International Journal of Innovation and Applied Studies
ISSN 2028-9324 Vol. 32 No. 2 Mar. 2021, pp. 271-290

http://www.ijias.issr-journals.org/

L'Algorithme de rétro-propagation de gradient dans le perceptron multicouche: Bases et étude de cas

[Gradient Back-Propagation Algorithm in the Multi layer Perceptron: Foundations and Case Study]

Nsenge Mpia Héritier and Inipaivudu Baelani Nephtali

Département d'Informatique de Gestion, Université de l'Assomption au Congo, B.P 104, Butembo, Nord-Kivu, RD Congo

ABSTRACT: Training a multi-layer neural network is sometimes difficult, especially for novices in Artificial Intelligence. Many people believe that this training must be relayed to computers in order to be able to perform these ultra-powerful calculations. As a result, we can't figure out what is going on behind these calculations, thinking that there is too much mathematics, making it difficult for humans to understand what is at stake. Far from this mythical caricature stuck to neural networks. The training of a neural network consists in finding synaptic weights such that the output layer allows to classify with precision the observed values of a training set with the aim of allowing the created model to present generalisation capacities on examples that it will never have encountered during the training phase.

KEYWORDS: Artificiel Intelligence, Loss Function, Epoch, Logitic Function, Neural Network.

RESUME: Entrainer un réseau neuronal multicouche s'avère difficile, surtout pour des novices en Intelligence artificielle. Nombreuses personnes estiment qu'il faut relayer cet entrainement aux ordinateurs pour pouvoir effectuer ces calculs ultra puissants. Du coup, on n'arrive pas à cerner ce qui se passe derrière ces calculs, pensant qu'il y a trop de mathématiques, difficile pour l'homme d'en comprendre des enjeux. Loin de là cette caricature mythique collée aux réseaux de neurones. L'entraînement d'un réseau de neurones consiste à trouver des poids synaptiques tels que la couche de sortie permette de classer avec précision les valeurs observées d'un ensemble d'entraînement dans le but de permettre à ce que le modèle créé présente des capacités de généralisation sur des exemples qu'il n'aura jamais rencontrés lors de la phase d'entrainement.

MOTS-CLEFS: Intelligence Artificielle, Fonction Coût, Itération, Fonction Logistique, Réseau de Neurones.

1 INTRODUCTION

Lorsqu'on parle de réseau de neurones en Intelligence Artificielle, nombreuses personnes, spécialement profanes en la matière, pensent directement à des grandes fonctions mathématiques et au fonctionnement complexe du cerveau humain. Du coup, ils ne veulent même pas entrer à fond de ce domaine recent qui révolutionne l'informatique de nos jours afin d'en comprendre les merveilles qu'il cache et même la simplicité de son fonctionnement dans la résolution presque de la quasi totalité des problèmes complexes modernes. Bien sûr, la mise au point d'un réseau de neurones ne s'écarte pas de l'implémentation de quelques fonctions mathématiques et dont chacune réalise une tâche bien précise. On y trouve, de ce fait, des fonctions permettant:

- D'activer la sortie d'une information qui a été prise à l'entrée du neurone et traitée dans les différentes couches du réseau. Pour cela, le modèle a besoin d'une fonction qui soit dérivable en tout point pour calculer la sortie du perceptron. Pour cet article, nous utiliserons la fonction sigmoïde:

$$S_{(x)} = \frac{1}{1+e^{-x}}$$

- D'optimiser l'algorithme d'apprentissage. En fait, le but principal d'un réseau de neurones, c'est d'approximer la fonction *f* ayant un ensemble de paramètres θ et des valeurs x que le réseau doit calculer à une fonction inconnue *g*. Sur ce, la capacité de f (x, θ) à approximer la fonction $g\ (x)$, qui contient la valeur exacte cible, est évaluée en utilisant la fonction coût. Dans notre cas, nous utiliserons la méthode de Descente de gradient afin de minimiser le coût d'erreur dans l'apprentissage du réseau et de modifier des poids des paramètres qui constituent les connexions entre les neurones. Etant donné qu'il existe plusieurs types de fonctions coût dans cette méthode, nous ferons usage de la fonction de moindre carré [2]:

$$E(x,y,\theta) = \frac{1}{2}\sum_{i=1}^{m} (f\ (x_i,\theta)\ - y_i)^{\,2}$$

  E représente la fonction d'erreur (coût), $f\ (x_i,\theta)$ est la prédiction des neurones de sortie de la valeur x_i munie de certains paramètres θ et y_i c'est la valeur cible spécifiée dans l'ensemble d'entraînement.

- De calculer des gradients. Avec la fonction coût, on peut arriver à savoir l'erreur de l'avant-dernière couche du réseau. Ce qui permet ainsi de calculer l'erreur de la couche précédente afin de modifier les poids des différents paramètres du réseau. Par conséquent, ce retour en arrière pour calculer des erreurs des couches précédentes s'appelle la rétro-propagation [9]. Ce calcul se fait via le processus d'apprentissage par rétro-propagation qui utilise le théorème de dérivation des fonctions composées appelé en anglais *chain rule*. Ainsi, soit l'erreur $E(x,y,\theta)$, la mise à jour du poids de la connexion w_{jl}^{h} du neurone j de la couche h-1 vers le neurone l, est calculée comme suit [3]:

$$\frac{\partial E(x,y,\theta)}{\partial w_{jl}^{h}}$$

  Apres avoir appliqué ce *chain rule*, on actualisera ainsi les poids pour refaire l'entrainement du modèle afin d'observer, dans la prochaine itération (Epoch), les résultats obtenus, c'est-a-dire si f (x, θ^*) ≈ g (x). On suit, de ce fait, la règle suivante pour mettre à jour les poids des paramètres à chaque itération:

$$w_{jl}^{h+} = w_{jl}^{h} - \eta * \frac{\partial E(x,y,\theta)}{\partial w_{jl}^{h}}$$

  Avec w_{jl}^{h} la valeur du poids à l'iteration précédente, w_{jl}^{h+} le nouveau poids après ajustement et le η learning rate qui est un paramètre qui détermine dans quelle mesure les poids peuvent changer en réponse à une erreur observée sur l'ensemble de données à entrainer.

- Etc.

Dans cet article, nous voulons expliquer le fonctionnement de l'algorithme de la rétro-propagation de gradient en des termes simples par un exemple clair pour qu'après la lecture de cet article, le lecteur ait un meilleur aperçu de ce qu'est une fonction d'activation dans un modèle de machine learning, spécialement dans le réseau de neurones artificiels; du rôle de l'apprentissage par descente de gradient et du comment fonctionne la rétro-propagation.

2 RÉSEAU DE NEURONES

2.1 LA BASE D'UN RÉSEAU DE NEURONES

Les techniques de l'intelligence artificielle sont, de nos jours, utilisées dans plusieurs domaines de recherches notamment dans le traitement d'image, la médicine, le big data, la régulation de processus industriels, la prédiction d'une série temporelle,

etc. Dans cet article, notre effort se focalise sur la technique de réseau des neurones artificiels. Dans cette technique, l'élément le plus basic à connaitre est celui de Perceptron qui est l'algorithme d'apprentissage pour des réseaux de neurones formels (simplement neurone) à deux couches (couche d'entrée et couche de sortie). Rappelons qu'un neurone, dans son acception artificielle, est un opérateur mathématique qui transforme les valeurs de ses entrées en sortie selon des règles précises. Il peut ainsi effectuer la somme de ses entrées, comparer la valeur de la somme résultante à une valeur seuil pour enfin répondre en émettant un signal indiquant si cette somme est supérieure ou égale à ce seuil. En ce sens, un neurone est une fonction algébrique qui est non linéaire et est bornée, dont la valeur dépend des paramètres poids [16].

Considérons un ensemble de données contenant des informations sur des fruits ayant certaines caractérisques et dont on veut prédire si c'est un fruit délicieux ou pas:

Tableau 1. Exemple d'un dataset pour étude de cas d'un perceptron

Fruit	*Goût*	*Graine*	*Chair*	*Délicieux ?*
1	*1*	*1*	*0*	*1*
2	*1*	*0*	*1*	*1*
3	*0*	*0*	*0*	*0*
4	*1*	*1*	*1*	*1*
5	*0*	*0*	*1*	*0*

Dans l'exemple ci-haut, les caractéristiques sont: goût, graine et chair. C'est en fonction de ces trois caractéristiques que l'on peut savoir si le fruit est délicieux ou pas. Et, 1 dans la colonne ayant la cellule jaune représente délicieux et 0 représente pas délicieux. Cette dernière colonne est appelée Target (y). Tandis que les trois colonnes qui la précédent (caractéristiques) sont appelées Features. De ces données, il faut maintenant créer un modèle de neurone pour arriver à prédire, à partir des nouvelles données des fruits si, partant de leurs caractéristiques, ils sont délicieux ou pas. Ainsi, doter ce modèle de cette capacité, en terme technique se dit entrainer le modèle.

Précisons qu'un Perceptron est composé d'une couche d'entrée, appelée aussi vecteur de *n* neurones qui correspond chacune à une variable d'entrée (Dans notre exemple, le fruit 2 a pour variables d'entrée 1,0,1) et ces neurones vont transmettre la valeur de leur entrée à la couche suivante, dans le cas échéant, à la couche de sortie. Toutefois, pour ajouter de la flexibilité à l'algorithme du perceptron, en permettant de varier le seuil de déclenchement du neurone par l'ajustement des poids lors de l'apprentissage, la nouvelle version du perceptron intègre l'ajout d'une unité appelée biais qui prend souvent -1 et 1 comme valeur [3]. Ce qui fait que le vecteur fruit 2 devient $\vec{x} = (1, 1, 0{,}1)$, si on ajoute le biais de 1. Dans le cas de neurone formel, le biais joue le rôle de seuil d'activation. Ayant des valeurs d'entrées, le réseau ne manque que des poids qui pondèrent les entrées et peuvent être modifiés par apprentissage. On peut nommer le poids w. Le poids w_{jl} est donc la connexion entre le neurone d'entrée *j* et le neurone de sortie *l* dont le processus d'activation de la sortie s'illustre de la manière suivante:

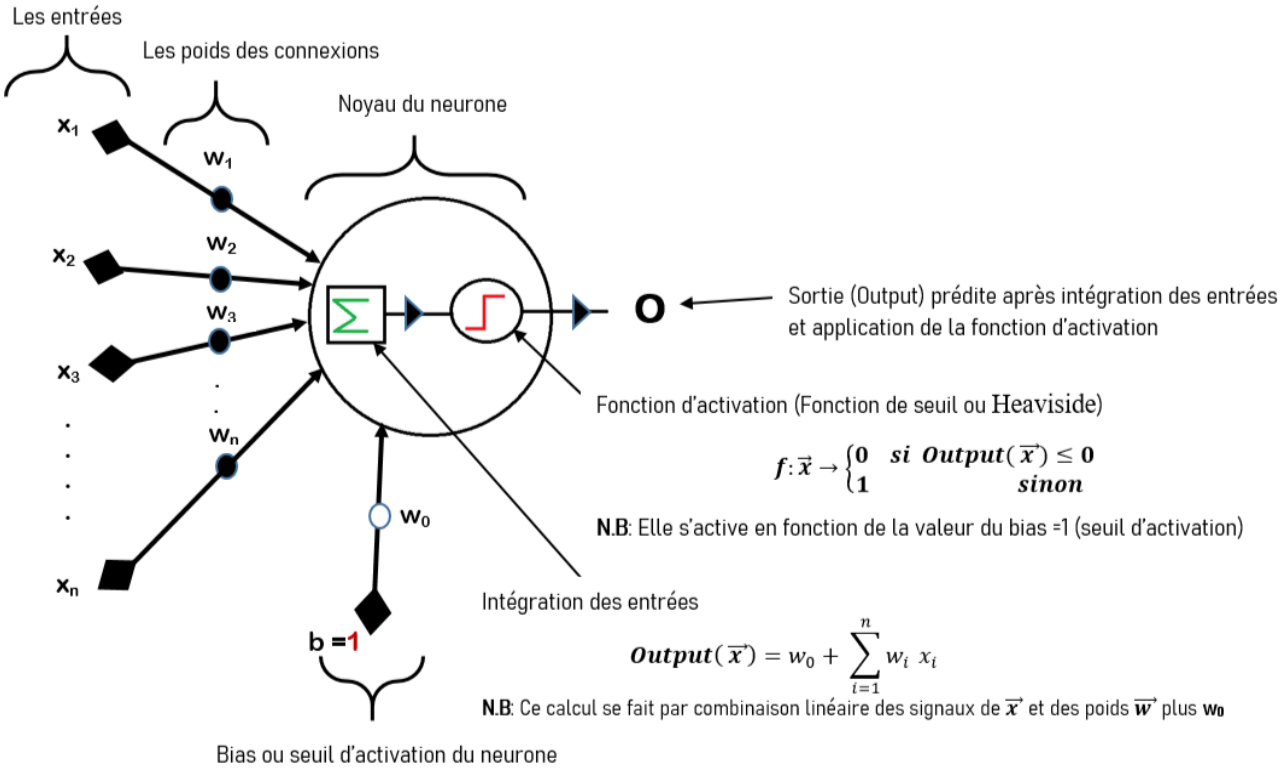

Fig. 1. Représentation du processus d'activation d'un neurone formel

Si nous retournons à l'exemple de notre dataset des fruits ci-haut, nous pouvons appliquer l'algorithme de perceptron en ceci pour le cas du fruit 1:

Observation du fruit 1

$\vec{x} = fruit\ 1 = (1, 1, 0)$ et $\vec{w}$**= (0, 0, 0)** sont les poids correspondant à chaque variable d'entrée x_i. Le biais est initialisé ici à 1 et vecteur de poids de connexion à 0, car l'algorithme d'entrainement nécessite l'initialisation aléatoire, au début, du vecteur de poids de connexion [3]. Signalons que la valeur du vecteur de poids de connexion n'est pas nécessairement 0, les poids sont initialisés avec des valeurs aléatoires tirées de l'intervalle entre (-1, +1) [10].

Afin d'entrainer notre modèle pour l'observation du fruit, nous devons préciser que l'entrainement se fait en suivant cet algorithme:

1. Insérer les valeurs d'entrée ($\vec{x}$) d'une observation
2. Initialiser aléatoirement les poids ($\vec{w}$)
3. Calculer la combinaison linéaire $\vec{x}$ et $\vec{w}$ de l'observation plus w_0
4. Faire vérifier si la valeur de sortie ($\hat{y}$) après l'activation marche avec le target (y)
   - Calculer l'erreur de prédiction pour l'observation
   - Mettre à jour les poids de connexion
5. Jusqu'à convergence entre **y** et $\hat{y}$

Sur ce, la combinaison linéaire output ($\vec{x}$) = $w_0 + \sum_{i=1}^{n} w_i\ x_i$ = 0 + (1*0) + (1*0) + (0*0) = 0. C'est ce 0 qui est dans le noyau du perceptron ci-dessous (Cf. Fig. 2). Si nous regardons la valeur d'output de la Fig. 2, nous constatons qu'elle vaut 0. En fait, étant donné que le résultat de output ($\vec{x}$) qui est égal à 0 est inférieur à la valeur du biais qui vaut 1. C'est ainsi que selon la règle d'activation (Cf. Fig.1), le output ($\hat{y}$) vaut 0.

Nous constatons que $\hat{y}$ ne correspond pas à y. En effet, la valeur de y pour le fruit 1 vaut 1, Cf. Il y a donc erreur. Pour calculer cette erreur, dans le cadre de neurone formel, on utilise la formule E = **y** - $\hat{y}$. Sur ce, E = 1- 0 = 1. Ce dernier indique réellement qu'il y a erreur de prédiction. Il faut donc ajuster les poids de connexion afin de permettre au perceptron de ne pas

commettre l'erreur de classification de ce premier fruit. En ce sens, on applique la règle suivante d'ajustement: $\Delta w_j = \eta\ (y - \hat{y})\ x_j$ et le poids ajusté sera calculé comme suit: $w_j^+ = w_j + \Delta w_j$.

Légende

$y - \hat{y}$: Erreur de prédiction. C'est elle qui détermine s'il faut corriger ou non les paramètres.

$\hat{y}$: Valeur de sortie prédite par le modèle

y: Target cible. Pour le fruit 1, c'est 1. Cf.

x_j: C'est la force du signal. En fait, c'est la variable se trouvant dans l'échelle du poids que l'on veut ajuster. Dans notre cas, nous avons 1, 1, 0.

η: Taux d'apprentissage qui détermine l'amplitude de l'apprentissage. Le choix de ce taux intrigue souvent. En effet, quand c'est trop petit, il y aura la lenteur de convergence et quand c'est trop grand, Il y a une sorte d'oscillation [11]. En général, le taux d'apprentissage est choisi entre 0.05 et 0.15. Dans le cas de notre exemple, nous avons choisi 0.25.

w_j: C'est le poids précédent

w_j^+: Le poids ajusté.

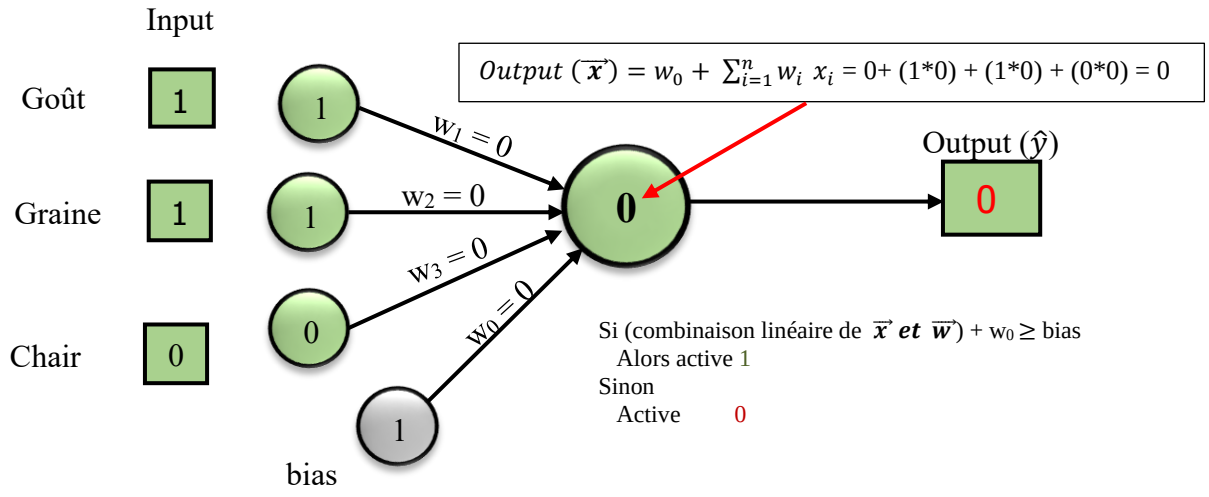

Fig. 2. Entrainement du perceptron avec l'observation du fruit

Cela étant, nous pouvons ajuster nos poids comme suit:

Tableau 2. Résultat des poids ajustés dans le premier entrainement

Features pour fruit 1	x_j	**Erreur** = $y - \hat{y}$ =1 - 0 = 1	η	$\Delta w_j = \eta\ (\text{Erreur})\ x_j$	w_j	$w_j^+ = w_j + \Delta w_j$
Goût	1	1	0.25	0.25	0	0.25
Graine	1	1	0.25	0.25	0	0.25
Chair	0	1	0.25	0	0	0

En remplaçant les nouvelles valeurs des poids (w_j^+) dans notre algorithme, nous obtenons le résultat suivant:

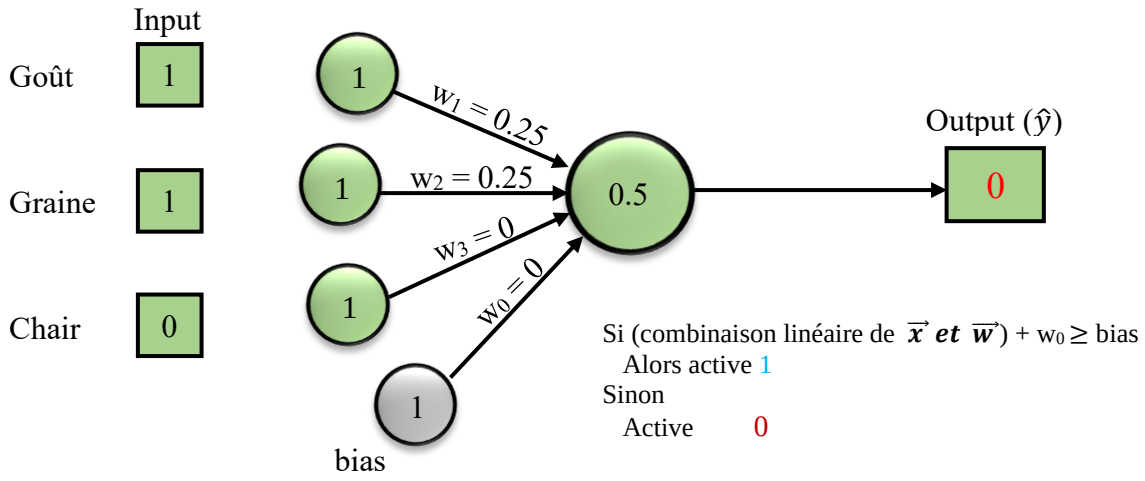

Fig. 3. Résultat du premier entrainement du perceptron avec l'observation du fruit 1

Le résultat de Fig. 3 montre que le perceptron a encore activé 0 car 0.5 est inférieur au biais. D'où, la continuation de l'entrainement du modèle jusqu'à ce qu'il apprenne et prédise le vrai résultat correspondant au target du fruit 1.

Tableau 3. Résultat des poids ajustés dans le deuxième entrainement

Features pour fruit 1	x_j	**Erreur** = $y - \hat{y}$ =1 - 0 = 1	η	$\Delta w_j = \eta\,(\text{Erreur})\,x_j$	w_j	$w_j^+ = w_j + \Delta w_j$
Goût	1	1	0.25	0.25	0.25	0.5
Graine	1	1	0.25	0.25	0.25	0.5
Chair	0	1	0.25	0	0	0

On constate dans le diagramme ci-dessous que notre modèle a appris pour cette observation car la sortie $\hat{y}$ correspond au target *y* cible pour l'observation du fruit 1. On peut ainsi entrainer, avec le même processus, les autres observations de

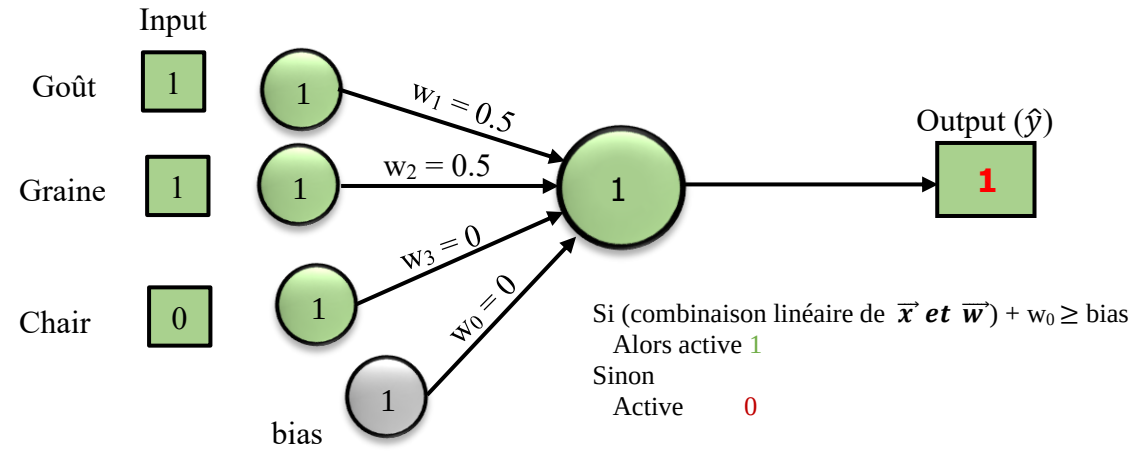

Fig. 4. Arrêt de l'apprentissage pour l'observation du fruit 1

En substance, nous venons là d'utiliser la fonction de seuil ou de Heaviside pour pouvoir prédire une étiquette binaire (soit 1, soit 0). En fait, le perceptron dans sa dimension unitaire est un classifieur linéaire qui classifie des données à partir d'une combinaison linéaire de ses entrées. De ce fait, il est incapable de classifier des données dans des classes non linéairement séparables. Ainsi, pour des cas des problèmes de classification multi-classe, on devrait modifier cette architecture que nous avons utilisé ci-haut de sorte à n'avoir non plus un seul neurone comme sortie mais plutôt *C* neurones dans la couche de sortie

tel que *C* le nombre de différentes classes de sortie. En ce sens, les *j* + 1 neurones de la couche d'entrée seront tous connectés à chacun de neurone de sortie. Sur ce, on utilise la fonction *softmax* comme fonction d'activation [3].

Bref, le perceptron pose de problème d'apprentissage car sa capacité de modélisation ne prend pas en compte les modèles non linéaires. Alors que nous faisons face aujourd'hui à des problèmes complexes non linéaires qui doivent être modélisés. D'où, l'importance de mise en place des perceptrons multicouche appelés aussi *Multi-layer perceptron* (MLP) afin de rendre l'apprentissage plus profond [5].

2.2 PERCEPTRON MULTI-COUCHE

Le PMC est un modèle de réseau qui contient, à la différence de neurones formels qui n'ont que la couche d'entrée et celle de sortie, des couches cachées ayant pour fonction de faire des traitements intermédiaires pour une meilleure prédiction [1]. Ces architectures font usage des fonctions qui sont non linéaires telles que la fonction tangente hyperbolique ou fonction logistique pour activer les sorties des neurones. On peut instruire avec ces modèles un réseau avec plusieurs couches cachées et de neurones. Ainsi, plus il y a de couches, plus le réseau est profond et le modèle devient riche. C'est du Deep Learning [7].

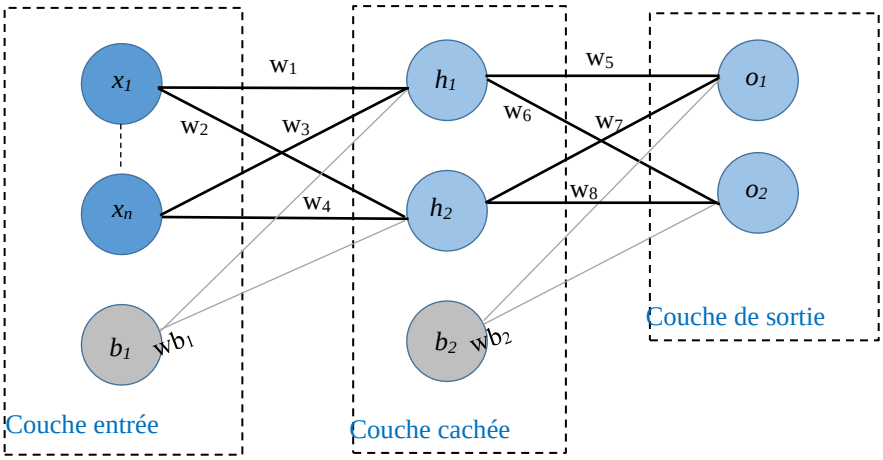

Fig. 5. Exemple d'une architecture du perceptron multicouche

Fig. 5 représente un réseau composé de neurones interconnectés en trois couches successives. La première couche, qui est celle d'entrée est composée de neurones dits transparents qui n'effectuent aucun calcul mais qui distribuent leurs entrées à tous les neurones de la couche suivante dite couche cachée. Les neurones de la couche cachée reçoivent ainsi les entrées {x_1, …, x_n} de la couche d'entrée avec les poids associés {w_1, …, w_n}. Chaque neurone de la couche cachée comprend deux phases. La première que l'on peut appeler *netinput* contient le résultat du calcul de la somme pondérée des entrées de la couche d'entrée et du biais: $\text{Netinput}_j = (x_1*w_1) + (x_2*w_2) + \ldots + (x_n*w_n) + (b_j*w_{bj})$ et la seconde *netoutput* comprend le résultat de la transformation de *netinput* par l'intermédiaire de la fonction d'activation. En général, les couches cachées utilisent la fonction ReLU (*Rectified Linear Unit*), en francais, Unité Linéaire Rectifiée pour le cas de prédiction des valeurs positives non dérivable et la couche de sortie utilise soit la fonction sigmoïde pour le cas d'une sortie binaire soit la fonction softmax pour des classifications multi-classe comme fonctions d'activation. Toutefois, dans le cadre de cet article, nous allons utiliser la fonction sigmoïde (logistique) comme fonction d'activation pour aussi bien la couche de sortie que celles cachées. Signalons que le même processus de *netinput* et de *netoutput* s'effectue aussi dans les neurones de la couche de sortie.

Dans le cas où il y a erreur dans la sortie, on n'utilisera plus le même processus comme pour le réseau de neurone formel, mais plutôt l'algorithme de la rétro-propagation qui est une méthode permettant de mettre à jour les coefficients d'un réseau de neurones multicouche en comparant la sortie obtenue et la sortie désirée (target). Cette méthode vérifie la contribution à l'erreur de chaque neurone après le traitement d'un batch (lot) de données [6]. Le point **4.1** illustre le fonctionnement de cet algorithme.

En sus, les perceptrons multicouches sont des réseaux neuronaux artificiels de type feedforward qui s'inspirent des capacités de la cognition humaine et du système cérébral pour traiter l'information [14]. En fait, un réseau neuronal de type feedforward peut être considéré comme étant une régression logistique, mais avec une projection sur certaines couches

cachées pour rendre l'entrée plus linéairement séparable. Cette méthode initialisée des poids aléatoires et est formée par la minimisation de l'erreur rétro-propagée en utilisant une méthode basée sur la descente de gradient [8].

3 PRÉDIRE LA VALEUR DE SORTIE AVEC LA FONCTION SIGMOÏDE

3.1 FONCTIONS D'ACTIVATION

Il existe plusieurs types de fonctions d'activation. On parle de fonction d'activation du perceptron non lisse lorsque le taux d'erreur du modèle est une fonction discontinue des paramètres de poids. Avec ce type de fonctions, l'ajustement des poids optimaux en minimisant la fonction de perte est souvent difficile. D'où, l'application d'une fonction d'activation continue afin de résoudre ce problème. En ce sens, la fonction sigmoïde logistique est un choix approprié à cet effet car elle est capable d'apprendre les limites de décision non linéaires pour les espaces séparables non linéaires. En fait, les fonctions d'activation dans le PMC doivent être non linéaires, continuellement différenciables et non décroissantes de façon monotone [4].

En outre, il est souhaitable de choisir une fonction d'activation dont la dérivée serait facilement calculée. En faisant ainsi usage d'une fonction d'activation non linéaire, par exemple sigmoïde, le PMC est un approximateur de fonction universelle. C'est-à-dire qu'un réseau de perceptron multicouche à une seule couche cachée associée à un nombre suffisamment grand et fini de neurones peut approcher et apprendre toute fonction continue sur un domaine compact avec une précision arbitraire [15].

3.2 NÉCESSITÉ DE LA FONCTION SIGMOÏDE

Lors de la rétro-propagation en PMC, l'objectif est de minimiser la fonction coût. Ainsi, les fonctions sigmoïde et tangente hyperbolique, calculent toutes deux leurs dérivés de manière très simple et efficace, ce qui justifie leur utilisation courante lors de l'optimisation du gradient dans un PMC. Bien que les fonctions ReLU sont utilisées dans des réseaux de neurones modernes et permettent un apprentissage plus rapide et plus efficace des réseaux neuronaux profonds sur des données complexes et à haute dimension, en calculant la fonction f (x) = max (0, x) et en seuillant simplement la matrice d'entrée à zéro, en ne nécessitant pas de calcul coûteux [15], la fonction sigmoïde est le plus utilisé car elle est dérivable en tout point [13].

On utilise ainsi la dérivée de la fonction Sigmoïde pour calculer le coefficient d'erreur. Elle permet de calculer les erreurs en faisant une marche en arrière tout en parcourant toutes les couches en partant de la sortie à l'entrée pour voir comment l'erreur a été propagée sur le réseau neuronal afin d'ajuster les poids des connexions des neurones [13]. Considérons l'exemple d'un réseau de neurones qui classifie des images qu'on lui donne en entrée afin de déterminer si l'image est celle d'un chien ou d'un chat (Fig. 6). Ce modèle, en cas d'une erreur de classification, fera une marche en arrière en faisant des calculs des dérivées partielles de l'erreur E (x, y, θ) par rapport à w_{jl}^{h} correspondant à la valeur du poids reliant le neurone l de la couche h à son entrée j.

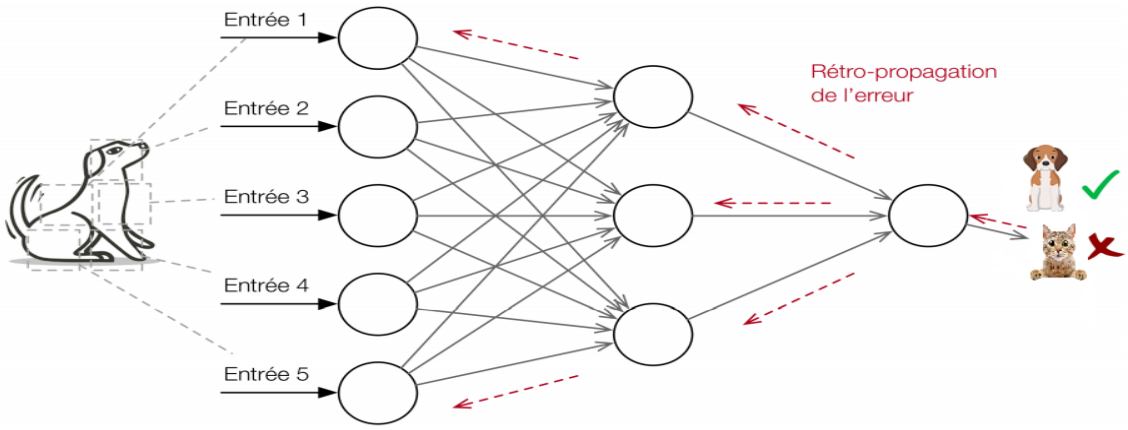

Fig. 6. Illustration graphique du processus de la rétro-propagation

4 ENTRAINER UN RESEAU DE NEURONES AVEC L'ALGORITHME DE RÉTRO-PROPAGATION DE GRADIENT

4.1 REVUE DE LITTÉRATURE

D'emblée, pour effectuer l'apprentissage, trois éléments sont essentiels. La *Force du signal*, l'*Erreur* et le *Taux d'apprentissage*. Toutefois, il est important de savoir, certaines notions élémentaires interviennent lors de l'apprentissage d'un modèle de perceptron multicouche.

Rappelons que l'algorithme de rétro-propagation de gradient est l'un des algorithmes les plus répandus dans les réseaux de neurones de type feedforward, c'est-à-dire à propagation directe. Cet algorithme comprend deux grandes phases:

1. Propagation ou Forward Pass: Dans cette première phase, à chaque itération (Epoch), on donne au réseau un ensemble des entrées. Ces entrées sont propagées jusqu'à la couche de sortie avec un résultat $\hat{y}$ (output). Si cette sortie ne correspond pas au target cible ***y***, l'algorithme va faire une marche en arrière (de la couche de sortie à la couche d'entrée) pour corriger cette erreur en modifiant progressivement les poids se trouvant dans chaque neurone de chaque couche.
2. Correction de l'erreur Backward Pass: Dans le premier Epoch, le réseau ne fournit pas souvent exactement ce que l'on attend comme résultat. Il doit d'abord apprendre. On calcule ainsi l'erreur entre la valeur de sortie et la valeur cible. De façon générale, on fait la somme quadratique moyenne des erreurs, en anglais *mean squared error* (MSE) pour tous les neurones de sortie que l'on rétro-propage dans le réseau.

Cela étant, les poids du réseau seront changés à chaque itération. Ce changement se fait de sorte à minimiser l'erreur entre la sortie désirée, le target cible et le résultat du réseau à une entrée x_i. Ce changement est ainsi effectué via la méthode de descente de gradient. En fait, partant de deux phases ci-haut, l'algorithme propage le signal d'entrée dans le réseau à chaque itération dans le sens de la sortie afin d'obtenir une sortie et l'on calcule l'erreur entre cette sortie et la sortie désirée. Ensuite, par rétro-propagation, on calcule des erreurs intermédiaires qui correspondent aux couches cachées. Enfin, on ajuste des poids w_{jl}^{h}. Le but de l'apprentissage est de trouver donc les poids corrects des connexions, c'est-à-dire donnant au réseau un comportement se rapprochant le plus possible de celui du target. D'où, l'algorithme de rétro-propagation consiste à effectuer plusieurs itérations (Epoch) jusqu'à ce que l'erreur soit suffisamment petite [16].

Algorithme de rétro-propagation de gradient

Initialisation aléatoire des poids w_{ij} tirés de l'intervalle [-1, +1]

Normalisation dans l'intervalle [0, 1] des données d'apprentissage

Tant que la fonction coût E > erreur max ou Epoch n < Epoch max **Fait**

Propager un exemple X: x_i =f (SP_i) avec SP_i la somme pondérée ((x_1*w_1) + (x_2*w_2) + … + (x_n*w_n) + (b_j*w_{bj}))

Rétro-propager:

Couche de sortie: δ_i= -$f'(SP_i)$ $(target_i - x_i)$ avec x_i la valeur prédite après activation

Couche cachée: $\delta_i = f'(SP_i) \sum_k w_{ki}\, \delta_k$ avec *k* la couche suivante

Mettre à jour chaque poids $w_{ij}^{+} = w_{ij} - \eta * \frac{\partial E(x,y,\theta)}{\partial w_{ij}}$

Fin Tant que

4.2 ETUDE DE CAS

Imaginons une base d'observation ayant pour entrées x_1 = 2; x_2 = 3, avec les biais b1 = 1; b2 = 1; b3 = 1; b4 = 1 ayant respectivement pour poids 0.25; 0.45; 0.15; 0.35. Le taux d'apprentissage est donné par η = 0.1. Les targets cibles pour x_1 vaut 1 et x_2 vaut 0.2. Enfin, les poids de connexion sont donnés par W1 = 0.3; W2 = 0.2; W3 = -0.4; W4 = 0.6; W5 = 0.7; W6 = -0.3; W7 = 0.5; W8 = -0.1. On peut représenter ce réseau de la manière suivante:

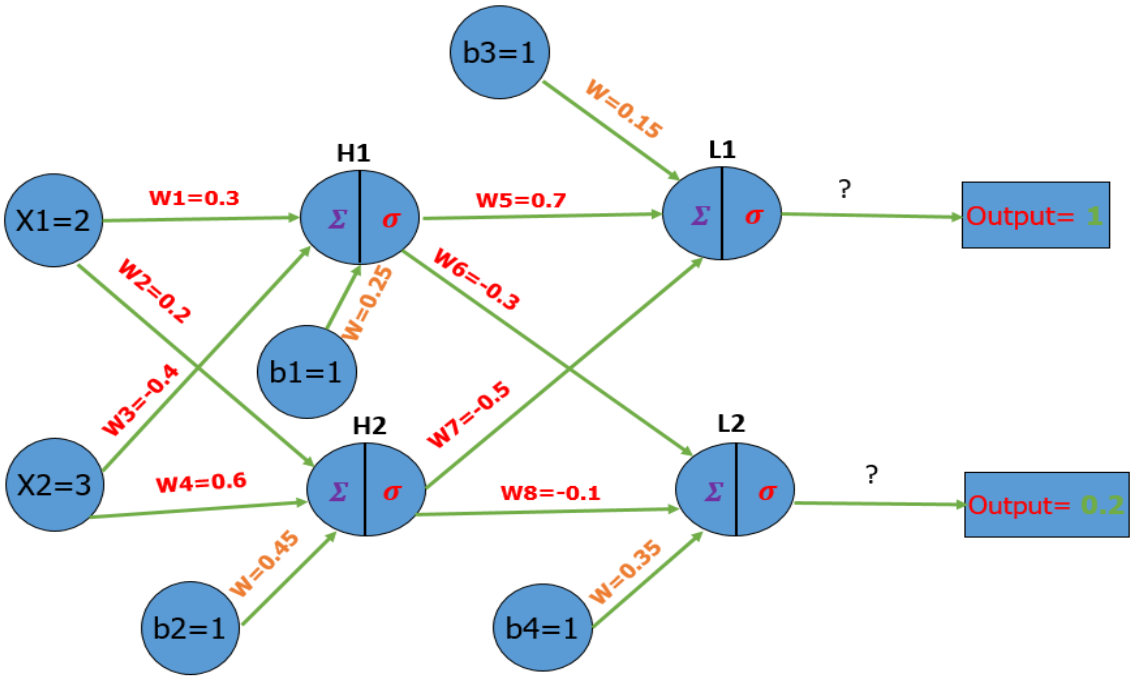

Fig. 7. Réseau relatif à l'observation

Le réseau ci-dessus comprend une couche cachée (H) et une couche de sortie (L). Le but est d'entrainer ce modèle pour qu'il soit généralisé pour d'autres observations. On voit également que dans chaque neurone, il y a deux parties Σ et σ. Le symbole de somme représente le Netinput tandis que le sigma représente le Netoutput. Rappelons que nous allons entrainer ce modèle en itérant les deux phases de l'algorithme de rétro-propagation de gradient:

Epoch 1

Forward Pass

Par definition,

NetinputH1 = (w1*x1 + w3*x2 + bias1*wbias1)

= [(0.3*2) + ((-0.4) *3) + (0.25*1)]

= **-0.35**

et

NetoutputH1 = $\frac{1}{1+e^{-NetinputH1}}$

= $\frac{1}{1+e^{-(-0.35)}}$

= 0.413382421

En appliquant les mêmes procédures pour les autres neurones, on obtient ce résultat:

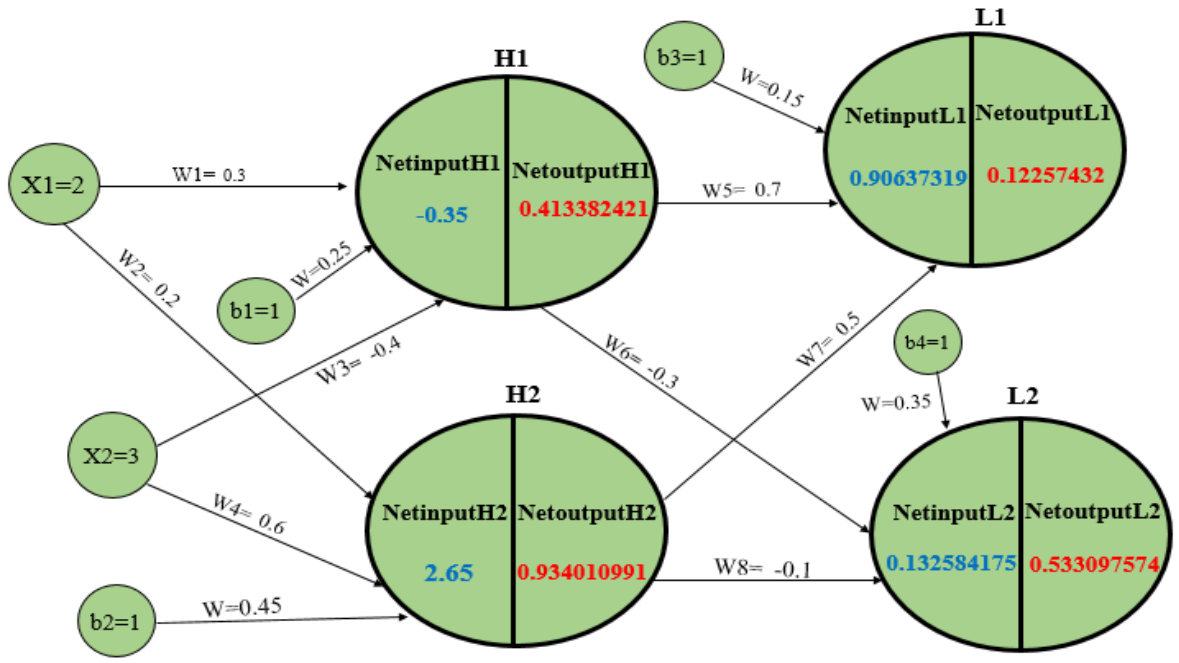

Fig. 8. Forward Pass Epoch 1

Nous pouvons voir que le target cible pour x_1 est 1, mais le modèle à prédire 0,712257432, il y a donc une erreur. De même, pour x_2 qui est 0,2, la sortie du réseau neuronal vaut 0,533097574, ce qui est également une erreur. Nous calculons donc l'erreur pour enfin faire le backward Pass et ajuster les poids. En effet, nous voulons trouver des poids et biais qui feront que, quelles que soient les entrées x_1 et x_2, nous ayons toujours un NetoutputL1 le plus proche de 1 et NetoutputL2 le plus proche de 0.2.

Calcul d'Erreur

En faisant usage de la fonction de moindre carré, nous obtenons:

$E_{outputL1} = 1/2\ (Target1 - OutputL1)^2$

$= 1/2\ (1 - 0.712257432)^2$

$= 0.041397893$

$E_{outputL2} = 1/2\ (Target2 - OutputL2)^2$

$= 1/2\ (0.2 - 0.533097574)^2$

$= 0.055476997$

$E_{Total} = E_{outputL1} + E_{outputL2}$

$= 0.041397893 + 0.055476997$

$= 0.096874889$

Etant donné que l'erreur obtenue ci-haut, nous allons maintenant passer à la seconde phase de la rétro-propagation.

Backward Pass

Comme les résultats de sortie ne sont pas les mêmes avec les résultats attendus, et avec l'obtention d'une erreur, nous allons maintenant calculer le changement de poids en utilisant la règle d'apprentissage. Nous pouvons donc appliquer le théorème de dérivation des fonctions composées en dérivant l'erreur par rapport à chaque poids.

Soit la fonction composée f (g (h (x))), sa dérivée partielle donne:

$$\frac{\partial f}{\partial x} = \frac{\partial h}{\partial x} * \frac{\partial g}{\partial h} * \frac{\partial f}{\partial g}$$

Dans le cas échéant,

f: $E_{outputi}$

x: w_{jl}^{h}

g: $Netoutput_i$

h: $Netintput_i$

Ce qui donne:

$$w1 = \frac{\partial Etotal}{\partial w1} = \frac{\partial NetinputH1}{\partial w1} * \frac{\partial NetoutputH1}{\partial NetinputH1} * \frac{\partial Etotal}{\partial NetoutputH1}$$

$$w2 = \frac{\partial Etotal}{\partial w2} = \frac{\partial NetinputH2}{\partial w2} * \frac{\partial NetoutputH2}{\partial NetinputH2} * \frac{\partial Etotal}{\partial NetoutputH2}$$

$$w3 = \frac{\partial Etotal}{\partial w3} = \frac{\partial NetinputH1}{\partial w3} * \frac{\partial NetoutputH1}{\partial NetinputH1} * \frac{\partial Etotal}{\partial NetoutputH1}$$

$$w4 = \frac{\partial Etotal}{\partial w4} = \frac{\partial NetinputH2}{\partial w4} * \frac{\partial NetoutputH2}{\partial NetinputH2} * \frac{\partial Etotal}{\partial NetoutputH2}$$

$$w5 = \frac{\partial Etotal}{\partial w5} = \frac{\partial NetinputL1}{\partial w5} * \frac{\partial NetoutputL1}{\partial NetinputL1} * \frac{\partial Etotal}{\partial NetoutputL1}$$

$$w6 = \frac{\partial Etotal}{\partial w6} = \frac{\partial NetinputL2}{\partial w6} * \frac{\partial NetoutputL2}{\partial NetinputL2} * \frac{\partial Etotal}{\partial NetoutputL2}$$

$$w7 = \frac{\partial Etotal}{\partial w7} = \frac{\partial NetinputL1}{\partial w7} * \frac{\partial NetoutputL1}{\partial NetinputL1} * \frac{\partial Etotal}{\partial NetoutputL1}$$

$$w8 = \frac{\partial Etotal}{\partial w8} = \frac{\partial NetinputL2}{\partial w8} * \frac{\partial NetoutputL2}{\partial NetinputL2} * \frac{\partial Etotal}{\partial NetoutputL2}$$

Notes:

$$\frac{\partial Etotal}{\partial NetoutputH1} = \frac{\partial EoutputL1}{\partial NetoutputH1} + \frac{\partial EoutputL2}{\partial NetoutputH1}$$

$$\frac{\partial EoutputL1}{\partial NetoutputH1} = \frac{\partial EoutputL1}{\partial NetL1} * \frac{\partial NetinputL1}{\partial NetoutputH1}$$

$$\frac{\partial EoutputL1}{\partial NetinputL1} = \frac{\partial EoutputL1}{\partial NetoutputL1} * \frac{\partial EoutputL1}{\partial NetinputL1}$$

$$\frac{\partial EoutputL1}{\partial NetoutputL1} = -(Target1 - NetoutputL1)$$

$$\frac{\partial EoutputL1}{\partial NetinputL1} = NetoutputL1 * (1 - NetoutputL1)$$

$$\frac{\partial NetinputL1}{\partial NetoutputH1} = w5 = 0.7$$

$$\frac{\partial EoutputL2}{\partial NetoutputH1} = \frac{\partial EoutputL2}{\partial NetinputL2} * \frac{\partial NetinputL2}{\partial NetinputH1}$$

$$\frac{\partial EoutputL2}{\partial NetinputL2} = \frac{\partial EoutputL2}{\partial NetoutputL2} * \frac{\partial EoutputL2}{\partial NetinputL2}$$

$$\frac{\partial EoutputL2}{\partial NetoutputL2} = -(Target2 - NetoutputL2)$$

$$\frac{\partial EoutputL2}{\partial NetinputL2} = NetoutputL2 * (1 - NetoutputL2)$$

$$\frac{\partial NetinputL2}{\partial NetinputH1} = w6 = -0.3$$

$$\frac{\partial NetinputH1}{\partial w1} = x1 = 2$$

$$\frac{\partial NetoutputH1}{\partial NetinputH1} = NetoutputH1 * (1 - NetoutputH1)$$

$$w1 = \frac{\partial Etotal}{\partial w1}$$

$$\frac{\partial Etotal}{\partial NetoutputH2} = \frac{\partial EoutputL2}{\partial NetoutputH2} + \frac{\partial EoutputL1}{\partial NetoutputH2}$$

$$\frac{\partial EoutputL2}{\partial NetoutputH2} = \frac{\partial EoutputL2}{\partial NetinputL2} * \frac{\partial NetinputL2}{\partial NetoutputH2}$$

$$\frac{\partial EoutputL2}{\partial NetinputL2} = \frac{\partial EoutputL2}{\partial NetoutputL2} * \frac{\partial EoutputL2}{\partial NetinputL2}$$

$$\frac{\partial EoutputL2}{\partial NetoutputL2} = -(Target2 - NetoutputL2)$$

$$\frac{\partial EoutputL2}{\partial NetinputL2} = NetoutputL2 * (1 - NetoutputL2)$$

$$\frac{\partial NetinputL2}{\partial NetoutputH2} = w8 = -0.1$$

$$\frac{\partial EoutputL1}{\partial NetoutputH2} = \frac{\partial EoutputL1}{\partial NetinputL1} * \frac{\partial NetinputL1}{\partial NetinputH2}$$

$$\frac{\partial EoutputL1}{\partial NetinputL1} = \frac{\partial EoutputL1}{\partial NetoutputL1} * \frac{\partial EoutputL1}{\partial NetinputL1}$$

$$\frac{\partial EoutputL1}{\partial NetoutputL1} = -(Target1 - NetoutputL1)$$

$$\frac{\partial EoutputL1}{\partial NetinputL1} = NetoutputL1 * (1 - NetoutputL1)$$

$$\frac{\partial NetinputL1}{\partial NetinputH2} = w7 = 0.5$$

$$\frac{\partial NetinputH2}{\partial w2} = x1 = 2$$

$$\frac{\partial NetoutputH2}{\partial NetinputH2} = NetoutputH2 * (1 - NetoutputH2)$$

$$w2 = \frac{\partial Etotal}{\partial w2}$$

$$\frac{\partial Etotal}{\partial NetoutputH1} = \frac{\partial EoutputL1}{\partial NetoutputH1} + \frac{\partial EoutputL2}{\partial NetoutputH1}$$

$$\frac{\partial EoutputL1}{\partial NetoutputH1} = \frac{\partial EoutputL1}{\partial NetinputL1} * \frac{\partial NetinputL1}{\partial NetoutputH1}$$

$$\frac{\partial EoutputL1}{\partial NetinputL1} = \frac{\partial EoutputL1}{\partial NetoutputL1} * \frac{\partial EoutputL1}{\partial NetinputL1}$$

$$\frac{\partial EoutputL1}{\partial NetoutputL1} = -(Target1 - NetoutputL1)$$

$$\frac{\partial EoutputL1}{\partial NetinputL1} = NetoutputL1 * (1 - NetoutputL1)$$

$$\frac{\partial NetinputL1}{\partial NetoutputH1} = w5 = 0.7$$

$$\frac{\partial EoutputL2}{\partial NetoutputH1} = \frac{\partial EoutputL2}{\partial NetinputL2} * \frac{\partial NetinputL2}{\partial NetinputH1}$$

$$\frac{\partial EoutputL2}{\partial NetinputL2} = \frac{\partial EoutputL2}{\partial NetoutputL2} * \frac{\partial EoutputL2}{\partial NetinputL2}$$

$$\frac{\partial EoutputL2}{\partial NetoutputL2} = -(Target2 - NetoutputL2)$$

$$\frac{\partial EoutputL2}{\partial NetinputL2} = NetoutputL2 * (1 - NetoutputL2)$$

$$\frac{\partial NetinputL2}{\partial NetinputH1} = w6 = -0.3$$

$$\frac{\partial NetinputH1}{\partial w3} = x2 = 3$$

$$\frac{\partial NetoutputH1}{\partial NetinputH1} = NetoutputH1 * (1 - NetoutputH1)$$

$$w3 = \frac{\partial Etotal}{\partial w3}$$

$$\frac{\partial Etotal}{\partial NetoutputH2} = \frac{\partial EoutputL2}{\partial NetoutputH2} + \frac{\partial EoutputL1}{\partial NetoutputH2}$$

$$\frac{\partial EoutputL2}{\partial NetoutputH2} = \frac{\partial EoutputL2}{\partial NetinputL2} * \frac{\partial NetinputL2}{\partial NetoutputH2}$$

$$\frac{\partial EoutputL2}{\partial NetinputL2} = \frac{\partial EoutputL2}{\partial NetoutputL2} * \frac{\partial EoutputL2}{\partial NetinputL2}$$

$$\frac{\partial EoutputL2}{\partial NetoutputL2} = -(Target2 - NetoutputL2)$$

$$\frac{\partial EoutputL2}{\partial NetinputL2} = NetoutputL2 * (1 - NetoutputL2)$$

$$\frac{\partial NetinputL2}{\partial NetoutputH2} = w8 = -0.1$$

$$\frac{\partial EoutputL1}{\partial NetoutputH2} = \frac{\partial EoutputL1}{\partial NetinputL1} * \frac{\partial NetinputL1}{\partial NetinputH2}$$

$$\frac{\partial EoutputL1}{\partial NetinputL1} = \frac{\partial EoutputL1}{\partial NetoutputL1} * \frac{\partial EoutputL1}{\partial NetinputL1}$$

$$\frac{\partial EoutputL1}{\partial NetoutputL1} = -(Target1 - NetoutputL1)$$

$$\frac{\partial EoutputL1}{\partial NetinputL1} = NetoutputL1 * (1 - NetoutputL1)$$

$$\frac{\partial NetinputL1}{\partial NetinputH2} = w7 = 0.5$$

$$\frac{\partial NetinputH2}{\partial w4} = x2 = 3$$

$$\frac{\partial NetoutputH2}{\partial NetinputH2} = NetoutputH2 * (1 - NetoutputH2)$$

$$w4 = \frac{\partial Etotal}{\partial w4}$$

$$\frac{\partial Etotal}{\partial NetoutputL1} = -(\text{Target1} - \text{NetoutputL1})$$
$$\frac{\partial NetinputL1}{\partial w5} = NetoutputH1 = \frac{1}{1+e^{-NetinputH1}}$$
$$\frac{\partial NetoutputL1}{\partial NetinputL1} = \text{NetoutputL1} * (1 - \text{NetoutputL1})$$
$$w5 = \frac{\partial Etotal}{\partial w5}$$

$$\frac{\partial Etotal}{\partial NetoutputL2} = -(\text{Target2} - \text{NetoutputL2})$$
$$\frac{\partial NetinputL2}{\partial w6} = NetoutputH1 = \frac{1}{1+e^{-NetinputH1}}$$
$$\frac{\partial NetoutputL2}{\partial NetinputL2} = \text{NetoutputL2} * (1 - \text{NetoutputL2})$$
$$w6 = \frac{\partial Etotal}{\partial w6}$$

$$\frac{\partial Etotal}{\partial NetoutputL1} = -(\text{Target1} - \text{NetoutputL1})$$
$$\frac{\partial NetinputL1}{\partial w7} = NetoutputH2 = \frac{1}{1+e^{-NetinputH2}}$$
$$\frac{\partial NetoutputL1}{\partial NetinputL1} = \text{NetoutputL1} * (1 - \text{NetoutputL1})$$
$$w7 = \frac{\partial Etotal}{\partial w7}$$

$$\frac{\partial Etotal}{\partial NetoutputL2} = -(\text{Target2} - \text{NetoutputL2})$$
$$\frac{\partial NetinputL2}{\partial w8} = NetoutputH2 = \frac{1}{1+e^{-NetinputH2}}$$
$$\frac{\partial NetoutputL2}{\partial NetinputL2} = \text{NetoutputL2} * (1 - \text{NetoutputL2})$$
$$w8 = \frac{\partial Etotal}{\partial w8}$$

En regardant la Fig. 8, l'on se rend compte que dans ce réseau de neurones, NetoutputH1 et NetoutputH2 influencent L1 et donc $E_{outputL1}$, mais aussi L2 (et donc $E_{output2}$). C'est ainsi que pour W1, W2, W3 et W4, nous nous sommes intéressé aux dérivées de E_{total} par rapport respectivement à NetoutputL1 et à NetoutputL2.

En revanche, nos dérivées partielles valent:

W1= $\frac{\partial Etotal}{\partial w1}$ = (((- (Target1-NetoutputL1) * NetoutputL1 * (1-NetoutputL1)) * W5) + ((- (Target2-NetoutputL2) * NetoutputL2 * (1-NetoutputL2)) * W6)) * x_1 * NetoutputH1 * (1-NetoutputH1)

= (((- (1- 0.712257432) * 0.712257432 * (1-0.712257432)) * 0.7) + ((- (0.2 – 0.533097574) * 0.533097574 * (1 - 0.533097574)) * (-0.3))) * **2** * (0.413382421) * (1 - 0.413382421)

= (-0.290008845) * **2** * (0.242497395)

= **-0.140652779**

W2= $\frac{\partial Etotal}{\partial w2}$ = (((- (Target2-NetoutputL2) * NetoutputL2 * (1-NetoutputL2)) * W8) + ((- (Target1-NetoutputL1) * NetoutputL1 * (1-NetoutputL1)) * W7)) * x_1 * NetoutputH2 * (1-NetoutputH2)

= (((- (0.2 - 0.533097574) * 0.533097574* (1-0.533097574)) * (-0.1)) + ((- (1- 0.712257432) * 0.712257432 * (1 - 0.712257432)) * 0.5)) * **2** * (0.934010991) * (1 - 0.934010991)

= (-0.037776907) * **2** * (0.06163446)

= **-0.004656719**

W3 = $\frac{\partial Etotal}{\partial w3}$ = (((- (Target1-NetoutputL1) * NetoutputL1 * (1-NetoutputL1)) * W5) + ((- (Target2-NetoutputL2) * NetoutputL2 * (1-NetoutputL2)) * W6)) * x_2 * NetoutputH1 * (1-NetoutputH1)

= (((- (1- 0.712257432) * 0.712257432 * (1-0.712257432)) * 0.7) + ((- (0.2 – 0.533097574) * 0.533097574 * (1 - 0.533097574)) * (-0.3))) * **3** * (0.413382421) * (1 - 0.413382421)

= (-0.290008845) * **3** * (0.242497395)

= **-0.210979168**

W4 = $\frac{\partial Etotal}{\partial w4}$ = (((- (Target2-NetoutputL2) * NetoutputL2 * (1-NetoutputL2)) * W8) + ((- (Target1-NetoutputL1) * NetoutputL1 * (1-NetoutputL1)) * W7)) * x_2 * NetoutputH2 * (1-NetoutputH2)

= (((- (0.2 - 0.533097574) * 0.533097574* (1-0.533097574)) * (-0.1)) + ((- (1- 0.712257432) * 0.712257432 * (1 - 0.712257432)) * 0.5)) * **3** * (0.934010991) * (1 - 0.934010991)

= (-0.037776907) * **3** * (0.06163446)

= **-0.006985078**

W5 = $\frac{\partial Etotal}{\partial w5}$ = (- (Target1-NetoutputL1)) * $\frac{1}{1+e^{-NetinputH1}}$ * NetoutputL1 * (1-NetoutputL1)

= (- (1 - 0.712257432)) * **0.413382421** * 0.712257432* (1 - 0.712257432)

= **-0.024377952**

W6 = $\frac{\partial Etotal}{\partial w6}$ = (- (Target2-NetoutputL2)) * $\frac{1}{1+e^{-NetinputH1}}$ * NetoutputL2 * (1-NetoutputL2)

= (- (0.2 - 0.533097574)) * **0.413382421** * 0.533097574* (1 - 0.533097574)

= **0.034273331**

W7 = $\frac{\partial Etotal}{\partial w7}$ = (- (Target1-NetoutputL1)) * $\frac{1}{1+e^{-NetinputH2}}$ * NetoutputL1 * (1-NetoutputL1)

= (- (1 - 0.712257432)) * **0.934010991** * 0.712257432* (1 - 0.712257432)

= **-0.055080415**

W8 = $\frac{\partial Etotal}{\partial w8}$ = (- (Target2-NetoutputL2)) * $\frac{1}{1+e^{-NetinputH2}}$ * NetoutputL2 * (1-NetoutputL2)

= (- (0.2 - 0.533097574)) * **0.934010991** * 0.533097574 * (1 - 0.533097574)

= **0.077438386**

Après avoir obtenu les dérivées partielles de l'erreur totale par rapport à chaque poids, nous allons maintenant mettre à jour nos poids. Nous le savons, pour mettre à jour les poids, on utilise cette formule:

$$W_i^+ = Wi - \eta * \frac{\partial Etotal}{\partial W_i}$$

avec η le taux d'apprentissage, Wi le poids à l'état actuel, et W_i^+ le nouveau poids mise à jour.

Ainsi,

W_1^+ = 0.3 – 0.1 * (-0.140652779)

= **0.314065278**

W_2^+ = 0.2 – 0.1 * (-0.004656719)

= **0.200465672**

W_3^+ = (-0.4) – 0.1 * (-0.210979168)

= **-0.378902083**

W_4^+ = 0.6 – 0.1 * (-0.006985078)

= **0.600698508**

W_5^+ = 0.7 – 0.1 * (-0.024377952)

= **0.702437795**

W_6^+ = (-0.3) – 0.1 * (0.034273331)

= **-0.303427333**

W_7^+ = 0.5 – 0.1 * (-0.055080415)

= **0.505508042**

W_8^+ = (-0.1) – 0.1 * (0.077438386)

= **-0.107743839**

En remplaçant ces nouveaux poids dans le réseau, nous sommes passé à l'Epoch 2 où nous avons obtenu ce Forward Pass:

Epoch 2

Forward Pass

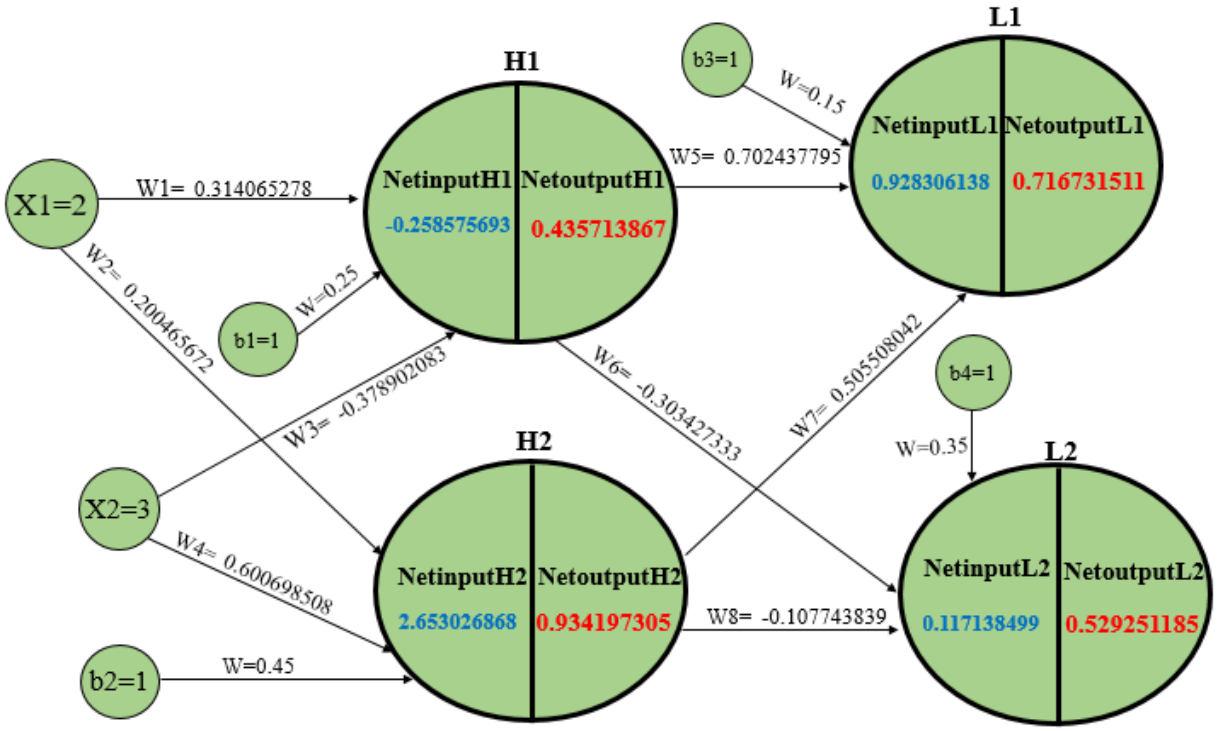

Fig. 9. Forward Pass Epoch 2

Le constat est que NetoutputL1 est toujours différent du target 1. En fait, 0.716731511 < > 1 et NetoutputL2 (0.529251185) différent de target 2 (0.2). Avec la même formule de l'erreur, nous obtenons 0.09432369. Les résultats des nouveaux poids, après l'Epoch 1, nous ont montré que le réseau neuronal a appris un peu. En effet, lorsque nous avons remplacé ces poids à l'Epoch 2, nous nous sommes rendu compte que les résultats ont évolué positivement. En examinant les valeurs des targets attendus (target1 = 1; target2 = 0.2), et les sorties prédites (NetoutputL1 = 0.712257432; NetoutputL2 = 0.533097574) de l'Epoch 1, nous avons remarqué que les sorties (NetoutputL1 = 0.716731511; NetoutputL2 = 0.529251185) de Forward Pass dans l'Epoch 2 commencent à se rapprocher des valeurs cibles. En fait, le NetoutputL1 de l'Epoch 2 a augmenté et le NetoutputL2 a diminué. En outre, l'erreur de l'Epoch 2 (0.09432369) est inférieure à celle de l'Epoch 1 (0.096874889). Sur ce, nous allons faire le Backward Pass de l'Epoch 2 afin d'ajuster encore nos poids:

Backward Pass

W1 = $\frac{\partial Etotal}{\partial w1}$ = **-0.0321**

W2 = $\frac{\partial Etotal}{\partial w2}$ = **-0.00466**

W3 = $\frac{\partial Etotal}{\partial w3}$ = **-0.04815697**

W4 = $\frac{\partial Etotal}{\partial w4}$ = **-0.007**

W5 = $\frac{\partial Etotal}{\partial w5}$ = **-0.025**

W6 = $\frac{\partial Etotal}{\partial w6}$ = **0.035742**

W7 = $\frac{\partial Etotal}{\partial w7}$ = **-0.0537268824**

W8 = $\frac{\partial Etotal}{\partial w8}$ = **0.076633**

Ainsi,

W_1^+ = 0.314065278 – 0.1 * (-0.0321)

= **0.317275278**

W_2^+ = 0.200465672 – 0.1 * (-0.00466)

= **0.200931672**

W_3^+ = (-0.378902083) – 0.1 * (-0.04815697)

= **-0.374086386**

W_4^+ = 0.600698508 – 0.1 * (-0.007)

= **0.601398508**

W_5^+ = 0.702437795 – 0.1 * (-0.025)

= **0.704937795**

W_6^+ = (-0.303427333) – 0.1 * (0.035742)

= **-0.307001533**

W_7^+ = 0.505508042 – 0.1 * (-0.0537268824)

= **0.51088073024**

W_8^+ = (-0.107743839) – 0.1 * (0.076633)

= **-0.115407139**

En remplaçant les nouveaux poids de l'Epoch 2 dans le Forward Pass de l'Epoch 3, nous constatons que nous n'obtenons pas encore des prédictions proches des targets cibles. Ainsi, nous continuons à itérer jusqu'à ce que nous arrivions à l'Epoch qui aura pour prédictions des valeurs proches de target 1 et target 2. Poursuivant les itérations jusqu'à l'Epoch 925, nous nous sommes rendu compte que notre machine a enfin appris. En fait, elle a pu trouver des paramètres du modèle qui minimisent la fonction coût (L'erreur). C'est cela l'essentiel même du Machine Learning.

On pourrait par exemple développer un algorithme qui tente au hasard plusieurs combinaisons des paramètres, et qui retient la combinaison avec la Fonction Coût la plus faible. C'est un peu comme organiser un concours d'archers pour ne garder que le meilleur. Cette stratégie est cependant assez inefficace la plupart du temps. Notre effort a été de considérer la Fonction Coût comme une fonction **convexe**, n'ayant qu'un seul minimum. Ce minimum a été trouver via l'algorithme de minimisation appelé **Gradient Descent**, en français **Descente de gradient.**

Il est clair qu'il fallait entrainer notre modèle jusqu'à 925 Epoch car la fonction coût qui a été, à l'Epoch 1, 0.096874889, a pu être minimisée à 0.001600727, à la phase Forward Pass de l'Epoch 926. Ainsi, le NetoutputL1, qui est égal à 0.943418607,

de cette dernière itération est proche du target 1 qui vaut 1 et le NetoutputL2, qui vaut 0.200000245, est proche du target 2 qui a été donné à 0.2. Ce qui fait que le réseau a pu ajuster les paramètres Durant toutes ses itérations pour enfin trouver des bons paramètres pour pouvoir généraliser le modèle. Ces paramètres sont: W_1 = 0.744985268; W_2 = 0.314552231; W_3 = 0.267478622; W_4 = 0.771832141; W_5 = 1.422782196; W_6 = -0.908934577; W_7 = 1.389947119; W_8 = -0.923640014.

5 CONCLUSION

L'apprentissage par descente de gradient permet d'optimiser les paramètres du réseau de neurones par rapport à la fonction d'erreur. Toutefois, signalons que l'algorithme de rétro-propagation dans sa forme basic, utilisant la technique de descente du gradient, n'est pas très efficace car elle utilise peu d'information sur la réduction de l'erreur. Ainsi, il existe d'autres algorithmes puissants tels que le Gradient descendant avec un taux d'apprentissage variable; l'Algorithme de Quasi-Newton; la Rétro-propagation résiliente; l'Algorithme du gradient conjugué; Algorithme de Levenberg-Marquardt; l'Algorithme de Fletcher-Reeves; etc [16].

Avec l'apprentissage, le but c'est de pouvoir généraliser le modèle après l'avoir entrainé. Cette généralisation se fait après validation du modèle. La généralisation concerne la tâche réalisée par le réseau de neurones une fois on finit la phase d'apprentissage du modèle. On peut ainsi évaluer cette généralisation en testant le réseau sur des données qui n'ont pas servi à l'apprentissage. En général, la force d'un modèle de réseau de neurones, pour qu'il soit généralisable, dépend des facteurs notamment de l'algorithme d'apprentissage, c'est-à-dire de son aptitude à optimiser; de la complexité de l'échantillon: le nombre d'exemples; et de la complexité du réseau (nombre de poids, nombre d'Epoch et des couches cachées).

Et, au cas où les données observées sont immenses, on introduit la notion de batch qui, à chaque itération, un ensemble d'observations est présenté au réseau pour moyenner les gradients et mettre à jour des poids de façon rapide. Toutefois, il faudra veiller à la saturation du réseau car si les poids prennent des grandes valeurs, les sorties deviendront grandes et se rapprocheront de la zone de saturation de la fonction d'activation. Ce qui nécessite le choix d'un pas de correction petit et l'initialisation des poids à de très petites valeurs pour éviter le problème de sur-apprentissage (overfitting).

En définitive, nous tenons à préciser qu'il existe deux modes de fonctionnement dans un réseau de neurones. Le premier mode consiste à ajuster les paramètres du réseau grâce à la présentation d'exemples et des variables pour lesquels on connaît la réponse désirée, le target. Ce mode est appelé phase d'apprentissage. Le second mode fonctionne de manière à exploiter le réseau tout en lui présentant des données inconnues. C'est le mode appelé phase de reconnaissance ou phase de généralisation. Dans les exemples proposés dans cet article, nous avons utilisé le premier mode. Toutefois, ces deux modes peuvent être théoriquement entrelacées. Il ne s'agit pas, de ce fait, de phases distinctes mais plutôt de modes de fonctionnement [12].

REMERCIEMENTS

Ces quelques mots sont l'expression de notre gratitude à tous ceux qui nous permis d'obtenir les matériels nécessaires pour que cet article se réalise. Nous disowns merci de façon particulière à Nsenge Mpia Héritier, initiateur de ce projet qui vise à éclaircir certains concepts de Backprogagation à des termes simples. En dépit de ses occupations comme Enseignant à l'Université de l'Assomption, comme étudiant en Théologie, et comme Doctorant en Systèmes d'Informations, il a pu se consacrer à cette recherche qui aiderait nombreuses personnes qui s'interésseraient au comment fornctionne l'algorithme de retro-propagation de gradient.

Nous tenons à renmercier l'assistant Inipaivudu Baelani Nephtali qui a collaboré dans la mise au point de ce projet, ses contributions dans la verification de certains résultats ont été profondement bénéfiques.

REFERENCES

[1] A. Cornuéjols, L. Miclet, and V. Barra, Apprentissage artificiel. Deep learning, concepts et algorithmes, Paris, Eyrolles, 2018.

[2] C. Hardy, Contribution au développement de l'apprentissage profond dans les systèmes distribués, Thèse de Doctorat en Informatique, Université Rennes 1, Rennes, 2019.

[3] C.A. Azencott, Introduction au Machine Learning, Dunod, Paris, 2018.

[4] C.E. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, Activation Functions: Comparison of Trends in Practice and Research for Deep Learning, arXiv Prepr arXiv181103378, 2018.

[5] F. Chollet, Deep Learning with Python, Manning Publications Co, Shelter Island, 2018.

[6] F.C. Ribeiro, R.T. Carvalho, P.C. Cortez, V.H.C. De Albuquerque, and P.R. Filho, "Binary Neural Networks for Classification of Voice Commands from Throat Microphone, " IEEE Access, vol. 6, pp. 70130-70144, 2018.

[7] G. Saint-Cirgue, Apprendre le Machine Learning en une semaine, Machine Learnia, Londres, 2019.

[8] I.H. Witten, E. Frank, and M.A Hall, Data Mining: Practical Machine Learning Tools and Techniques, 3rd edition, Morgan Kaufmann, Massachusetts, 2011.

[9] Jouannic, Thibault, Deep learning: La rétropropagation du gradient, 2017. .

[10] Online] Available: https://www.miximum.fr/blog/introduction-au-deep-learning-2 (December 10, 2020).

[11] M. Kordos and W. Duch, "Variable step search mlp training method," International Journal of Information Technology and Intelligent Computing, pp.45-56, 2006.

[12] M.T.H. Rakotomalala Andrianandrasana, Réseau de neurones artificiels et application des séries temporelles, Mémoire du Diplôme d'Études Approfondies de Mathématiques, Inédit, Université d'Antananarivo, 2015.

[13] P. Comon, "Réseau de neurones. Classification supervisée par réseaux multicouches, " Traitement du Signal, vol. 8, no. 6, pp. 387-407, 1992.

[14] R. Asaad and I.A. Rasan, "Back Propagation Neural Network and Sigmoid Activation Function in Multi-Layer Networks, " Academic Journal of Nawroz University, pp. 1-6, 2019.

[15] S. Haykin, Neural Networks and Learning Machines, 3rd edition, Pearson education, New Jersey, 2009.

[16] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 315-323, 2011.

[17] Y. Djeriri, Les réseaux de neurones artificiels, UDL-SBA, Sidi Bel Abbès, 2017.