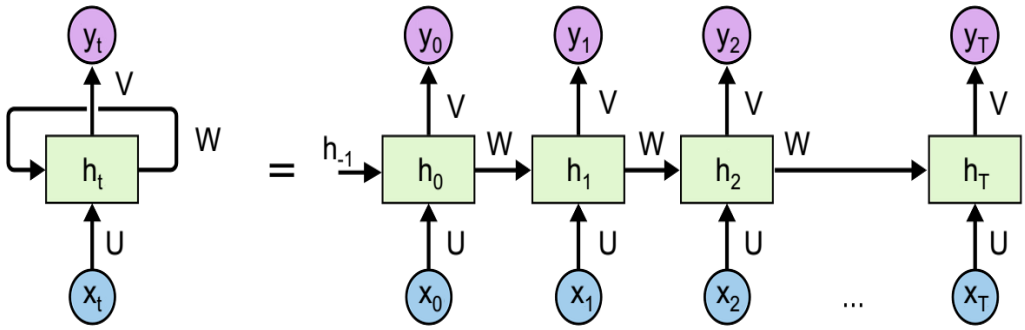

Figure 2.19: Forme compact et dépliée d'un RNN. *(http://colah.github.io).*

U, V, W sont les paramètres du réseau partagés à travers le temps.

Une couche RNN est une succession cellules possédant chacune deux entrées. La première cellule qui n'a pas d'antécédent prend un vecteur initialisé aléatoirement et parcoure successivement les entrées X_0 à X_T. A l'instant *t*, la $t^{ième}$ cellule combine l'entrée courante X_t avec la prédiction au pas précédent h_{t-1} pour calculer une sortie h_t de taille R. Le dernier vecteur calculé h_T de taille R est la sortie finale de la couche RNN : on en déduit qu'une couche RNN définit donc une relation de récurrence de la forme $h_t = f(X_t, h_{t-1})$. $t = 0,1,2, \dots, T$

La $t^{ième}$ cellule n'est rien d'autre qu'une couche dense de taille R dont l'entrée est la concaténation de X_t et de h_{t-1} de taille R également. La fonction d'activation utilisée pour les cellules RNN est la tangente hyperbolique. Cette utilisation se justifie par sa capacité d'accélérer la convergence du réseau contrairement à la fonction sigmoïde. L'on évite ainsi que les sorties des neurones ne deviennent trop grands cars cela demanderait plus de temps de calculs et de mémoire d'ordinateur. La performance du réseau est donc régie par l'équation ci-dessous :

$$h_t = \tanh\left(Ux_t + Wh_{t-1}\right) \tag{2.39}$$

Celle de la sortie du réseau est régie par l'équation suivante :

$$y_t = f(Vh_t) \tag{2.40}$$

2.11.2 Évaluation de l'erreur globale de propagation dans le réseau RNN

Considérons le schéma de la figure 2.20 (forme dépliée) légèrement modifié où les E_i représentent les erreurs générées par le réseau pour chaque sortie Y_i.

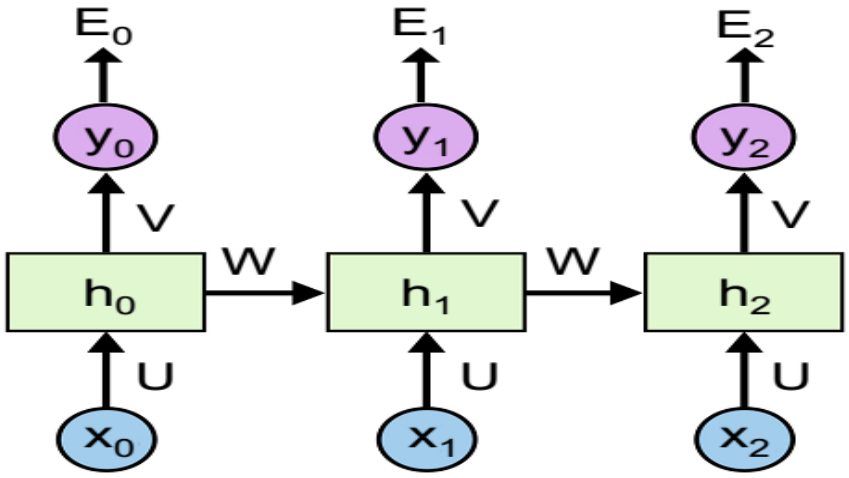

Figure 2.20: Illustration de l'erreur générée à chaque sortie *(Hinton et al., 2012)*

L'erreur globale qui se propage à travers le réseau au bout d'un certain temps s'exprime comme suit :

$$E = \sum_{t=0}^{T} E_t \tag{2.41}$$

L'existence de cette erreur traduit le fait que les paramètres du réseau (U, V, W) ne soient pas parfaitement à jour pour obtenir la sortie souhaitée. Elle s'estime en évaluant sa variation par rapport aux dis paramètres du réseau. Elle sera évaluée par rapport au paramètre U et l'expression obtenue sera généralisée pour les deux autres paramètres. Puisque cette propagation s'effectue de la droite vers la gauche, nous l'illustrerons à travers quelques schémas (César Laurent, 2017).

$$\frac{\partial E}{\partial U} = \sum_{t=0}^{t=T} \frac{\partial E_t}{\partial U} \qquad (2.42)$$

Evaluons tout d'abord la propagation de cette erreur sur la dernière cellule à droite. Cela consiste à la propager jusqu'à son état interne H_2 que l'on peut illustrer par la figure ci-dessous :

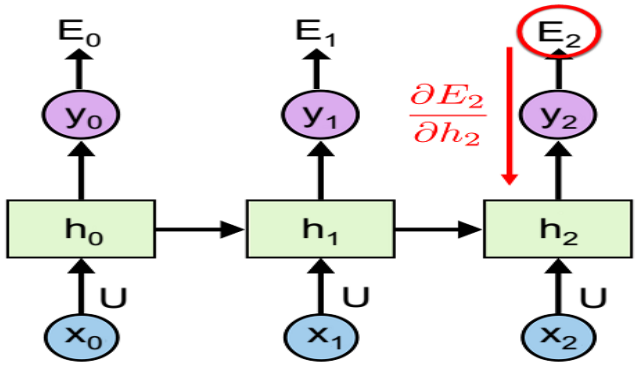

Figure 2.21: Propagation de l'erreur à travers l'état interne H2 (*César Laurent, 2017).*

Il en ressort de cette figure que $\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2}$ ...

L'erreur étant ensuite propagée sur U qui est un paramètre pondérant l'entrée X_2 au deuxième pas de temps.

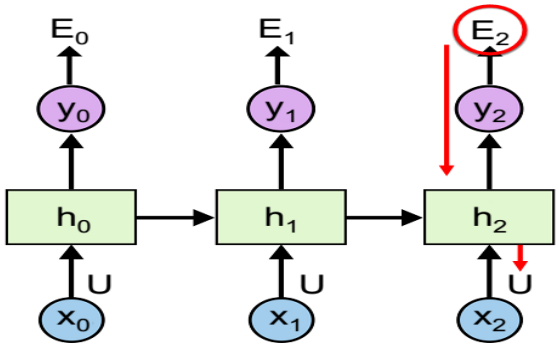

Figure 2.22: Propagation de l'erreur générée à travers U. (*César Laurent, 2017).*

D'après la figure 2.22, l'on peut écrire $\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial H_2}(X_2^T \ldots)$ (où … représente la suite de l'expression de la dérivée). Étant donné que U intervient également dans le calcul de H_0 et H_1 d'après la formule 2.39, il faut donc propager l'erreur à travers le temps tel comme l'illustre la figure 2.24 :

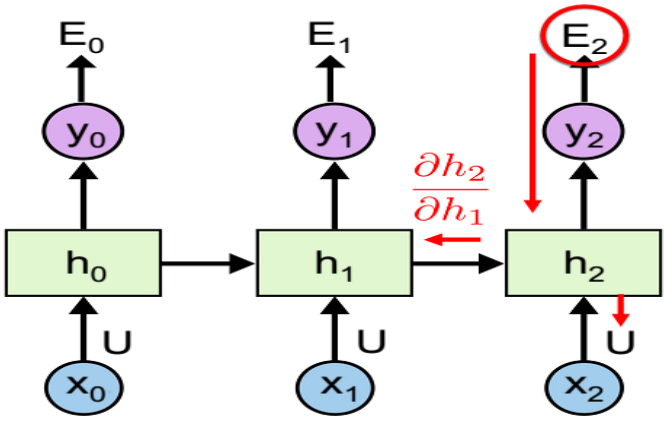

Figure 2.23: Propagation de l'erreur générée à travers H1 (*César Laurent.,2017).*.

Ce qui permet d'écrire que $\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial H_2}(X_2^T + \frac{\partial h_2}{\partial h_1}(\ldots$

L'erreur est ensuite propagée sur U qui pondère l'entrée X_1 .

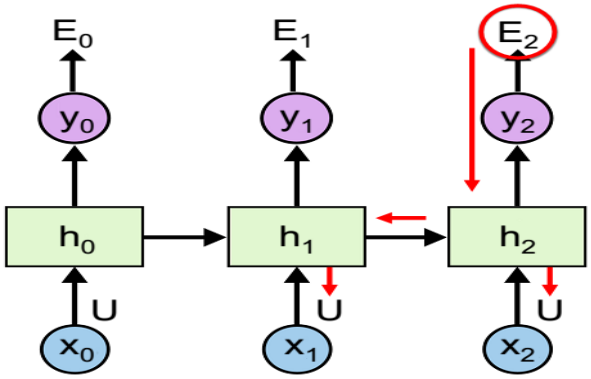

Figure 2.24: Propagation de l'erreur générée à travers U (*César Laurent.,2017).*

Permettant ainsi d'écrire que $\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial H_2}(X_2^T + \frac{\partial h_2}{\partial h_1}(X_1^T \ldots$

Dans la même logique, celle-ci se propage aussi sur H_0 comme le montre les flèches rouges de la figure 2.25.

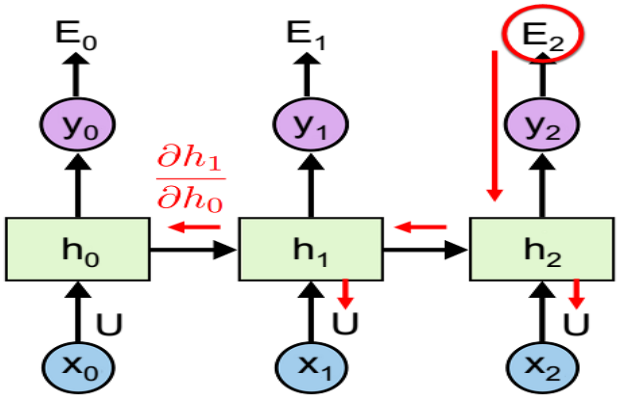

Figure 2.25: propagation de l'erreur générée à travers H0 (*César Laurent.,2017).*

D'où l'expression $\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial H_2}(X_2^T + \frac{\partial h_2}{\partial h_1}(X_1^T + \frac{\partial h_1}{\partial h_0} \ldots$

L'erreur E_2 générée par la sortie Y_2 achève sa propagation à travers U qui pondère X_0 tel qu'il suit :

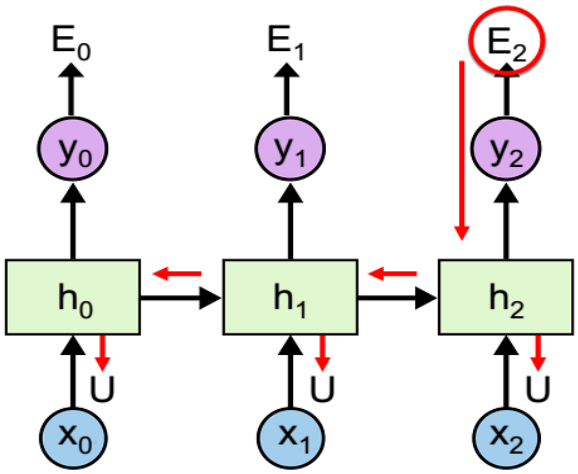

Figure 2.26: Propagation de l'erreur générée à travers U (*César Laurent, 2017).*

Cela permet d'obtenir l'expression partielle de l'erreur globale générée par la dernière sortie et propagée à travers le réseau durant le temps T tel que :

$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial H_2}\left(X_2^T + \frac{\partial h_2}{\partial h_1}\left(X_1^T + \frac{\partial h_1}{\partial h_0} X_0^T\right)\right) \qquad (2.43)$$

De la même manière on évalue les gradients sur les autres paramètres que l'on somme par la suite pour obtenir l'expression finale de l'erreur globale tel que précisé plus haut.

2.11.3 Inconvénients des *RNNs*

Il a été établi au paragraphe 2.11.2 que l'évaluation de l'effet des variations des paramètres du réseau sur la fonction d'erreur doit prendre en compte tous les changements de valeurs aux différents pas de temps. Cette exigence constitue la difficulté de cette évaluation. Il existe d'autres inconvénients propres aux *RNNs* que nous présentons dans les paragraphes qui suivent.

2.11.3.1 Architecture à courte mémoire

Chaque cellule prend des données en entrée qui sont transformées par la fonction tangente hyperbolique. En effet, après un premier passage par cette fonction, les valeurs se situent dans l'intervalle$]-1,1[$, après un deuxième passage, l'on se retrouve dans l'intervalle$]-0.76,0.76[$, …. Il en ressort que les données qui passent successivement par cette fonction tendront vers 0 après un certain nombre d'itérations. Si nous reconsidérons la séquence d'entrée$[X_1, X_2, \dots, X_T]$, d'après le principe de fonctionnement d'un RNN, X_1 passera T fois à travers la fonction tangente hyperbolique tandis que X_T ne passera qu'une seule fois. Comme conséquence, l'influence de l'information véhiculée par X_1 devient insignifiante dans la tâche de prédiction devant celle véhiculée par X_T. Ce phénomène met en exergue l'incapacité de cette architecture à mémoriser une grande quantité d'information lorsque les séquences en entrée sont longues.

2.11.3.2 Explosion des gradients

En fonction des valeurs des poids ainsi que la fonction d'activation choisie, l'on peut obtenir une valeur du gradient supérieure à 1. En effet, si les poids initiaux ont une grande valeur, le calcul du gradient de l'erreur sur les premières couches du réseau mettra en jeu le produit des termes de grande valeur, ce qui engendrera d'autres valeurs plus élevées au fil du temps. C'est l'une des raisons de l'explosion du gradient.

2.11.3.3 Disparition du gradient

La principale difficulté inhérente à l'entraînement d'un réseau de type *RNN* est le problème de disparition du gradient ou *vanishing gradient*. L'entraînement du réseau se résume aux étapes suivantes : initialisation aléatoire des poids (U, V, W), évaluation de la fonction de coût L (différence entre la grandeur réelle et la grandeur prédite),

évaluation du gradient de L par rapport aux poids, mise à jour des poids pour la minimisation de la fonction de coût en effectuant la descente du gradient calculée à l'étape précédente. D'après la figure 2.20 et la formule 2.39, l'on peut écrire :

$$h_2 = f\big(W(X_2, h_1)\big) = f(W(X_2, f(W(X_1, h_0)) \tag{2.44}$$

La formule ci-dessus nous indique que le gradient de h_2 par rapport à W sera proportionnel aux dérivées de la fonction tanh qui prend des valeurs dans$[0,1[$. Il est évident que plus l'on multiplie les valeurs entre 0 et 1 entre elles, plus le résultat se rapproche de 0. A l'observation des formules (2.43), il est évident que l'évaluation de $\frac{dL}{dW}$ prendra des valeurs très petites pour des séquences longues. Ceci ralentit la mise à jour des paramètres du réseau et n'assure plus la convergence de ce dernier.

Plusieurs solutions ont été proposées pour remédier à ces défauts des RNNs. Parmi celle-ci, l'on peut citer l'utilisation de la fonction d'activation *ReLu* en lieu et place de TanH, l'usage du *gradient clipping* (Pascanu et *al., 2013*), l'utilisation de la *BPTT* tronquée (Tallec et Ollivier., 2018) ainsi que le réseau *LSTM* (S. Hochreiter et J. Schmidhuber). Dans ce travail, nous nous intéressons à la solution qui fait usage du réseau *LSTM*.

2.12 Réseau de type *LSTM*

L'objectif principal des *LSTM* est de scinder le signal entre ce qui est important à court terme via le *hidden state* et ce qui le sera à long terme à travers le *cell state*. Une cellule *LSTM* est principalement composée de quatre couches *RNN*. Les trois premières fonctionnent comme les portes logiques et la dernière permet d'approvisionner l'état interne en informations issues des portes d'entrées. Le fonctionnement des 3 premières couches identique à celui des portes logiques permet de contrôler exclusivement :

- Le flux d'information qui entre dans la cellule *LSTM*,
- Le flux d'information retenue par l'état interne de la cellule à chaque pas de temps,
- Le flux d'information qui sort de la cellule *LSTM*.

2.12.1 Principe de fonctionnement du réseau *LSTM*

En observant la figure 2.28 ci-dessous, l'on constate que, tout comme le *RNN*, le *LSTM* définit lui aussi une relation de récurrence tout en utilisant une variable supplémentaire qui est la *Cell state C*.

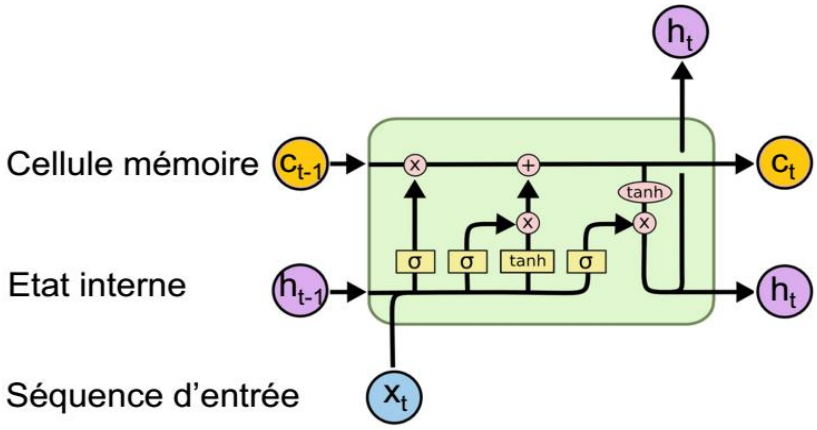

Figure 2.27: Forme compact d'un *LSTM*. *(http://colah.github.io)*.

$$(h_t, c_t) = f(X_t, h_{t-1}, c_{t-1}) \qquad (2.45)$$

Ce réseau est capable de supprimer ou de rajouter une information selon le besoin à travers la *Cell state* ou état cellulaire qui constitue la mémoire de cette architecture. Nous expliquerons son fonctionnement en quatre étapes en nous focalisant sur cette *Cell state* qui est le canal privilégié de transit de l'information sur une séquence. Elle est mise à jour de façon additive durant chaque étape sans passer par une quelconque

fonction d'activation (Figure 2.28) afin d'éviter le problème de *vanishing gradient* mentionné plus haut.

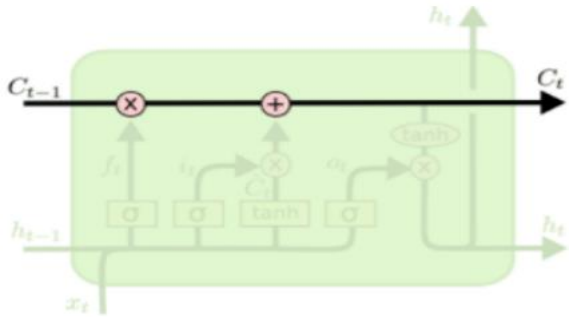

Figure 2.28: Transit de l'information via la Cell state. *(http://colah.github.io).*

2.12.1.1 Sauvegarde ou destruction de l'information dans la cellule

À travers les entrées h_{t-1} et X_t , la *forget gate layer* qui est une couche dense avec une activation de type sigmoïde produit un vecteur f_t dont les valeurs sont comprises entre 0 et 1. Une valeur égale à 1 signifie que l'information sera conservée tandis que 0 signifie destruction de l'information. La *forget gate* permet donc d'oublier ce qui est dans la mémoire. Une multiplication terme à terme s'effectue entre f_t et c_{t-1} avec pour résultat l'annulation des composants de c_{t-1} dont les homologues côtés f_t sont proches de 0 permettant ainsi d'obtenir un *cell state* filtrée. L'état du vecteur f_t est régit par l'équation 2.46 et présenté par la figure 2.29.

$$f_t = \sigma(U_f X_t + W_f h_{t-1} + b_f) \qquad (2.46)$$

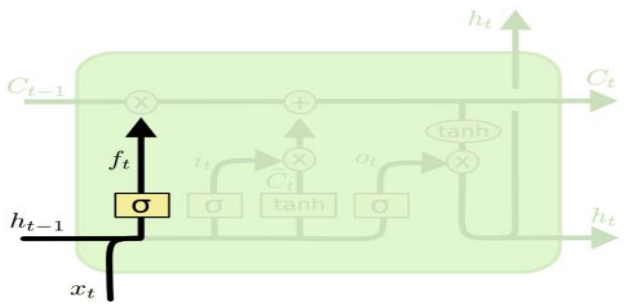

Figure 2.29: Transit de l'information via la *Forget gate. (http://colah.github.io).*

2.12.1.2 Première mise à jour de l'information dans la *cell state*

L'*input gate layer* décide de la valeur à mettre à jour dans la cellule à travers son filtre i_t dont les valeurs comprises entre 0 et 1 sont obtenues similairement au *forget gate*. Un vecteur candidat $\tilde{C}_t$ pour la mise à jour de la cell state est créé au moyen d'une couche tanh génératrice des valeurs possibles pouvant figurer dans la *cell state.* Le candidat $\tilde{C}_t$ est par la suite filtré par l'*input gate* i_t via une multiplication terme à terme permettant d'obtenir une mise à jour de la *cell state*. Cette deuxième étape se résume par les équations (2.47) et (2.48) et est schématisé par la figure 2.30.

$$i_t = \sigma(U_i X_t + W_I h_{t-1} + b_i) \tag{2.47}$$

$$\tilde{C}_t = \tanh\left(U_g X_t + W_G h_{t-1} + b_g\right) \tag{2.48}$$

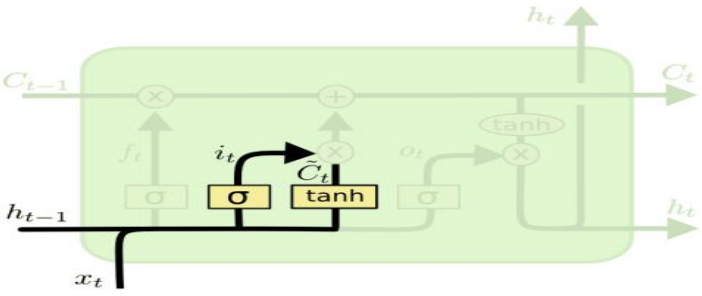

Figure 2.30: Mise à jour de l'information dans la *cell state. (http://colah.github.io).*

2.12.1.3 Deuxième mise à jour de l'information dans la cell state

La *cell state* filtrée obtenue à l'étape 1 est mise à jour grâce au vecteur candidat $\tilde{C}_t$ filtré obtenu à l'étape précédente puisque l'input gate permet d'ajouter l'information dans la cellule et la *forget gate* permet d'oublier l'information déjà présente dans la cellule. La mise à jour définitive devient tout simplement une addition terme à terme de ces deux vecteurs permettant ainsi d'obtenir le nouveau *cell state* c_t au moyen de l'équation suivante :

$$C_t = i_t \odot \tilde{C}_t + f_t \odot C_{t-1} \tag{2.49}$$

Chacune des portes d'entrée n'a accès qu'à la composante des états internes sur laquelle elle a une influence.

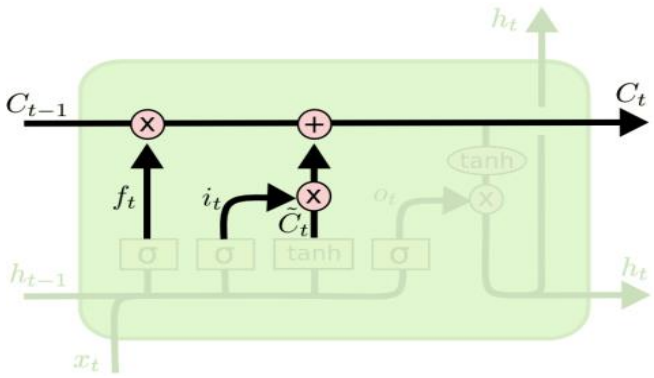

Figure 2.31: Mise à jour définitive dans la *cell state. (http://colah.github.io).*

2.12.1.4 Décision sur la valeur à mettre en sortie du réseau

L'*output gate* produit un vecteur filtre dont les valeurs sont comprises entre 0 et 1. Les valeurs de la *cell state* calculées à l'étape 3 sont ramenées à l'intervalle $]-1,1[$ par la fonction d'activation tanh et il s'ensuit un filtrage de l'*output gate* O_t pour obtenir la sortie finale du réseau qui est h_t. La figure 2.32 schématise cette décision.

$$O_t = \sigma(U_O X_t + W_O h_{t-1} + b_O) \quad (2.50)$$

$$h_t = O_t \odot \tanh(C_t) \quad (2.51)$$

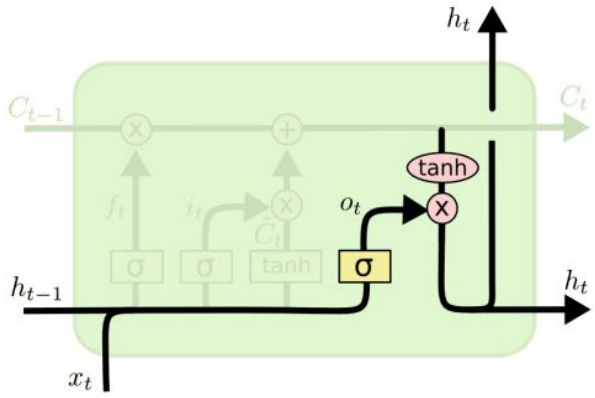

Figure 2.32: Illustration du calcul de la valeur de sortie du réseau. (http://colah.github.io).

En définitive, c'est grâce à la mémoire *cell state* que le *LSTM* compense la perte d'information causée par le *vanishing gradient* dans le *RNN* permettant ainsi de conserver les informations essentielles pour faire des prédictions.

$i(t), f(t), h(t)\ et\ o(t)$ Sont respectivement les vecteurs activant la :

- Porte d'entrée contrôlant la quantité d'information qui entre dans la couche du *LSTM* ;
- Porte d'oubli qui contrôle la quantité d'information à mémorisée,
- États internes de la couche *LSTM* ;
- Porte de sortie qui contrôle la quantité d'information qui sort de la couche *LSTM*.

Il est à noter que les *LSTM* utilisent pour l'apprentissage une version légèrement différente de la RPG qui est la rétro propagation dans le temps aussi connu sous l'appellation *Back Propagation Through Time* (*BPTT*) (Werbos., 1990). À la différence de la RPG, ils retro-propagent la contribution des gradients des liens récurrents.

2.13 Réseau de type *Gated Recurrent Units* (*GRU*)

Le *GRU* a été introduit par Cho et al. 2014 avec l'idée d'apprendre les dépendances à long terme d'une séquence en utilisant un mécanisme de portes proche de celui du *LSTM*. Il est constitué de deux portes, une porte de réinitialisation r, et une porte de mise-à-jour z. La porte de réinitialisation détermine comment combiner la nouvelle entrée avec la mémoire précédente et la porte de mise à jour définit la quantité de mémoire précédente à conserver. Si nous définissons la réinitialisation à tous les 1 et la mise à jour de la porte à tous les 0, nous arrivons à nouveau à notre modèle *RNN* simple. La différence avec les *GRU* peut se résumer en ce qui suit :

- Un *GRU* possède deux portes tandis qu'un *LSTM* en a trois
- Un *GRU* ne possède pas la mémoire interne (C_t) différente de l'état caché exposé. Ils n'ont pas la porte de sortie qui est présente dans le *LSTM*.
- L'*input gate* et la *forget gate* sont couplés par une porte de mise à jour z et la porte de réinitialisation r est appliquée directement à l'état caché précédent.

Ainsi, la responsabilité de la porte de réinitialisation dans un *LSTM* est vraiment divisée en r (*reset gate*) et z (*update gate*). La porte de réinitialisation r détermine la quantité d'informations à oublier tandis que la porte de mise à jour Z définit la quantité d'informations à conserver.

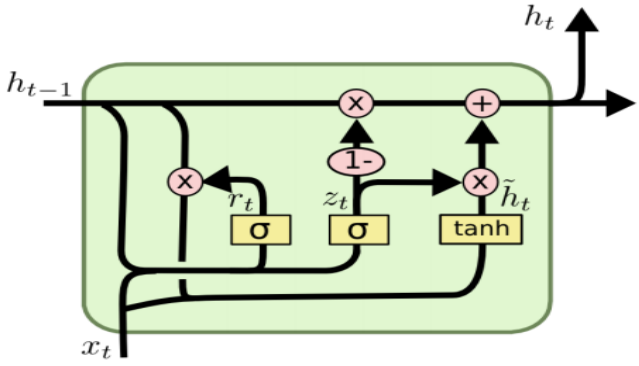

Figure 2.33: Présentation d'un GRU. *(http://colah.github.io).*

Les équations régissant le fonctionnement de ce réseau sont :

$$\begin{cases} Z_t = \sigma(U_Z X_t + W_z h_{t-1} + b_z) \\ r_t = \sigma(U_r X_t + W_r h_{t-1} + b_r) \\ g_t = tanH(U_g X_t + W_g(r_t \odot h_{t-1}) + b_g \\ h_t = Z_t \odot g_t + (1 - z_t) \odot h_{t-1} \end{cases} \quad (2.52)$$

Conclusion

Ce chapitre qui s'achève nous a permis de présenter les concepts généraux qui guident le fonctionnement des *RNA* classique et profond. Nous avons essayé de monter les différences qui existent entre ces différentes architectures de *RNA* tout en précisant leurs domaines d'utilisation. À l'observation de ces différences, il ressort qu'un RNA de type *LSTM* est indiqué pour la prédiction des évènements à court ou à long terme. Le principe de fonctionnement de cette architecture *LSTM* a également été présenté. Il ne nous reste plus qu'à simuler son fonctionnement et d'analyser ses résultats. Cette tâche fera l'objet du prochain chapitre.

CHAPITRE III

SIMULATION ET INTERPRÉTATION DES RÉSULTATS

L'occurrence des pannes sur le réseau de transport d'énergie se distingue par son caractère aléatoire. Cette réalité complique parfois la conception et le dimensionnement des dispositifs qui assurent cette fonction de transport. La fréquence de collecte des données liées à l'occurrence des pannes doit être connue car cela a un impact sur l'utilisation des dites données. Compte tenu de la difficulté liée à la collecte des données pour un réseau aussi étendu que celui du transport/distribution, il est recommandé de pouvoir générer des longues séries de données fiables. Il est assez difficile de développer un modèle assez précis de prédiction si les données d'entrée sont approximatives. Dans le chapitre précédent, nous avons décrit le principe de fonctionnement d'une architecture profonde de type *LSTM*. Dans ce chapitre, nous allons simuler le fonctionnement de cette architecture après avoir présenté le concept de séries temporelles attaché à notre base de données. Nous terminerons par une interprétation les résultats obtenus et une comparaison de ces derniers avec ceux fournis par d'autres outils d'apprentissage automatique. La *RMSE* présentée au chapitre 2 permet d'effectuer la comparaison des résultats.

3.1. Base de données

Pour effectuer les prédictions, l'on a besoin d'informations concernant le passé. On doit pouvoir supposer que certains comportements du passé continueront à se reproduire dans le futur. Cela s'appelle l'hypothèse de continuité (Steven C et al ,1998).

3.1.1 Notion de série temporelle

Une série temporelle est un ensemble de points de données qui s'étendent séquentiellement sur une période de temps suivant un ordre naturel. Etant donné que les puissances perdues par le réseau peuvent prendre des valeurs aléatoires en tout temps, les données attachées à celles-ci constituent une série temporelle de type continue. Nous nous intéressons ici à trois des quatre objectifs associés à l'analyse des séries temporelles selon (Chatfield, 2000). Ces objectifs sont :

- Description graphique des données issue des mesures statistiques ;
- Établissement du modèle statistique élucident la génération des données ;
- Estimation des valeurs futures (prédiction) ;
- Contrôle et anticipation du processus décrit par les données.

3.1.1 Données relatives aux pertes de puissance

Il a été dressé depuis plusieurs années par le centre des opérations urgentes de l'administration pour l'information sur l'énergie, des tableaux mensuels illustrant pour plusieurs localités du pays les dates d'occurrences des pannes, la cause des pannes (actes de vandalismes, tornade, tempêtes de neige, défaut sur transformateur, disfonctionnement des relais, vents violents, décrochage d'un générateur, défaut phase

terre, disfonctionnement des disjoncteurs, tremblement de terre, inadéquation entre le type de source de production et la charge, défaut sur les lignes de transport, etc.), ainsi que l'estimation en Méga Watt de la quantité de puissance perdue. Nous présentons ici sous forme de tableau à deux entrées l'estimation des puissances perdus tous les mois sur une période de 14 ans. Notons néanmoins que d'après la source, il s'agit des valeurs estimées et qui ne sauraient en aucun cas constituer une sorte de vérité absolue compte tenu des difficultés liées à la collecte desdites données. Notre base de données est extraite des données recueillis mensuellement par Emergency Operations Center Form EIA-417R sur une période allant de Janvier 2005 à Décembre 2018. Le lien pour y accéder est : « ***http//www.oe.netl .doe.gov/OE417_annual_summary.aspx***».

Tableau 3.1 : Récapitulatif des pertes de puissance (MW)

MOIS ANNEE	**JAN**	**FÉV**	**MAR**	**AVR**	**MAY**	**JUN**	**JUL**	**AOÛ**	**SEP**	**OCT**	**NOV**	**DÉC**
2005	227	402	180	190	509	109	171	366	305	281	191	292
2006	75	234	200	162	343	318	304	159	284	342	203	354
2007	380	353	137	132	235	435	275	118	81	374	100	275
2008	307	208	236	0	117	358	246	212	282	200	235	131
2009	352	233	235	200	302	318	163	308	0	180	715	400
2010	295	508	200	15	318	321	508	248	339	252	185	198
2011	848	253	301	279	345	301	258	326	388	125	0	176
2012	150	0	303	218	420	426	226	224	537	49	520	243
2013	101	140	462	80	157	515	450	327	302	115	195	264
2014	10	495	793	260	4	368	303	165	50	261	132	178
2015	0	610	67	10	275	350	234	305	150	209	500	135
2016	0	400	200	360	270	304	386	678	498	722	0	0
2017	100	280	857	86	378	170	645	100	910	460	80	450
2018	960	167	70	400	302	127	100	65	805	101	32	200

3.1.2 Évolution temporelle des données

Il s'agit d'une fonction x(t) dans laquelle t qui représente le temps constitue un indice et est compris entre 1 et n. L'horizon de prédiction constitue le nombre de points indicés

entre le présent et le futur. Dans ce travail, l'horizon de prédiction est compris entre un et plusieurs mois.

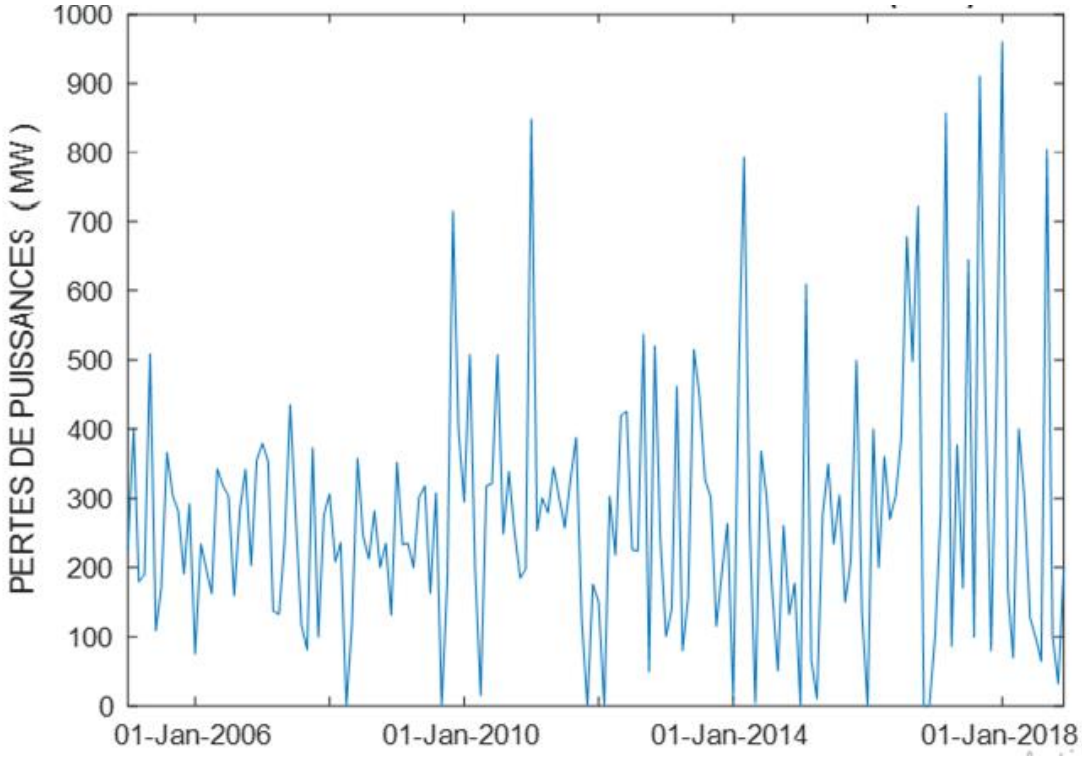

Figure 3.1: Représentation graphique des pertes de puissance

3.1.3 Gestion de la base de données

La répartition est faite d'une façon aléatoire, l'utilisateur choisit les pourcentages de répartition de l'échantillon de base en trois sous- ensembles :

- Base d'apprentissage

Elle est utilisée pour former ou construire le modèle. Précisément, elle permet d'ajuster le modèle en calculant les coefficients ou les poids synaptiques. Dans ce travail, elle constituera 70% des éléments de la base de données.

- Base de validation

Elle est utilisée pour mesurer la généralisation du modèle et interrompre l'entrainement lorsque la généralisation cesse de s'améliorer. Elle constituera 10% de la base de données.

- Base de test

L'exactitude d'un modèle sur l'ensemble de validation est toujours une estimation optimiste de la façon dont il se comporterait en ayant des données inconnues. C'est parce que le modèle final est sorti comme le gagnant parmi les modèles concurrents basé sur le fait que son exactitude sur l'ensemble de validation est la plus élevée que nous avons besoin de mettre de côté une autre partie des données. Cette partie qui n'est utilisable ni dans l'apprentissage, ni dans la validation est connue sous le nom de base de test. La précision du modèle sur les données de test donne une estimation réalisée de la performance du modèle sur des données complètement inconnues.

3.2 Environnement de développement – Bibliothèque - Matériel utilisé

3.2.1 Environnement de développement

Le choix d'un environnement de développement se fait sur la base des avantages et des inconvénients liés au coût, à la flexibilité d'utilisation, le type de plateforme et l'ergonomie de l'interface. Pour des raisons économiques, nous avons choisi l'environnement MATLAB (R2020a). L'approche matricielle de MATLAB permet de

traiter les données sans aucune limitation de taille, de réaliser des calculs numériques et symboliques de façon fiable et rapide. Grâce aux fonctions graphiques de Matlab, il devient très facile de modifier interactivement les différents paramètres des graphiques pour les adapter selon nos souhaits. L'approche ouverte de Matlab permet de construire un outil sur mesure. Matlab comprend aussi un ensemble d'outils spécifiques à des domaines scientifique appelés *Toolbox* (Boîtes à Outils) ou bibliothèques qui sont indispensables à la plupart des utilisateurs. Matlab possède des avantages tel que les bibliothèques mathématiques très compréhensive, l'outil graphique incluant des fonctions d'interface graphique, des utilitaires, …

3.2.2 Bibliothèque utilisée

Les Bibliothèques sont des collections de fonctions qui étendent l'environnement de développement Matlab pour résoudre des catégories spécifiques de problèmes. Dans l'environnement de Matlab (2020a), celle dédiée aux RNA se nomme *Deep Learning Toolbox*. Elle offre les fonctionnalités permettant de créer, d'entraîner et valider un modèle de RNA dans le but d'effectuer une tâche bien précise.

3.2.3 Matériel utilisé

La configuration matérielle utilisée pour cette implémentation est la suivante :

- Un PC portable HP i3 CPU 3.4 GHZ
- Carte graphique : Nvidia GeForce GT525M
- Taille de la RAM : 4 GO
- Taille du Disque dur : 1 TO
- Système d'exploitation Windows 10

3.3 Simulation et résultats

3.3.1 Hyper paramètres de simulation

Il s'agit des valeurs d'initialisation qui ne peuvent pas être apprises durant l'entrainement du réseau. Ces hyper paramètres sont de type architectural et comportemental. Parmi les hypers paramètres de type architectural, on a le nombre de couche cachée et le nombre de neurone par couche. Les hypers paramètres de type comportemental concernent le nombre d'*epochs* (itérations), le taux d'apprentissage, la fonction de perte (*loss function* c'est une fonction de coût de type delta qui permet de mesurer la qualité d'une prédiction par rapport à une sortie désirée) et la fonction d'activation. Nous choisirons ces hypers paramètres en utilisant la technique d'horaire d'entrainement qui consiste établir à l'avance un horaire indiquant la valeur de l'hyper paramètre en fonction de l'époque.

3.3.2 Principe de simulation

Nous avons pour objectif de proposer pour un modèle de prédiction sa caractéristique architecturale ainsi que ses paramètres d'apprentissage. Pour y parvenir, nous nous appuyons sur l'algorithme d'optimisation des poids, le taux d'apprentissage du réseau et la *RMSE.* Pour choisir l'algorithme d'optimisation des poids parmi les trois qui ont été étudiés au chapitre 2, nous fixons certains hyper paramètres tout en faisant varier les autres. L'analyse du comportement de la *loss function* permettra de faire le choix. Pour déterminer le taux d'apprentissage, nous considérons le nombre d'*epoch* utilisé précédemment et nous observons l'évolution de la *loss function* ou la valeur de *RMSE* pour chaque valeur du taux d'apprentissage testée.

3.3.2.1 Sélection de l'algorithme d'optimisation des poids

- Hyper paramètres fixés

Nombre de couches cachées : 100 ;

Taux d'apprentissage : 0,003.

- Paramètres variables

Nombre d'*epoch* : 10 - 50 – 300 ; Algorithmes : *Adam, Adagrad, rmsprop.*

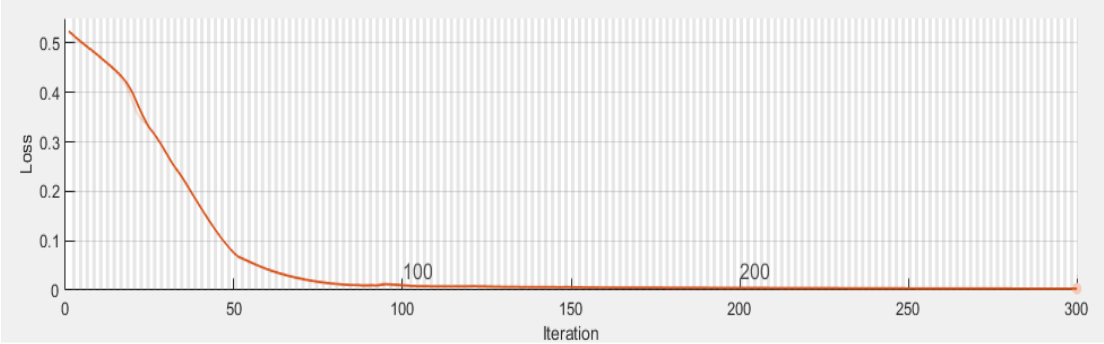

Figure 3.2: Évolution de la *loss function* pour l'algorithme *Adam*

Pour chaque algorithme d'optimisation des poids, nous avons fait varier le nombre d'itération pour l'entraînement du réseau. Nous avons constaté que pour des petites valeurs du nombre d'*epoch* et indépendamment de l'algorithme utilisé, la fonction de perte ne converge pas rapidement vers zéro. En utilisant un nombre d'*epoch* plus grand, la fonction de perte converge rapidement vers zéro pour tous les algorithmes d'optimisation. Notre choix s'est porté sur *Adam* parce qu'il converge plus vite que les autres (précisément à partir de la 100ème itération elle est presque nulle). À l'issue de ce constat, nous avons fixé le nombre d'itération pour les prochaines simulations à 300. L'ensemble des résultats obtenus avec tous les algorithmes d'optimisation en variant le nombre d'unités cachées sont disponibles en annexe A.

3.3.2.2 Détermination du taux d'apprentissage

- Hyper paramètres fixés

Nombre d'*epoch* : 300 ;

Nombre de couches cachées : 100 ;

- Hyper paramètre variable

Taux d'apprentissage : de 0,001 à 0,009 par pas de 0,001

Nous avons remarqué que pour des taux très bas (autour de 0.00001), la *loss function* diverge au fur et à mesure que le nombre d'itérations augmente. Nous faisons également un constat selon lequel les taux se situant au-dessus de 0,001, la *loss function* décroît plus rapidement. À cette valeur de 0,001, la *loss function* devient constante vers la 150ème itération. Lorsque ce taux est fixé à 0,005 cette fonction de perte qui décroit devient constante vers la 60ème itération traduisant ainsi une convergence rapide. Cette valeur est donc retenue pour la suite.

3.3.3 Modèles de prédiction des pertes de puissance

3.3.3.1 Modèle de prédiction à court terme

Ce modèle couvre une période de quatre mois. En considérant les hyper paramètres obtenus dans les paragraphes précédents, nous faisons varier un hyper paramètre architectural (nombre d'unités cachées) afin d'apprécier la valeur de *RMSE* obtenue.

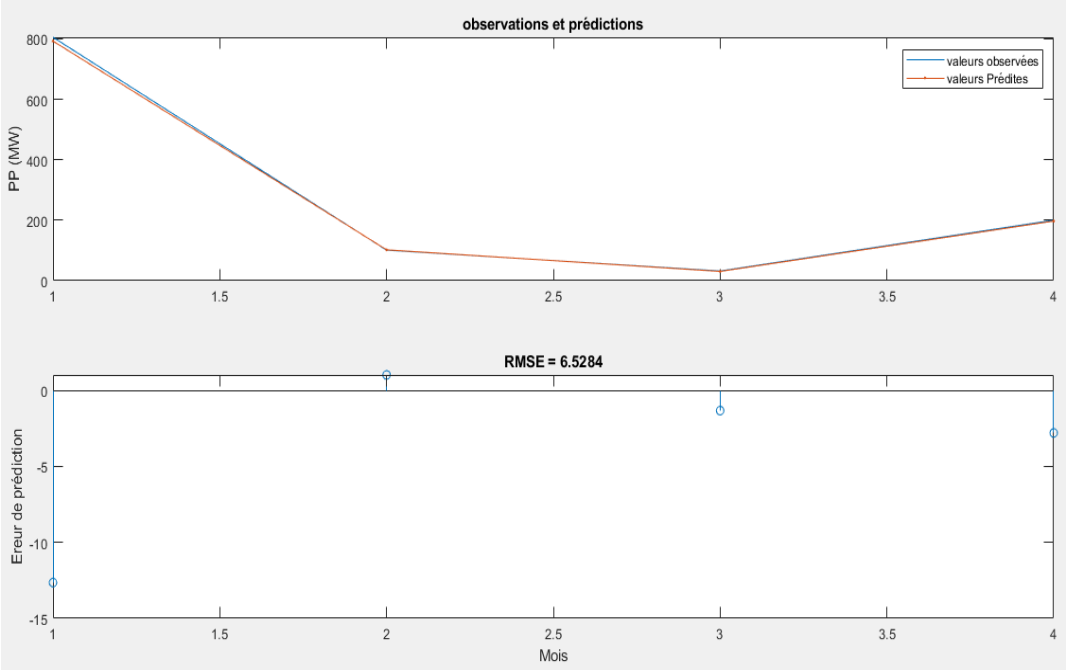

Figure 3.3: Graphe de prédiction et chronogramme d'erreurs à court terme

Tableau 3.2 : Paramètres du modèle à court terme

Durée de l'apprentissage	143 secondes
Algorithme d'apprentissage	Rétropropagation du gradient
Algorithme d'optimisation des poids	*Adam*
Algorithme d'initialisation des poids	Algorithme de *Random_uniform*
Taux d'apprentissage	0,005
Nombre d'*epoch*	300
RMSE	6,52
Nombre de couche cachée	300

3.3.3.2 Modèle de prédiction à long terme

Il couvre une période de 42 mois. C'est un temps nécessaire pour planifier des grands travaux relatifs à la maintenance préventive et même corrective. Le graphe ci-dessous illustre ses performances.

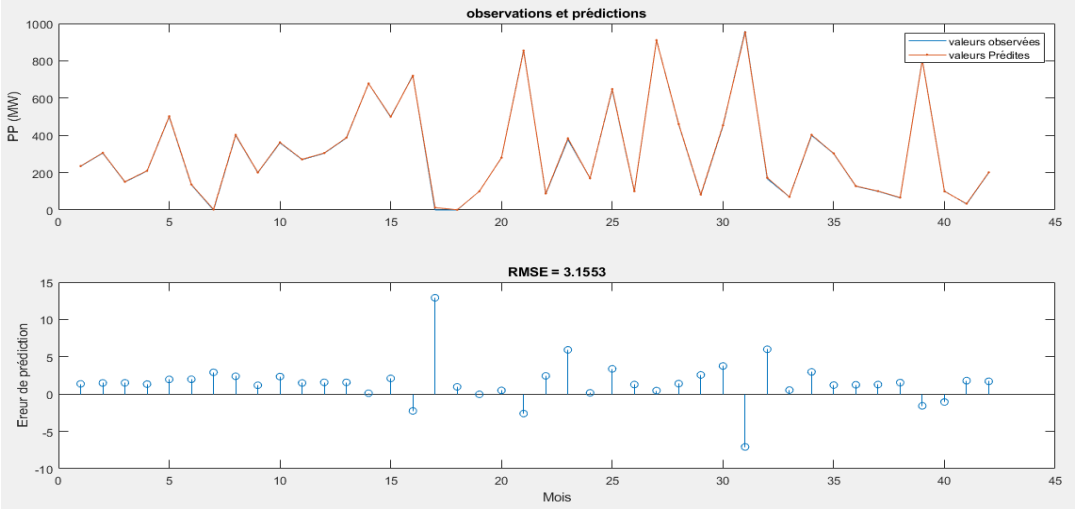

Figure 3.4: Graphe de prédiction et chronogramme d'erreurs à long terme

Tableau 3.3 : Paramètres du modèle à long terme

Durée de l'apprentissage	31 secondes
Algorithme d'apprentissage	Rétropropagation du gradient
Algorithme d'optimisation des poids	*Adam*
Algorithme d'initialisation des poids	Algorithme de *Random_uniform*
Taux d'apprentissage	0,005
Nombre d'*epoch*	300
RMSE	3,15
Nombre de couche cachée	150

3.4 Résultats obtenus avec d'autres outils d'apprentissage automatique dans Matlab

Nous présentons ici la valeur de la métrique d'évaluation pour différents modèles d'un outil d'apprentissage automatique. En annexes C et D sont présentés les codes pour la simulation du modèle de prédiction par SVM utilisant une fonction cubique pour le noyau ainsi que celui du modèle de régression linéaire utilisant une fonction de type *robust linear*.

3.4.1 Régression linéaire

Tableau 3.4 : Résultats de prédiction des modèles de régression linéaire

Modèle	*Linear*	*Interaction Linear*	*Robust linear*	*Stepwise linear*
RMSE	187,93	187,93	193,26	189,15

3.4.2 Arbres de décision

Tableau 3.5 : Résultats de prédiction des modèles par arbre de décision

Modèle	*Fine tree* (4 feuilles minimum)	*Medium tree* (12 feuilles minimum)	*Coarse tree* (36 feuilles minimum)	*Optimizable tree* (84 feuilles minimum)
RMSE	219,17	198,79	190,87	187,26

3.4.3 Support vecteur machine

Tableau 3.6 : Résultats de prédiction des modèles de SVM

Modèle	*Linear SVM*	*Quadratic SVM*	*Cubic SVM*	*Fine Gaussian SVM*	*Medium Gaussian SVM*	*Coarse Gaussian SVM*
RMSE	191,67	194,32	195,4	195,76	194,71	192,22

3.5. Interprétations des résultats et choix d'un modèle de prédiction

3.5.1 Interprétation des résultats

Le chapitre 2 a décrit le principe de fonctionnement de l'architecture *LSTM* tout en nous situant sur son champ de performance. Dans le paragraphe 3.3.3, nous avons présenté quelques résultats de simulation illustrant les performances de cette architecture dans des conditions précises. Dans ce paragraphe, nous essaierons d'interpréter ces résultats afin de savoir s'ils sont en conformité avec nos aspirations du chapitre 2. Cette interprétation nous permettra également de faire le choix du modèle de prédiction des pannes le plus fiable pour la gestion dudit réseau. Pour chaque horizon de prédiction, plusieurs architectures de *LSTM* ont été testées. La *RMSE* est la métrique que nous avons choisie pour évaluer la performance de chaque modèle. D'après cette métrique, un modèle est performant lorsque la valeur de la *RMSE* est le plus petit possible. À ce critère, nous avons ajouté celui lié au temps d'entrainement ainsi que le nombre de neurone utilisé. Nous avons constaté que le temps d'entrainement d'un modèle évolue dans le même sens que le nombre d'unité cachée utilisé dans l'architecture. Les tableaux 3.2 et 3.3 nous présentent les caractéristiques des modèles pour deux horizons de prédiction. La valeur de la *RMSE* pour chaque horizon de prédiction est assez faible. Ces valeurs ayant été obtenues en utilisant les paramètres préconisés au chapitre 2, cela confirme bien le fait que les architectures

neuronales de type *LSTM* sont bien indiquées pour la prédiction des séries temporelles. Les résultats présentés dans le paragraphe 3.4 issus des outils d'apprentissage automatique de type *linear regression* sont très éloignés de ceux obtenus avec le *LSTM*. Cet écart est en partie dû à l'impossibilité de modéliser l'évolution des données d'apprentissage par une fonction de type linéaire ou affine. La figure 3.1 nous montre que l'évolution des données présente une allure qui n'est pas facilement approchable avec une fonction classique connue. C'est le cas de l'outil SVM où, pour quatre différentes fonctions utilisées pour le noyau, l'on obtient des mauvais résultats.

3.5.2 Choix du modèle et de l'horizon de prédiction

En se basant sur la métrique d'évaluation, nous constatons que les modèles de prédiction issus des outils présentés au paragraphe 3.4 ont des mauvaises performances. Par conséquent, nous ne pouvons les choisir. Le modèle de prédiction de type neuronale est celui qui nous permet de faire un compromis entre la valeur de *RMSE* et la taille de son architecture interne. Compte tenu également des performances de l'algorithme d'optimisation d'Adam, nous choisissons le modèle dont la *RMSE* vaut **2.23** et possède 250 neurones dans sa couche interne. Cette architecture correspond à un horizon de prédiction à long terme (42 mois). L'annexe A présente le tableau contenant les résultats obtenus avec d'autres architectures de *LSTM* tandis que l'annexe B nous présente le code Matlab de simulation d'une architecture *LSTM*.

Conclusion

Le chapitre qui s'achève nous a permis de simuler des modèles de prédiction des pannes sur un réseau de transport d'énergie électrique. Pour y parvenir nous avons essayé de respecter les étapes clés de la mise en place d'un projet en intelligence artificielle. Plusieurs modèles ont été simulés et le choix s'est fait sur la base d'un compromis qui a été illustré dans le paragraphe précédent. Le modèle de prédiction sur un horizon de 42 mois a été retenu comme étant le meilleur.

CONCLUSION

Dans ce travail, nous avons proposé des modèles informatiques empiriques de prédiction des pertes de puissance d'un réseau électrique. Ce modèle est construit à base d'un LSTM qui est l'une des meilleures architectures neuronales profonde dédiée à la prédiction des séries temporelles. La structuration du chapitre 3 s'est faite dans le but de respecter les quatre étapes clés relatives à la réalisation d'un projet en IA. Il s'agit précisément de la détermination de l'objectif du projet, collecte et préparation des données pertinentes, classement des données plus choix des outils et enfin entraînement du modèle. Les résultats obtenus démontrent à suffisance l'avantage lié à l'utilisation des RNA profond comme outil Prédictif par rapport aux méthodes probabilistes présentées au chapitre 1. Cela peut être en partie dû au fait que la prédiction dans le cas échéant se réduit à un problème de continuation d'une série temporelle dès lors qu'elle s'appuie sur les données historiques. Parmi ces résultats, le plus saillant est celui du modèle à long terme entraînés avec l'algorithme *Adam* qui a une très grande précision. Ce résultat vient confirmer l'avantage comparatif de cet algorithme vis-à-vis des deux autres. Ce modèle de prédiction constitue une contribution significative en vue de l'amélioration de la maintenance préventive. La procédure utilisée pour établir ces modèles empiriques de prédiction peut être reprise dans d'autres domaines tel que la santé, l'économie, … à condition de disposer d'un historique de données fiable et très vaste.

Notre travail comporte toutefois des limites :

- Les « trous » présents dans la base de données n'ont pas facilité son exploitation. En effet, il y'a des périodes où les rapporteurs n'ont pas pu obtenir ou estimer les valeurs relatives aux pertes de puissance. Par conséquent, les données utilisées pour l'entrainement couvrent une période de 14 ans au lieu de 19 ans. Cette réalité va à l'encontre des exigences d'une architecture

neuronale de type profonde qui a besoin d'une base de données d'entrainement la plus vaste possible afin de produire les meilleurs résultats de prédiction.

- L'architecture de LSTM utilisée dans ce travail est la variante de base qui a été proposée en 1997 par Hochreiter et Schmidhuber. Des nouvelles variantes de LSTM ont vues le jour depuis l'an 2000 jusqu'à la décennie en cours.
- Le traitement effectué sur les données : en effet, l'on pourrait affiner les horizons de prédiction en les ramenant du mensuel au journalier. Cela exige la réalisation d'une interpolation linéaire à base de splines cubiques sur les données initiales.
- La petite taille de la base de données. Nous n'avons que 168 points de données. Pour effectuer une analyse plus solide, il nous faudrait plus de 1000 points de données.

À la lumière de ces limites, nous recommandons aux rapporteurs dans ces différents secteurs d'activités de mettre tout en œuvre pour une collecte efficace des données relatives à l'évolution d'un processus impactant le fonctionnement de ladite activité. Nous pensons également qu'il serait intéressant de simuler les performances de ces modèles avec les nouvelles variantes de *LSTM* tel que *GRU*, *LSTM* avec *peephole*, *Grid LSTM* ou les réseaux de type 'Attention' afin de garantir la transportabilité des modèles.

ANNEXE A

Tableau récapitulatif des résultats obtenus avec autres architectures de LSTM

Prédictions (Mois)	***Adam***			***AdaGrad***			***RMSProp***		
	Nombre D'unités cachées	**Architecture réseau**	***RMSE***	**Nombre D'unités cachées**	**Architecture réseau**	***RMSE***	**Nombre D'unités cachées**	**Architecture réseau**	***RMSE***
4	50	1-50-1	**5,47**	50	1-50-1	**3,16**	50	1-50-1	**11,7**
	100	1-100-1	**4,8**	100	1-100-1	**2,73**	100	1-100-1	**17,2**
	150	1-150-1	**7,08**	150	1-150-1	**10,5**	150	1-150-1	**36,6**
	200	1-200-1	**4,8**	200	1-200-1	**7,35**	200	1-200-1	**31**
	250	1-250-1	**6,14**	250	1-250-1	**4,86**	250	1-250-1	**10,5**
	300	1-300-1	**6,52**	300	1-300-1	**5,06**	300	1-300-1	**13,8**
9	50	1-50-1	**8,25**	50	1-50-1	**7,12**	50	1-50-1	**7,79**
	100	1-100-1	**20**	100	1-100-1	**6,54**	100	1-100-1	**21,5**
	150	1-150-1	**9,60**	150	1-150-1	**5,31**	150	1-150-1	**11,4**
	200	1-200-1	**11,7**	200	1-200-1	**5,07**	200	1-200-1	**23,8**
	250	1-250-1	**9,36**	250	1-250-1	**6,84**	250	1-250-1	**25,9**
	300	1-300-1	**1,72**	300	1-300-1	**11,6**	300	1-300-1	**11,4**
17	100	1-100-1	**16,8**	100	1-100-1	**4,37**	100	1-100-1	**26,4**
	150	1-150-1	**10,1**	150	1-150-1	**5,05**	150	1-150-1	**37**
	200	1-200-1	**10,5**	200	1-200-1	**6,41**	200	1-200-1	**29,9**
	250	1-250-1	**8,23**	250	1-250-1	**6,16**	250	1-250-1	**40,5**
	300	1-300-1	**9,25**	300	1-300-1	**4,18**	300	1-300-1	**23,4**

21	100	1-100-1	**12,7**	100	1-100-1	**5,45**	100	1-100-1	**34,5**
	150	1-150-1	**13,6**	150	1-150-1	**5,3**	150	1-150-1	**42,8**
	200	1-200-1	**12,7**	200	1-200-1	**11,3**	200	1-200-1	**38,0**
	250	1-250-1	**10,6**	250	1-250-1	**5,00**	250	1-250-1	**25,5**
	300	1-300-1	**6,05**	300	1-300-1	**6,28**	300	1-300-1	**14,9**
26	100	1-100-1	**15,5**	100	1-100-1	**5,46**	100	1-100-1	**38,8**
	150	1-150-1	**14,2**	150	1-150-1	**5,87**	150	1-150-1	**28,4**
	200	1-200-1	**14,9**	200	1-200-1	**5,28**	200	1-200-1	**23,0**
	250	1-250-1	**20,2**	250	1-250-1	**5,36**	250	1-250-1	**42,8**
	300	1-300-1	**12,4**	300	1-300-1	**7,65**	300	1-300-1	**34,2**
34	100	1-100-1	**12,2**	100	1-100-1	**15,01**	100	1-100-1	**24,7**
	150	1-150-1	**12,1**	150	1-150-1	**16,94**	150	1-150-1	**20,9**
	200	1-200-1	**7,99**	200	1-200-1	**10,9**	200	1-200-1	**43,2**
	250	1-250-1	**13,5**	250	1-250-1	**15,52**	250	1-250-1	**21,0**
	300	1-300-1	**7,9**	300	1-300-1	**8,51**	300	1-300-1	**38,1**
42	100	1-100-1	**12,7**	100	1-100-1	**6,19**	100	1-100-1	**20,3**
	150	1-150-1	**10,8**	150	1-150-1	**7,2**	150	1-150-1	**24,3**
	200	1-200-1	**19,1**	200	1-200-1	**5,06**	200	1-200-1	**20,2**
	250	1-250-1	**2,23**	250	1-250-1	**11,5**	250	1-250-1	**18,9**
	300	1-300-1	**7,79**	300	1-300-1	**5,52**	300	1-300-1	**23,3**

ANNEXE B

Code pour la simulation d'une architecture LSTM dans Matlab

```
% Valeurs mensuels des PP(MW) de janvier 2005 Ã  DÃ©cembre 2018
x =  [227 402 180 190 509 109 171 366 305 281 191 292 75 234 200 162 343 318 304 159 284 342 203 354 380 353 137 132 235 435 275
data = timeseries(x,1:168);
plot(data);
xlabel('Mois');
ylabel('Puissances Perdus  (MW)');
title('Graphe des Pertes de Puissance');
% partition des donnÃ©es en 75% pour l'entrainement et 30% pour les tests
numTimeStepsTrain = floor(0.75*numel(x));
dataTrain = x(1:numTimeStepsTrain);     %variable contenant les donnÃ©es d'entrainement
dataTest = x(numTimeStepsTrain+1:end);    %variable contenant les donnÃ©es du test
% Standardisation ou normalisation des donnÃ©es d'entrainnement pour qu'il aient une moyenne nulle et une variance unitaire
mu = mean(dataTrain);             % calcul de la moyenne
sig = std(dataTrain);             % calcul de variance(standard deviation)
dataTrainStandardized = (dataTrain - mu) / sig;   % formule de normamisation
% dÃ©finition des prÃ©dicteurs et de leurs rÃ©ponses:prÃ©diction par pas de
% 1(utile si l'on veut prÃ©dire sur 1 mos
XTrain = dataTrainStandardized(1:end);  % le prÃ©dicteur est constituÃ© de la sÃ©quence d'entrainnement sans le pas final
YTrain = dataTrainStandardized(1:end);    % la rÃ©ponse est celle de la sÃ©quence d'entrainnement reculÃ©e d'un pas
%configuration du rÃ©seau LSTM
numFeatures = 1;
numResponses = 1;
numHiddenUnits = 150;  %nombre d'unitÃ©s cachÃ©es du rÃ©seau
layers = [ ...
     sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits)
    fullyConnectedLayer(numResponses)
    regressionLayer];
%paramÃ¨tres d'entrainement du rÃ©seau
options = trainingOptions('adam', ...
```

```
options = trainingOptions('adam', ...
    'MaxEpochs',300, ...
    'GradientThreshold',1, ...
    'InitialLearnRate',0.001, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',50, ...
    'LearnRateDropFactor',0.9, ...
    'Verbose',0, ...
    'Plots','training-progress'); % tracÃ© de la courbe de progression de l'entrainement en fonction de du nombre d'itteration
 % Syntaxe d'entrainement du rÃ©seau
 net = trainNetwork(XTrain,YTrain,layers,options);
 % prÃ©diction sur plusieurs intervalles de temps en faisant la mise Ã  jour
 % avec les donnÃ©es prÃ©dites
  dataTestStandardized = (dataTest - mu) / sig; % normalisation des donnÃ©es de test
 XTest = dataTestStandardized(1:end);  % syntaxe de parcours des donnÃ©es de test
 net = predictAndUpdateState(net,XTrain);  %fait la prÃ©diction sur les valeurs d'entrainement
 [net,YPred] = predictAndUpdateState(net,YTrain(end)); %fait la premiÃ¨re prÃ©diction pour avoir la rÃ©ponse de la derniÃ¨re valeur des donn
 numTimeStepsTest = numel(XTest); %compte le nombre d'Ã©lement que contient la base de donnÃ©es de test
for i = 2:numTimeStepsTest
    [net,YPred(:,i)] = predictAndUpdateState(net,YPred(:,i-1),'ExecutionEnvironment','cpu');
end  % parcoure la base de donnÃ©es de test pour faire les prÃ©dictions des valeurs de la base de test
 % dÃ©normaalise les donnÃ©es prÃ©dites
  YPred = sig*YPred + mu;
 %calcul de RMSE Ã  partir de des valeurs prÃ©dites et celles de test
  YTest = dataTest(1:end); % parcoure de la base de test
 rmse = sqrt(mean((YPred-YTest).^2));
```

```
subplot(2,1,1);
plot(YTest);
hold on;
plot(YPred,'.-');
hold off;
legend(['Observed Forecast']);
ylabel('Cases');
title('Forecast');
subplot(2,1,2);
stem(YPred - YTest);
xlabel('Mois');
ylabel('Erreur');
title("RMSE =  " + rmse);
% Mise Ã  jour du rÃ©seau avec les valeurs observÃ©es (meilleure mÃ©thode que celle qui consiste Ã  le faire avec les valeurs prÃ©dites
%net = resetState(net);  % reinitialisation de l'Ã©tat du rÃ©seau
net = predictAndUpdateState(net,XTrain); % prÃ©diction dans le rÃ©seau Ã  partir des donnÃ©es d'entrainement
YPred = []; %la valeur prÃ©dite est contenue dans un vecteur ligne
numTimeStepsTest = numel(XTest);% compte le nombre de donnÃ©es qu'il y'a dans la BD de test
for i = 1:numTimeStepsTest % parcoure ces donnÃ©es de la premiÃ¨re jusqu'Ã  la derniÃ¨re
    [net,YPred(:,i)] = predictAndUpdateState(net,XTest(:,i),'ExecutionEnvironment','cpu');%prÃ©dit chacune d'elle et fait la mise Ã  jo
end
% dÃ©normalise les valeurs prÃ©dites et calcule la rmse
YPred = sig*YPred + mu;
rmse = sqrt(mean((YPred-YTest).^2));
%comparaison des valeurs prÃ©dites avec celle de la base de test
figure
subplot(2,1,1);
plot(YTest);
hold on;
plot(YPred,'.-');
```

```
hold off;
legend(["valeurs observÃ©es" "valeurs PrÃ©dites"]);
ylabel('PP (MW)');
title('observations et prÃ©dictions');

subplot(2,1,2);
stem(YPred - YTest);
xlabel('Mois');
ylabel('Ereur de prÃ©diction');
title("RMSE = " + rmse);
```

ANNEXE C

Code pour la simulation d'un modèle de prédiction à base de SVM

```
inputTable = array2table(trainingData', 'VariableNames', {'mois'});

predictorNames = {'mois'};
predictors = inputTable(:, predictorNames);
response = responseData(:);
isCategoricalPredictor = [false];
responseScale = iqr(response);
if ~isfinite(responseScale) || responseScale == 0.0
    responseScale = 1.0;
end
boxConstraint = responseScale/1.349;
epsilon = responseScale/13.49;
regressionSVM = fitrsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 3, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', boxConstraint, ...
    'Epsilon', epsilon, ...
    'Standardize', true);
predictorExtractionFcn = @(x) array2table(x', 'VariableNames', predictorNames);
svmPredictFcn = @(x) predict(regressionSVM, x);
trainedModel.predictFcn = @(x) svmPredictFcn(predictorExtractionFcn(x));
trainedModel.RegressionSVM = regressionSVM;

inputTable = array2table(trainingData', 'VariableNames', {'mois'});
predictorNames = {'mois'};
predictors = inputTable(:, predictorNames);
```

```
response = responseData(:);
isCategoricalPredictor = [false];

KFolds = 5;
cvp = cvpartition(size(response, 1), 'KFold', KFolds);
validationPredictions = response;
for fold = 1:KFolds
    trainingPredictors = predictors(cvp.training(fold), :);
    trainingResponse = response(cvp.training(fold), :);
    foldIsCategoricalPredictor = isCategoricalPredictor;

        responseScale = iqr(trainingResponse);
    if ~isfinite(responseScale) || responseScale == 0.0
        responseScale = 1.0;
    end
    boxConstraint = responseScale/1.349;
    epsilon = responseScale/13.49;
    regressionSVM = fitrsvm(...
        trainingPredictors, ...
        trainingResponse, ...
        'KernelFunction', 'polynomial', ...
        'PolynomialOrder', 3, ...
        'KernelScale', 'auto', ...
        'BoxConstraint', boxConstraint, ...
        'Epsilon', epsilon, ...
        'Standardize', true);


    svmPredictFcn = @(x) predict(regressionSVM, x);
    validationPredictFcn = @(x) svmPredictFcn(x);

      validationPredictors = predictors(cvp.test(fold), :);
    foldPredictions = validationPredictFcn(validationPredictors);

end

isNotMissing = ~isnan(validationPredictions) & ~isnan(response);
validationRMSE = sqrt(nansum(( validationPredictions - response ).^2) / numel(response(isNotMissing) ));
```

ANNEXE D

Code de simulation d'un modèle de prédiction base sur la régression linéaire

```
inputTable = array2table(trainingData', 'VariableNames', {'mois'});

predictorNames = {'mois'};
predictors = inputTable(:, predictorNames);
response = responseData(:);
isCategoricalPredictor = [false];

concatenatedPredictorsAndResponse = predictors;
concatenatedPredictorsAndResponse.pp = response;
linearModel = fitlm(...
    concatenatedPredictorsAndResponse, ...
    'linear', ...
    'RobustOpts', 'on');

predictorExtractionFcn = @(x) array2table(x', 'VariableNames', predictorNames);
linearModelPredictFcn = @(x) predict(linearModel, x);
trainedModel.predictFcn = @(x) linearModelPredictFcn(predictorExtractionFcn(x));

trainedModel.LinearModel = linearModel;
trainedModel.About = 'This struct is a trained model exported from Regression Learner R2020a.';
trainedModel.HowToPredict = sprintf('To make predictions on a new predictor row matrix, X, use: '

inputTable = array2table(trainingData', 'VariableNames', {'mois'});

predictorNames = {'mois'};
predictors = inputTable(:, predictorNames);
response = responseData(:);
isCategoricalPredictor = [false];
KFolds = 5;

cvp = cvpartition(size(response, 1), 'KFold', KFolds);

validationPredictions = response;
for fold = 1:KFolds
    trainingPredictors = predictors(cvp.training(fold), :);
    trainingResponse = response(cvp.training(fold), :);
    foldIsCategoricalPredictor = isCategoricalPredictor;

    concatenatedPredictorsAndResponse = trainingPredictors;
    concatenatedPredictorsAndResponse.pp = trainingResponse;
    linearModel = fitlm(...
        concatenatedPredictorsAndResponse, ...
        'linear', ...
        'RobustOpts', 'on');

    linearModelPredictFcn = @(x) predict(linearModel, x);
    validationPredictFcn = @(x) linearModelPredictFcn(x);

    validationPredictors = predictors(cvp.test(fold), :);
    foldPredictions = validationPredictFcn(validationPredictors);

    validationPredictions(cvp.test(fold), :) = foldPredictions;
end

isNotMissing = ~isnan(validationPredictions) & ~isnan(response);
validationRMSE = sqrt(nansum(( validationPredictions - response ).^2) / numel(response(isNotMissing) ));
```

BIBLIOGRAPHIE

A. Y. Ng and M. I. Jordan. (2002). "*On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes,"* Advances in neural information processing systems, vol. 2, p. 841–848.

Ajit Kumar Verma and Durga Rao Karanki. (2010). «*Reliability and Safety Engineering »,* Indian Institute of Technology: Springer.

Ange Tato. (2018) "*Apprentissage Machine : Techniques et applications,"* in cours INF7470, Université du Québec à Montréal.

Bengio Y., Sinard P. and Frasconi P. (1994). "*Learning Long-Term Dependencies with Gradient Descent is Difficult",* IEEE transactions on neural networks, 5, 2.

César Laurent. (2017, 24 août). "*Réseaux Récurrents* ", in Ecole d'été francophone en apprentissage profond.

Chatfield C. (2000). *Time-Series Forecasting.* CRC Press.

D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. (1985). "*A learning algorithm for Boltzmann machines'',* Cognitive science, vol. 9, no. 1, p. 147–169.

D. Kingma and J. Ba. (2014). "*Adam: A method for stochastic optimization,"* arXiv preprint arXiv: 1412.6980.

Dario A MODEI et al. (2016). « *Deep Speech 2*: *End-to-End Speech Recognition in English and Mandarin »,* in: Proceedings of the 33rd International Conference on Machine Learning.

Farzaneh M. and Zhang J. (2000). "*Modelling of DC Arc Discharge on Ice Surfaces". IEEE Proceedings Generation, Transmission and Distribution,* Vol.147, No.2.

G. Cybenko. (1989). "*Approximation by superpositions of a sigmoidal function,"* Mathematics of Control, Signals, and Systems (MCSS), vol. 2, no. 4, p. 303–314.

G. Hinton, N. Srivastava, and K. Swersky. (2012). "*Neural networks for machine learning lecture 6a overview of mini–batch gradient descent*,".

GUO, J. (2013). "*Backpropagation through time".* Rapport technique, Semantic scholar.

H. Wu. (2009). "*Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions,"* Information Sciences, vol. 179, no. 19, p. 3432–3441.

J. Duchi, E. Hazan, and Y. Singer. (2011). "*Adaptive subgradient methods for online learning and stochastic optimization,"* Journal of Machine Learning Research, vol. 12, no. Jul, p. 2121–2159.

James, G. D., Hastie T. et R. Tibshirani (2013). An introduction to statistical learning: With Application in R. New York: Edition Springer.

Les niveaux de maintenances industrielles. (2020, 5 août). Accueil - Le canada Français. Récupéré de https://www.canadafrancais.com/2020/08/05/les-niveaux-de-maintenance-industrielle/

Logofet DO, Lesnaya EV. (2000). *The mathematics of Markov models: what Markov chains can really predict in forest successions.* Ecological Modelling n°126, p. 285-298.

Ludovic Trottier. (2020). '*Optimisation pour l'apprentissage profond'.*

Marc Parizeau. (2004). '*Réseaux de neurones'.* Université de Laval.

P. J. Werbos. (1990). "*Back propagation through time: what it does and how to do it,"* Proceedings of the IEEE, vol. 78, no. 10, p. 1550–1560.

Pascanu R., Mikolov T., and Bengio Y. (2013). "*On the difficulty of training recurrent neural network"*. In 30ème conference international sur le machine learning, Atlanta, USA.

Pr. L. Meva'a. (2020, 21 février). Cours de maintenance et fiabilité industriel ENSP 4GIM. p. 28-32.

S. Hochreiter and J. Schmidhuber (1997). "*Long short-term memory,"* Neural computation, vol. 9, no. 8, p. 1735–1780.

S. Le Digabel. (2017). Ecole Polytechnique de Montréal. P.23

Tallec C. and Ollivier Y. (2018) "*Unbiasing truncated back propagation through time".* In ICLR 2018 conference, Vancouver, Canada.

Vapnik V. (1998). Statiscal Learning Theory. John Wiley and Sons, Chichester.

Villemeur A. (1988). « Sûreté de fonctionnement des systèmes industriels », Éditions Eyrolles, Collection de la Direction des études et recherches d'électricité de France, France.

W. S. McCulloch and W. Pitts. (1943). "*A logical calculus of the ideas immanent in nervous activity,"* The bulletin of mathematical biophysics, vol. 5, no. 4, p. 115–133.

X. Glorot, A. Bordes, and Y. Bengio. (2011). "*Deep sparse rectifier neural networks.,"* in Aistats, vol. 15, p. 275.

Y. Bengio, E. Laufer, G. Alain, and J. Yosinski. (2013). "*Deep generative stochastic networks trainable by backprop,"* in International Conference on Machine Learning.

Y. Dodge and V. Rousson. (2004). Analyse de régression appliquée Dunod.