Introduction au Data Mining
Réseaux de Neurones

Devoir Maison

Cours « Modélisation Paramétrique, Filtrage Optimal et Adaptatif »

Pascal SCALART (pascal.scalart@enssat.fr)

Ecole Nationale Supérieure des Sciences Appliquées et Technologie

Spécialité Electronique– 3ème Année

Introduction
aux réseaux de neurones

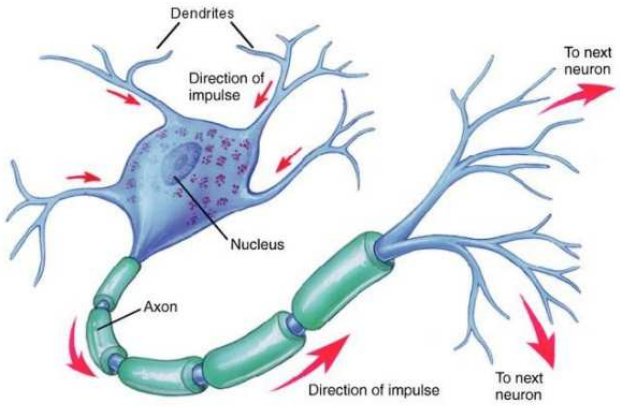
Propos Introductif

Qu'il soit utilisés par Google pour identifier les numéros de rues apparaissant sur les images captées par les voitures *Street View*; qu'il soit utilisé dans le projet *PlaNet* pour construire un système automatique capable reconnaitre le lieu où une photo a été prise (sans aucune information de géolocalisation) qu'il s'agisse d'un paysage, d'une rue ou d'un monument; qu'il soit utilisé pour battre l'intelligence humaine avec la victoire en janvier 2016 d'AlphaGo sur le champion du monde du jeu de go (l'un des jeux de stratégie les plus anciens et compliqués du monde) ; qu'il soit utilisé pour reconnaître et catégoriser les images (visages, objets, etc.) à la volée au sein de flux vidéos, le réseau de neurones profond comportant entre 2 à 10 milliards de neurones artificiels (comparé aux 100 milliards de neurones du cerveau humain) s'impose de jour en jour dans le traitement massif de données.

En effet, durant ces dix dernières années, l'apprentissage automatique est à l'origine des véhicules sans pilote, du gain d'efficacité de la recherche sur Internet et des percées considérables dans notre compréhension du génome humain.

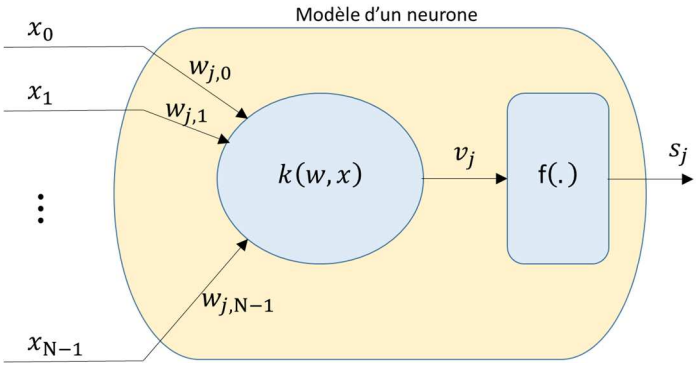
1. Modélisation d'un neurone

Dans les réseaux de neurones, l'élément de base est un neurone artificiel qui correspond à un modèle issu des observations faites sur les neurones biologiques.



Le modèle utilisé pour « copier » un neurone biologique est le suivant. Il comporte :

- un certain nombre de signaux d'entrée : $x_0, \dots, x_{N-1}$
- des poids de connexions : $w_{j,0}, \dots, w_{j,L-1}$
- une fonction d'activation : $v = k(w, x)$
- une fonction de transition : f
- un état de sortie : $s_j = f(v_j)$.



La valeur numérique du poids associé à une connexion entre deux unités reflète la force de la relation existant entre ces 2 unités. Si cette valeur est positive, la connexion est dite ***excitatrice*** et, si elle est négative, elle est dite ***exhibitrice***.

Il existe différents modèles de neurones suivant les choix effectués pour les fonctions d'activation et de transition. On distingue :

- le neurone « produit scalaire » : $k(w, x) = \langle w, x \rangle = \sum_{i=0}^{N-1} w_{j,i}\, x_i$
  Les réseaux de neurones ainsi obtenus sont nommés : ***MLP (Muti-Layer Perceptron)***
- le neurone « distance » : $k(w, x) = \|w - x\| = \sqrt{\sum_{i=0}^{N-1} (w_{j,i} - x_i)^2}$
  Les réseaux de neurones ainsi obtenus sont nommés : ***RBF (Radial-Based Function)***

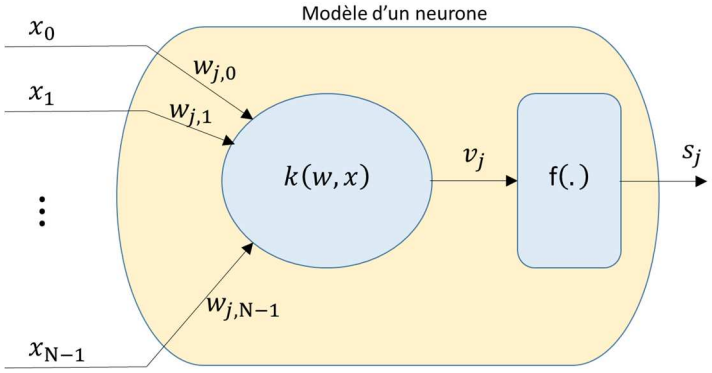
Ces deux neurones sont liés à travers la relation : $\|w - x\|^2 = \|w\|^2 + \|x\|^2 - 2\langle w, x \rangle$

2. Le perceptron et sa version multi-couche

Nous considérons tout d'abord un type bien particulier de neurones, ***le perceptron***, dont les caractéristiques sont les suivantes :

- la sortie v_j de la fonction d'activation n'interagit pas avec les entrées. Il n'y a donc pas de rebouclage de la sortie vers les entrées (structure feedforward).

- la fonction d'activation $k(w,x)$ est de type « produit scalaire » et donc elle effectue une opération linéaire sur les entrées.

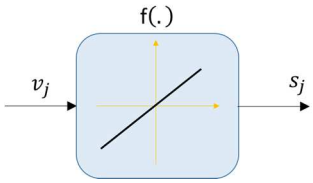


2.1. Le perceptron : fonctions de transition classiques

On distingue trois cas principaux décrits ci–après. Lorsque la sortie s_j est positive, on dit que le ***neurone est actif***. Dans le cas contraire, il est dit inactif.

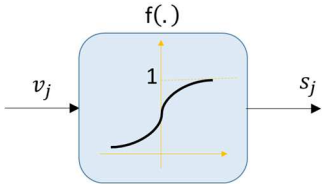
- la fonction de transition « linaire » :

$$s_j = f(v_j) = k.v_j \text{ avec } k \in \mathbb{R}$$



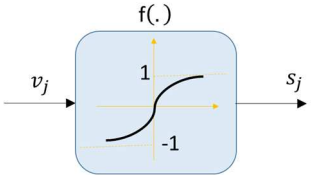
- la fonction de transition type « sigmoïde » ou « logistique » :

$$s_j = f(v_j) = \frac{1}{1-\exp(-v_j)}$$



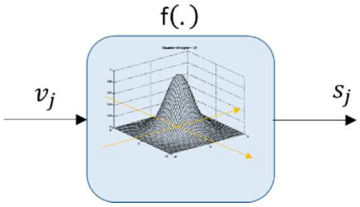
- la fonction « tangente hyperbolique » ou « bipolaire » :

$$s_j = f(v_j) = \frac{\exp(v_j)-\exp(-v_j)}{\exp(v_j)+\exp(-v_j)}$$



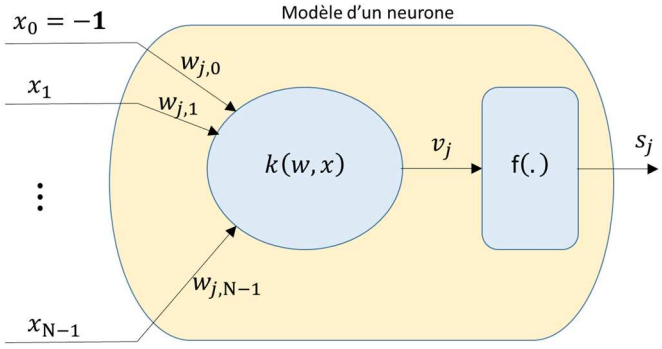
- la « fonction de base radiale »pour les réseaux RBF :

$$s_j = f(v_j) = \exp\left(-\frac{v_j^2}{2\sigma^2}\right)$$



2.2. Le perceptron : rôle de l'unité de biais

Parmi les différentes entrées de la fonction d'activation, on ajoute une entrée qui sera fixée à -1, dite unité de biais, qui permet d'introduire un terme de biais au niveau de chaque neurone Note : on peut également la fixer à +1.



Lorsque l'on choisit un neurone de type « produit scalaire » (car on travaille avec un neurone de type perceptron), on obtient alors en sortie de la fonction d'activation :

$$v_j = \sum_{k=0}^{N-1} w_{j,k}\, x_k \Rightarrow v_j = -w_{j,0} + \sum_{k=1}^{N-1} w_{j,k}\, x_k$$

Nous avons vu que le neurone n°j sera actif ssi sa sortie s_j est positive. Ainsi, le seuil d'activation (ou de non-activation) du neurone n°j sera déterminé par la condition :

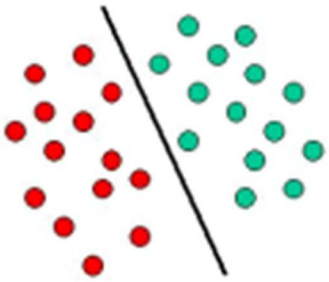
$$v_j = 0 \Rightarrow w_{j,0} = \sum_{k=1}^{N-1} w_{j,k}\, x_k$$

At donc ***l'unité de biais permet de fixer le seuil d'activité du neurone*** j.

Note : dans l'espace $\mathbb{R}^{N-1}$, l'équation $\sum_{k=1}^{N-1} w_{j,k}\, x_k = 0$ définit un hyperplan de dimension $N-2$, il s'agit de l'hyperplan de séparation lorsque $w_{j,0} = 0$ entre l'activation et la non-activation du neurone j. Ainsi la relation :

$$w_{j,0} = \sum_{k=1}^{N-1} w_{j,k}\, x_k$$

réalise un déplacement de cet hyperplan et de faire en sorte qu'il ne passe plus par le point origine.



2.3. L'erreur du perceptron : une fonction multidimensionelle

Reprenons le schéma du neurone n°j de type « produit scalaire » vu antérieurement. Si l'on suppose que la fonction de transition f est fixée, la sortie du neurone ne va dépendre que de la connaissance des poids des N connexions $\{w_{j,k}, k \in [\![0, N-1]\!]\}$. Dans ce cas, si l'on suppose que le neurone devrait produire « idéalement » la réponse y_j pour un vecteur d'entrée $\{x_i, i \in [\![0, N-1]\!]\}$ connu, alors l'erreur quadratique (ou fonction de coût) commise par le neurone est donnée par :

$$\mathcal{E} = \frac{1}{2}\, e_j^2$$

$$\text{avec l'erreur} \qquad e_j = y_j - f\left(\sum_{k=0}^{N-1} w_{j,k}\, x_i\right)$$

Nous retiendrons pour la suite que cette fonction de coût est donc une fonction de la variable multidimensionelle des poids des N connexions :

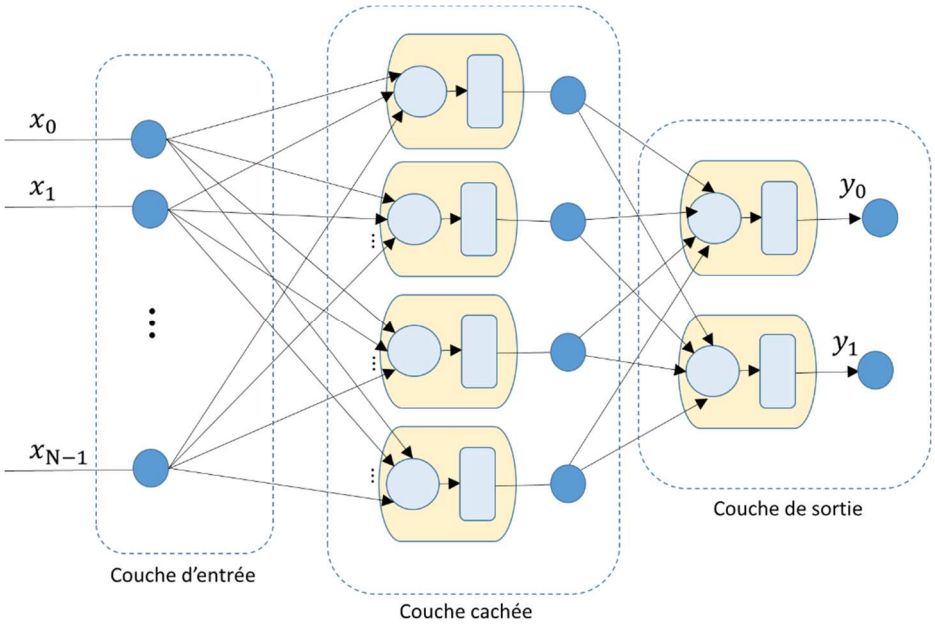
$$\mathcal{E} \ : \quad \mathbb{R}^N \longrightarrow \mathbb{R}$$

$$\boldsymbol{w}_j = \begin{pmatrix} w_{j,0} \\ \vdots \\ w_{j,N-1} \end{pmatrix} \longmapsto \mathcal{E}(\boldsymbol{w}_j) = \frac{1}{2}\left[y_j - f\left(\sum_{k=0}^{N-1} w_{j,k}\, x_k\right)\right]^2$$

2.4. Le perceptron multi-couches

Un ***réseau de neurones*** (ou réseau connexionniste) est un groupe orienté, constitué d'un ensemble d'unités simples (des neurones formels), réalisant des calculs élémentaires, structurées en couches successives capables d'échanger de l'information au moyen de connexions qui les relient. Il se caractérise par :

- son architecture,
- les fonctions (activation, transition) de ses éléments.



Le vecteur représentant les sorties du réseau $\boldsymbol{y} = \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}$ est obtenu, à partir du vecteur $\boldsymbol{x} = \begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix}$ représentant les signaux en entrée du réseau, suivant la relation : $\boldsymbol{y} = \Psi(\boldsymbol{w}, \boldsymbol{x})$. On distingue 3 phases décrivant 3 fonctionnements différents du réseau :

- la ***phase d'apprentissage*** visant à « apprendre » la relation $\Psi(.)$ liant les vecteurs $\boldsymbol{y}$ et $\boldsymbol{x}$,
- la ***phase de validation*** visant à vérifier que la relation $\Psi(.)$ apprise à l'étape précédente est pertinente (donc on connait la valeur des sorties escomptées),
- la ***phase de test*** visant à appliquer le réseau de neurones à de nouvelles entrées mais dont en ne connaît pas a priori la valeur des sorties correspondantes.

Il existe 2 techniques majeures afin de réaliser la phase d'apprentissage :

- l'apprentissage ***supervisé*** où l'on dispose d'exemples en entrée et de leurs sorties correspondantes,
- l'apprentissage ***non-supervisé*** où l'on ne dispose qu'uniquement des exemples en entrée (ou base d'apprentissage) mais pas de leurs sorties correspondantes. Il faudra alors estimer ces sorties.

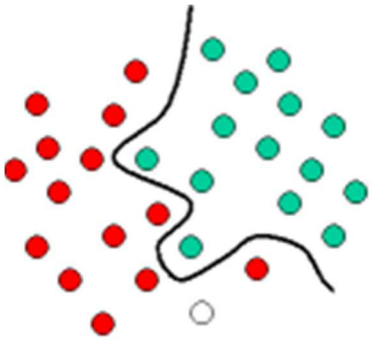
Les différents éléments du réseau pouvant être modifiés sont :

- la structure du réseau (nombre de connexions entrants à chaque neurones), nombre de couches cachées, etc…
- le type des fonctions d'activation, et de transition du réseau,
- la valeur du poids des connexions entre neurones.

On suppose ici qu'un choix a été fait quant à la topologie du réseau c'est-à-dire que les 2 premiers points ci-dessus (nombre d'entrées, nombre de sorties, nombre de neurones sur la couche cachées, et nombre de couches cachées) sont déjà spécifiés. Nous ne travaillerons qu'avec un type bien particulier de réseau de neurones, ***le perceptron multi-couche (PMC) ou Multi-Layer Perceptron (MLP)***, dont les caractéristiques sont les suivantes :

- le réseau est organisé en plusieurs couches,
- une couche = un groupe de neurones uniformes sans connexion les uns avec les autres,
- il comporte au moins 2 couches (une couche dite "cachée" et une couche de sortie),
- une couche au moins n'est pas linéaire,
- les dimensions d'entrée L_{in} et de sortie L_{out} peuvent être différentes,
- une couche ne peut pas utiliser que les sorties des couches précédentes (structure feedforward).

Théorème de Cybenko : un réseau à 2 couches (informations en entrée -> couche cachée -> couche de sortie) où chaque neurone de la ***couche cachée*** possède une fonction d'activation de ***type sigmoïde***, et où chaque neurone de la ***couche de sortie*** possède une ***fonction linéaire*** comme fonction d'activation, permet d'approximer n'importe quelle fonction continue comme fonction de séparation.



3. L'apprentissage supervisé

Une fois sélectionnée la topologie du réseau, il ne nous reste donc qu'à décrire le type d'apprentissage à réaliser : supervisé ou non supervisé. Dans l'apprentissage supervisé, on suppose que l'on dispose d'un 'oracle, ou d'un 'expert' ou bien encore d'un 'superviseur' qui va être à même de déterminer la valeur que doivent prendre les sorties $\boldsymbol{y}$ du réseau de neurones aux différentes entrées $\boldsymbol{x}(n)$ de la base d'apprentissage.
La tâche d'apprentissage consiste alors à chercher un moyen (c'est-à-dire un algorithme) pour apprendre automatiquement la valeur des poids optimaux $\boldsymbol{w}$ du réseau et satisfaire les 2 objectifs majeurs :

- (i) que les poids donnent de bons résultats lors de l'apprentissage à partir de K exemples connus :

$$\{\boldsymbol{y}(n) = \Psi(\boldsymbol{w}, \boldsymbol{x}(n))\}_{n\in[\![1,K]\!]}$$

- (ii) que les poids ainsi appris possèdent de bonnes ***capacités de généralisation*** $\boldsymbol{y} = \Psi(\boldsymbol{w}, \boldsymbol{x})$ à de nouveaux signaux $\boldsymbol{x}$ en entrées n'ayant pas appartenu à la base d'apprentissage.

Il s'agit alors d'éviter les situations de ***sur-apprentissage*** c'est-à-dire les cas particuliers où l'objectif (i) serait parfaitement atteint, mais les poids $\boldsymbol{w}$ ainsi appris seraient trop spécifiques de la base d'apprentissage et auraient donc trop faible pouvoir de généralisation. Nous verrons ultérieurement quelles sont les procédures à notre disposition afin d'éviter cette situation.

3.1. Choix de la base d'apprentissage

On pourrait penser que l'apprentissage est un processus passif devant les données, c'est-à-dire n'ayant pas d'initiative dans le choix des données d'apprentissage. Pourtant, comme au jeu de Mastermind, il est souvent plus intéressant de sélectionner les questions posées plutôt de les prendre au hasard. En apprentissage automatique c'est également la même chose, les exemples d'apprentissage ***doivent être le plus informatif*** *possible* pour qu'ils soient utiles (on parle alors d'entropie en théorie de l'information.

Exercice 1 – Enregistrement de la base de données d'apprentissage

Générer vos fichier.wav qui serviront à la phase d'apprentissage. Pour cela :

(a) installer/utiliser le logiciel **Audacity** et dans le menu **> Edition >Préférences…>Périphériques** configurer le choix **Canaux : 1(Mono)**

(b) sélectionner **> Fichier > Nouveau** puis appuyer sur le bouton enregistrement (le 'rouge') pour procéder à l'enregistrement d'un des 3 chiffres '1','2' ou '3'

(c) avec la souris, sélectionner la partie inutile de silence s'étendant du début de l'enregistrement jusqu'au début du signal utile numérisé. Supprimer cette partie à l'aide de la commande **Suppr**. Eviter à tout prix les saturations du signal.

(d) Réutiliser plusieurs fois la commande **Suppr** en début/fin de signal pour ne sélectionner qu'uniquement les 150 ms de signal utile les plus représentatives. Vous pourrez utiliser **Ctrl+E** pour zoomer sur la partie sélectionnée par la souris.

(e) une fois que subsiste environ 150 ms de signal utile , normaliser en amplitude le signal à partir de l'onglet **> Effets > Normaliser**… et sélectionner **Normaliser l'amplitude maximale à : -1dB.**

(f) Ecouter une dernière fois votre signal pour vérifier que le son est « intègre » et que vous n'avez pas trop supprimé de signal. Enregistrer le fichier à l'aide de l'onglet > **Fichier > Exporter la sélection…** en utilisant le format **WAV signé 16 bits PCM** et en précisant votre répertoire de travail.

(g) recommencer les étapes (b) à (f) pour effectuer un autre enregistrement.

Note : Vous allez créer successivement un fichier représentant un '1', puis un fichier représentant un '2', puis un fichier représentant un '3', etc….et vous répéterez cette opération 20 fois ce qui fera au total $3 \times 20 = 60$ fichiers dans la base d'apprentissage. Vous appelez ces

fichiers *BDD_1_1.wav*, *BDD_2_1.wav*, *BDD_3_1.wav*, …, *BDD_1_20.wav*, *BDD_2_20.wav*, *BDD_3_20.wav*. Le premier chiffre désigne le type de chiffre prononcé et le second chiffre, le numéro du fichier. De cette manière, si vous souhaitez ultérieurement ajouter un autre chiffre à votre base de données, il sera aisé de le faire sans apporter de gros changements dans vos programmes.

Par ailleurs, la théorie de l'apprentissage repose sur l'hypothèse que les données sont ***indépendantes et identiquement distribuées*** (i.i.d.), tant au cours de l'apprentissage, qu'après, en phase de test et de décision. C'est pourquoi, dans notre base d'apprentissage, nous intégrons exactement le même nombre de fichiers de '1', de '2' et de '3'.

3.2. Choix des attributs de la base d'apprentissage

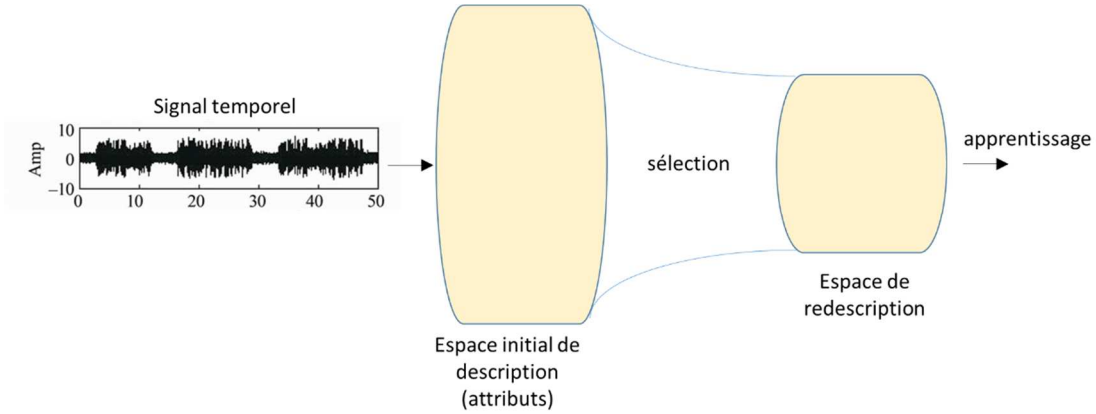
L'apprentissage s'appuie sur des données (des objets) qu'il faut représenter dans un espace de représentation des données. Cette description se fait généralement suivant 2 directions :

- La description des données à partir des ***échantillons***
- La description des données par un ensemble d'***attribus.*** On peut par exemple d'écrire un son à partir de son énergie, sa fréquence, sa durée…

La réduction de la dimension de l'espace d'entrée vise à rendre l'espace de représentation mieux adapté à la tâche d'apprentissage et de rendre ainsi l'induction possible alors même que la taille de l'échantillon d'apprentissage la rend problématique. En effet, un ensemble de descripteurs réduits peut conduire à l'obtention de résultats d'apprentissage plus simples et plus aisés à interpréter. Il faut alors porter attention :

- ***au choix des attributs*** utilisés pour effectuer la description des données. Il faut supprimer les attributs non pertinents, et analyser les corrélations/dépendances entre attributs. Une solution simple consiste à sélectionner un sous-ensemble de descripteurs au sein des descripteurs d'origine.
- ***à réduire la dimension*** de l'espace d'entrée. En effet, le nombre d'hypothèses possibles (c'est-à-dire les combinaisons possibles des poids du réseau) est fonction du nombre de dimensions de l'espace d'entrée. Et donc, plus la dimension est grande, et plus grand doit être le nombre d'exemples pour garantir en probabilité un lien entre le risque empirique et risque réel. C'est ce que l'on appelle la **malédiction de la dimensionnalité** : le nombre de dimensions de l'espace tend à nuire à la recherche de régularités dans les données.

Un cas extrême est celui dans lequel les descripteurs sont bien plus nombreux que les exemples. On retrouve ce cas dans l'analyse du génome, les textes ou images sur Internet.



Dans notre projet, chaque fichier de la base de données représente environ 150 ms de parole numérisée à la fréquence 44.1kHz soit $L \cong 6615$ échantillons qui constituent l'espace initial de représentation. La réduction de la dimensionnalité est effectuée à partir des étapes suivantes :

- Estimation de la DSP (densité spectrale de puissance) du signal à l'aide de la *méthode du périodogramme fenêtré (cf. Elec2)*
  - fenêtrage du signal à l'aide d'une fenêtre de Hamming de L échantillons
  - transformation fréquentielle du signal temporel ainsi fenêtré $\{x(n), n = 0, \dots, Nfft - 1\}$ à l'aide d'une DFT réalisée sur $Nfft = 8192$ points.
  - Estimation de la DSP du signal : calcul du carré du module des composantes de la DFT.

$$\hat{S}_{xx}(k) = \frac{1}{Nfft}|X(k)|^2 \qquad , k = 0,1,2, \dots, Nfft - 1$$

- Conservation des $1 + Nfft/2$ premières composantes de la DSP $\hat{S}_{xx}(k)$ (car les autres composantes se déduisent par symétrie hermitienne car le signal $x(n)$ est à valeurs dans $\mathbb{R}$).

Exercice 2 – Sélection des attributs de la base de données d'apprentissage

Réaliser un programme indépendant qui lit séquentiellement les $3 \times 20 = 60$ fichiers audio de la base d'apprentissage et qui extrait de chaque fichier les ***attributs*** (encore appelés ***descripteurs***) de ce signaux. Pour cela :

(a) Réaliser une double boucle for...end qui lit séquentiellement les 60 fichiers de la base d'apprentissage

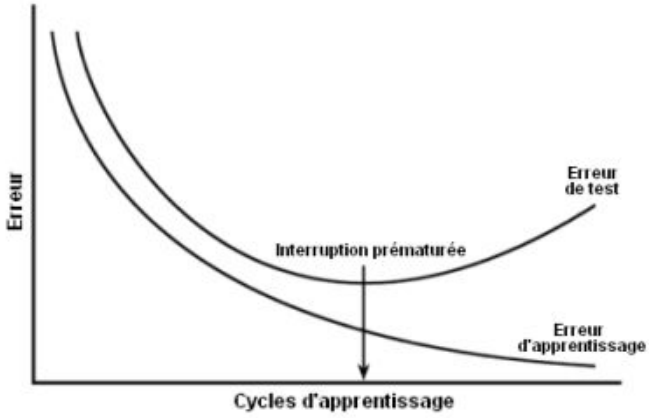
```
>> for numfich = 1:20
>>   for typeson = 1:3
>>     name=['BDD_',num2str(typeson),'_',num2str(numfich),'.wav'];
>>   [data,fs,Nbits] = wavread(name);% lecture fichier wav
>>   ...
>>   end
>> end
```

(b) sur chaque fichier, effectuer les différentes opérations mentionnées ci-dessus (estimation de la DSP, puis stockage des premières composantes). Stocker ces composantes dans un tableau bidimensionnel (soit donc une matrice que l'on nommera `Attributs`) de dimension 60×4097.

(c) sauvegarder les paramètres `Nfft` et `Attributs` dans un fichier de données au format ®matlab en utilisant l'instruction « `save` »

3.3. L'évaluation des performances de l'apprentissage

On considère ici un apprentissage supervisé à partir d'un échantillon d'apprentissage. Comment évaluer la performance d'une règle de classification (au sens large) obtenue à l'aide d'un l'algorithme ? Suffit-il de faire confiance au principe de minimisation du risque et de se fonder sur la performance mesurée sur la base d'apprentissage ? Un phénomène classique, appelé ***surapprentissage***, (schématisé ci-dessous) est que le risque empririque diminue au fur et à mesure que le système prend en compte davantage d'informations (soit par un accroissement du nombre d'exemples présentés, soit par une répétition des exemples d'apprentissage) tandis que le risque réel (évalué sur la base de test), d'abord décroissant, se met à augmenter après un certain stade.



Un algorithme d'apprentissage sélectionne une règle qui minimise le risque empirique (i.e. sur base apprentissage). Cependant, ***le risque réel est souvent considéré comme le critère de performance le plus important du système d'apprentissage***. Il est donc essentiel de pouvoir l'estimer de façon précise ce qui nécessite d'utiliser au mieux l'unique élément à notre décision : les exemples de la base d'apprentissage.

Le problème fréquent est que le nombre d'exemples d'apprentissage est souvent limité pour ne pas dire réduit. Plusieurs stratégies peuvent alors être envisagées :

- utiliser toutes données d'apprentissage à la fois pour l'apprentissage et pour l'estimation de la performance (i.e. le test). C'est la ***méthode de resubstuition*** qui conduit souvent à du ***surapprentissage*** et qui est en général optimiste (i.e. conduit à une valeur estimée de la performance qui sera plus faible que la valeur réelle). On dit qu'il y a un biais.
- Pour éviter ce biais, on opère une distinction entre les données utilisées pour l'apprentissage et les données utilisées pour le test. Il faut alors utiliser une méthode de validation. On distingue :
    - méthode de régularisation, méthode d'élagage
    - La validation croisée. Différentes méthodes peuvent être envisagées : *Testset validation*, *HoldOut method*, ou bien *leave-one-out cross validation*

Idéalement, nous devrions donc constituer deux autres bases : une base de validation et une base de test qui devront suivre la même distribution que l'échantillon d'apprentissage. Pour notre projet, nous allons uniquement constituer une base de test.

Exercice 3 – Enregistrement de la base de données de test

Générer 15 fichiers de la phase de test au format « .wav ». Pour cela, procéder comme précédemment (cf. exercice n°1) pour enregistrer de nouveaux fichiers correspondants à l'un des 3 chiffres '1','2' ou '3' . Il faudra que la base de test suive la même statistique que la base d'apprentissage, dans la base de test il y aura donc une même fréquence d'apparition des 3 chiffres '1','2' ou '3'. Eviter à tout prix les saturations du signal.

Vous allez créer successivement trois fichiers : l'un représentant un '1', puis un fichier représentant un '2', puis un fichier représentant un '3', etc....et vous répéterez cette opération 5 fois pour constituer l'ensemble des fichiers de la base de test. Vous appelez ces fichiers *TEST_1_1.wav*, *TEST_2_1.wav*, *TEST_3_1.wav* et ceci jusqu'aux derniers fichiers *TEST_1_5.wav*, *TEST_2_5.wav* , *TEST_3_5.wav* .

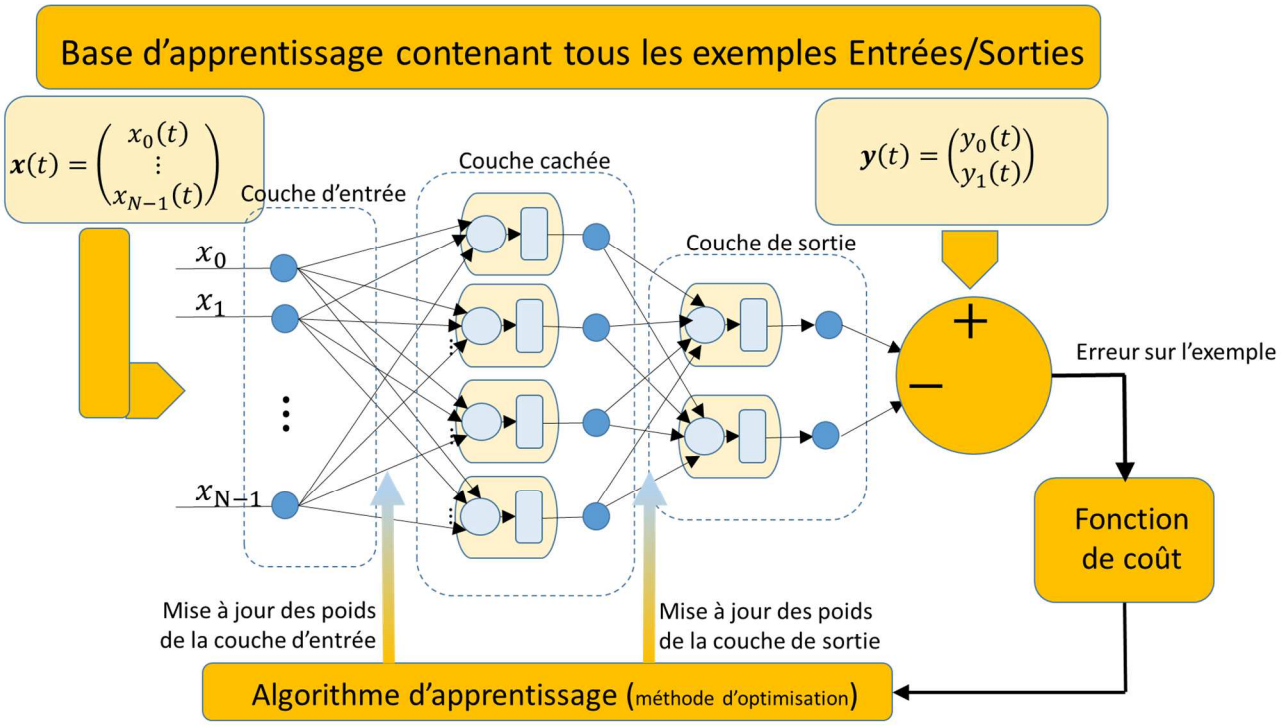
4. Algorithme d'apprentissage

Le principe général de l'apprentissage supervisé est résumé sur la figure ci-dessous. Il consiste, pour tous les instants $t = 0,1,2,..$, à présenter successivement à l'entrée du réseau de neurones un vecteur d'entrée $\boldsymbol{x}(t)$ dont on connaît la valeur attendue en sortie du réseau $\boldsymbol{y}(t)$. Par exemple, dans le cas d'une ***problématique de classification*** de plusieurs unités vocales élémentaires (« un », « deux », « trois »,..), on présentera en entrée du réseau les attributs relatifs aux unités vocales élémentaires et l'on voudra que les sorties à valeurs binaires (0 ou 1) soient activées à 1 lorsque l'unité vocale présente en entrée correspond à une classe donnée (et soient inactives dans le cas contraire). Si l'on souhaite ainsi discriminer les 9 unités vocales « un », « deux »,..., « neuf » et bien on choisira 9 sorties sur la couche de sortie.

En supposant que la couche de sortie est de dimension L_{out}, nous considèrerons une ***fonction de coût*** donnée par les moindres carrés pondérés (avec une fenêtre rectangulaire de pondération) suivante :

$$\mathcal{E}(t) = \frac{1}{L_{out}} \sum_{m=0}^{L_{out}} \frac{1}{2}\, e_m^2(t) \qquad \text{avec l'erreur} \qquad e_m(t) = y_m(t) - \hat{y}_m(t)$$

où les $\{\hat{y}_m(t), \in [\![0, L_{out} - 1]\!]\}$ correspondent aux sorties du réseau de neurones lorsque le vecteur $x(t)$ (décrivant les attributs su signal) est présenté à l'entrée du réseau, et les $\{y_m(t), m \in [\![0, L_{out} - 1]\!]\}$ sont les valeurs désirées en sortie. Dans le cas où l'on a $L_{out} = 2$ neurones sur la couche de sortie, l'ensemble du système d'apprentissage peut être schématisé suivant :



3.2. Cas du perceptron simple : Comment trouver les poids du réseau pour bien approcher les données ?

Comme vu précédemment au paragraphe 2.3, la fonction de coût associée au perceptron (une seule sortie, pas de couche cachée) est une fonction multidimensionnelle des N poids des connexions :

$$\mathcal{E} \ : \quad \mathbb{R}^N \longrightarrow \mathbb{R}$$

$$\boldsymbol{w}_j = \begin{pmatrix} w_{j,0} \\ \vdots \\ w_{j,N-1} \end{pmatrix} \longmapsto \mathcal{E}(\boldsymbol{w}_j) = \frac{1}{2}\left[y_j - f\left(\sum_{k=0}^{N-1} w_{j,k}\, x_k\right)\right]^2$$

Dans la phase d'apprentissage, les poids des connexions vont également varier dans le temps, on a donc :

$$\mathcal{E} \ : \quad \mathbb{R}^N \longrightarrow \mathbb{R}$$

$$\boldsymbol{w}_j(t) = \begin{pmatrix} w_{j,0}(t) \\ \vdots \\ w_{j,N-1}(t) \end{pmatrix} \longmapsto \mathcal{E}(\boldsymbol{w}_j, t) = \frac{1}{2}\left[y_j(t) - f\left(\sum_{k=0}^{N-1} w_{j,k}(t)\, x_k(t)\right)\right]^2$$

On souhaite trouver les poids optimaux de manière adaptative c'est-à-dire en incrémentant à chaque itération t les poids $\{w_{j,k}(t), k \in [\![0, N-1]\!]\}$ du neurone n°j selon :

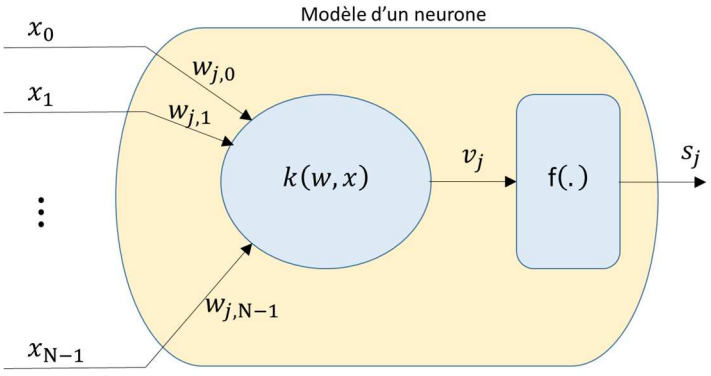
$$\boxed{w_{j,k}(t+1) = w_{j,k}(t) + \Delta w_{j,k}} \quad \text{pour } t = 0,1,2,\ldots \text{ et } \ k \in [\![0, N-1]\!]$$

où $\Delta w_{j,k}$ représente l'incrément du poids $w_{j,k}$ de la connexion entre les instants t et $t+1$. Si l'on considère les N poids des connexions, cette relation s'écrit sous forme vectorielle :

$$\begin{pmatrix} w_{j,0}(t+1) \\ \vdots \\ w_{j,N-1}(t+1) \end{pmatrix} = \begin{pmatrix} w_{j,0}(t) \\ \vdots \\ w_{j,N-1}(t) \end{pmatrix} + \begin{pmatrix} \Delta w_{j,0} \\ \vdots \\ \Delta w_{j,N-1} \end{pmatrix} \quad \text{pour} \quad t = 0,1,2,\ldots$$

$$\Leftrightarrow \ \boldsymbol{w}_j(t+1) = \boldsymbol{w}_j(t) + \Delta \boldsymbol{w}_j$$

Où la notation en gras indique que les variables sont maintenant multidimensionnelles, car il s'agit de vecteurs de dimension N. On souhaite que l'incrément $\Delta \boldsymbol{w}_j$ minimise la fonction de coût $\mathcal{E}$.



En considérant l'approximation au premier ordre de $\mathcal{E}$ donnée par son développement de Taylor-Young au 1er ordre (pour les fonctions de plusieurs variables) :

$$\mathcal{E}\left(\boldsymbol{w}_j(t+1)\right) = \mathcal{E}\left(\boldsymbol{w}_j(t) + \Delta \boldsymbol{w}_j\right)$$
$$= \mathcal{E}\left(\boldsymbol{w}_j(t)\right) + \langle \Delta \boldsymbol{w}_j, \nabla \mathcal{E}\left(\boldsymbol{w}_j(t)\right)\rangle + o\left(\Delta \boldsymbol{w}_j\right)$$

où le reste de Taylor et le produit scalaire possèdent les propriétés suivantes :

$$\lim_{\|\Delta \boldsymbol{w}_j\| \to \boldsymbol{0}} o\left(\Delta \boldsymbol{w}_{j,k}\right) = 0 \qquad \text{et} \qquad \langle \boldsymbol{u}, \mathbf{v} \rangle = \sum_{k=0}^{N-1} u(k) v(k)$$

On cherche à minimiser la fonction coût $\mathcal{E}$ soit, entre les instants t et t+1, à faire en sorte que la différence

$$\mathcal{E}\left(\boldsymbol{w}_j(t+1)\right) - \mathcal{E}\left(\boldsymbol{w}_j(t)\right) = \langle \Delta \boldsymbol{w}_j, \nabla \mathcal{E}\left(\boldsymbol{w}_j(t)\right)\rangle + o\left(\Delta \boldsymbol{w}_{j,k}\right)$$

soit la plus négative possible (décroissance de la fonction de coût). En supposant que l'incrément est faible soit lorsque $\|\Delta \boldsymbol{w}_j\| \to 0$, on a alors $\lim_{\|\Delta \boldsymbol{w}_j\| \to 0} o\left(\Delta \boldsymbol{w}_{j,k}\right) = 0$, et donc :

$$\lim_{\|\Delta w_j\| \to 0} \mathcal{E}\left(\boldsymbol{w}_j(t+1)\right) - \mathcal{E}\left(\boldsymbol{w}_j(t)\right) = \langle \Delta \boldsymbol{w}_j, \nabla \mathcal{E}\left(\boldsymbol{w}_j(t)\right)\rangle$$

Cette différence peut encore s'exprimer en fonction du produit scalaire entre les 2 vecteurs $\langle \Delta \boldsymbol{w}_j, \nabla \mathcal{E}\left(\boldsymbol{w}_j(t)\right)\rangle$ et l'on sait que ***le produit scalaire sera minimisé lorsque les 2 vecteurs seront colinéaires et en opposition*** soit :

$$\boxed{\Delta \boldsymbol{w}_j = -\mu \ \nabla \mathcal{E}\left(\boldsymbol{w}_j(t)\right)} \quad \text{avec} \quad \mu \in \mathbb{R}^{+*}$$

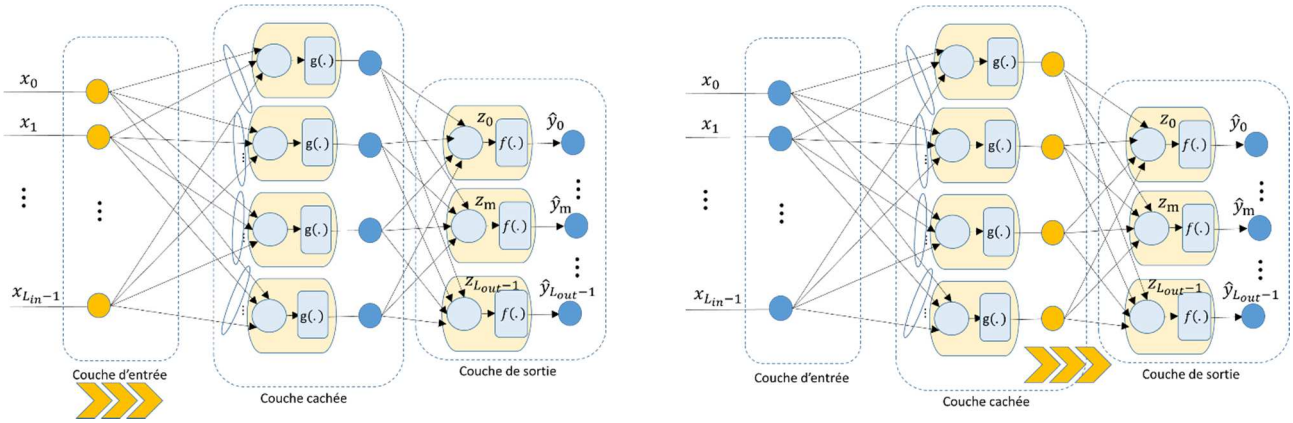
Il faut donc que l'incrément soit un vecteur opposé au gradient de la fonction de coût. L'algorithme d'adaptation s'obtient à l'aide des 2 formules précédemment encadrées :

$$\boxed{\boldsymbol{w}_j(t+1) = \boldsymbol{w}_j(t) - \mu \ \nabla \mathcal{E}\left(\boldsymbol{w}_j(t)\right)} \quad \text{pour} \quad t = 0,1,2,\ldots$$

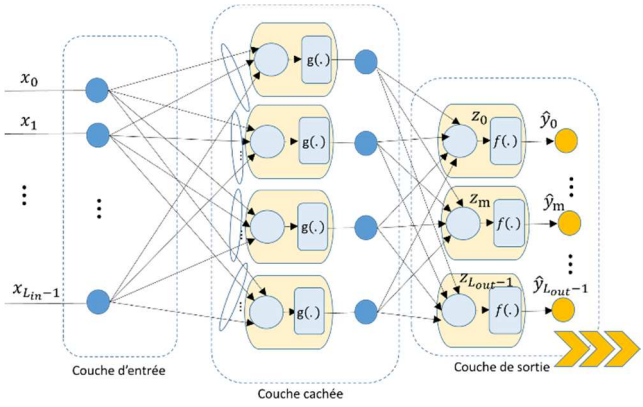
Ce qui s'écrit encore suivant :

$$\boxed{\begin{pmatrix} w_{j,0}(t+1) \\ \vdots \\ w_{j,N-1}(t+1) \end{pmatrix} = \begin{pmatrix} w_{j,0}(t) \\ \vdots \\ w_{j,N-1}(t) \end{pmatrix} - \mu \begin{pmatrix} \frac{\partial \mathcal{E}}{\partial w_{j,0}(t)} \\ \vdots \\ \frac{\partial \mathcal{E}}{\partial w_{j,N-1}(t)} \end{pmatrix}} \quad \text{pour} \quad t = 0,1,2,\ldots$$

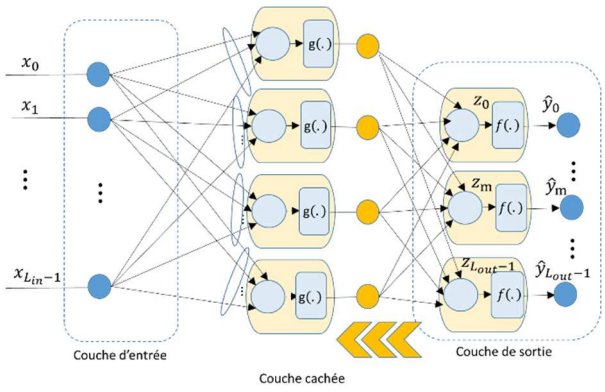
Cet algorithme est appelé ***algorithme du gradient***, ou bien encore ***algorithme de descente maximale*** (en raison de la décroissance maximale de la fonction de côut). La constante positive μ est appelée ***le pas de l'algorithme***, ou bien encore ***le taux d'apprentissage***. Dans le contexte des réseaux de neurones, cet algorithme est encore appelé ***algorithme de rétropropagation des erreurs*** (ou backpropagation).



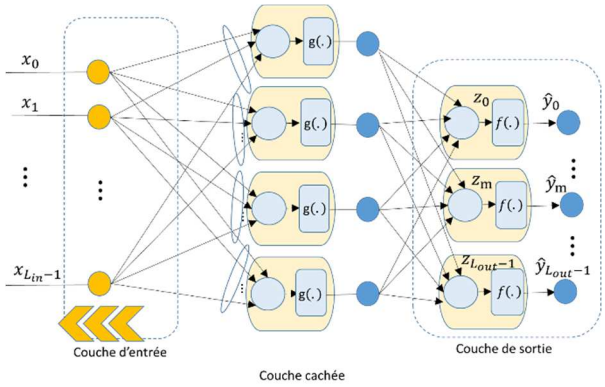
(a) propagation de l'activité entrée-> cachée (b) propagation de l'activité cachées->sortie



(b) calcul de de l'erreur



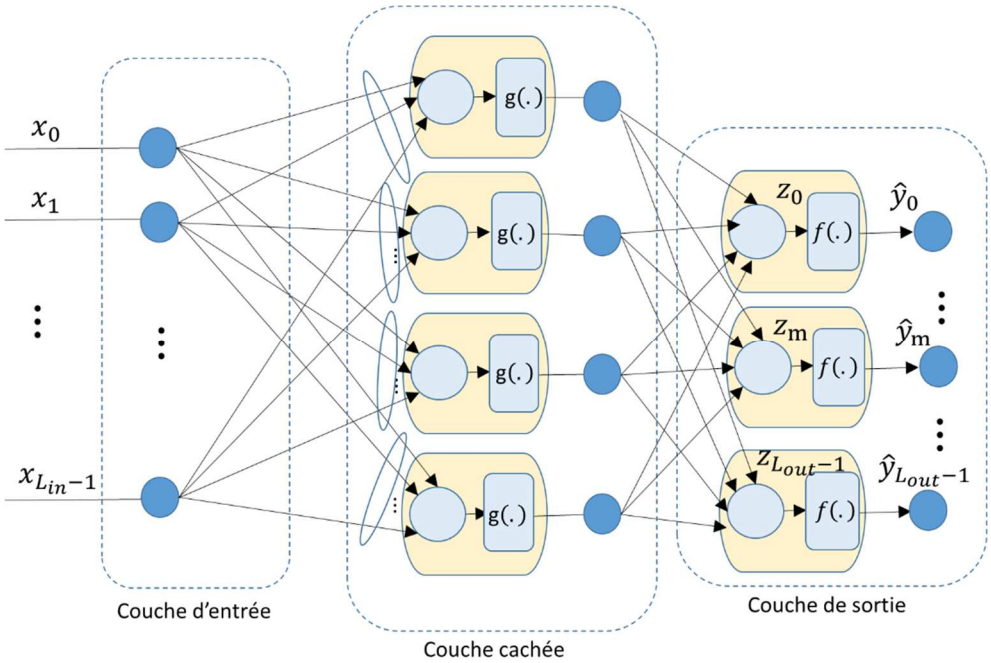
(d) retro-propagation de l'erreur (couche sortie)



(e)) retro-propagation de l'erreur (couche cachée)

3.3. Cas d'un perceptron multicouche : calcul du vecteur gradient

Considérons le réseau de neurones, de type perceptron multicouche, donné à la figure suivante pour lequel les fonctions de transitions sont $f(.)$ et $g(.)$ pour respectivement la couche de sortie et la couche cachée. Ces fonctions sont supposées différentiables indéfiniment (cf. paragraphe 2.1).



Afin de clarifier les notations considérons les connexions entre les valeurs en entrée, le neurone n°j sur la couche cachée ainsi que le neurone n°m sur la couche de sortie. On a donc les éléments suivants :

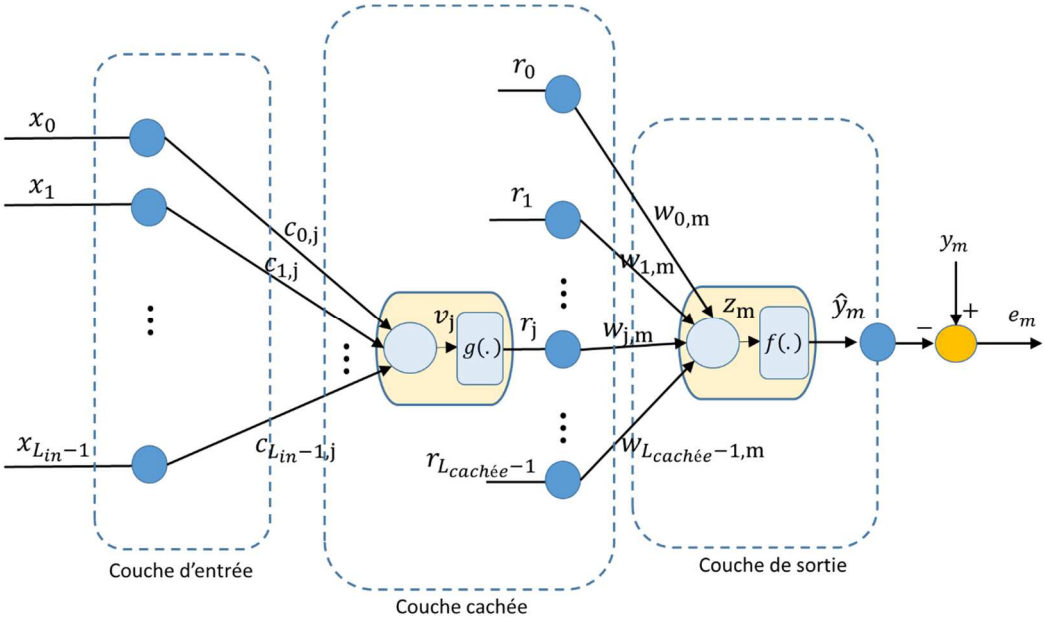


Figure : Schéma de base pour le calcul de la mise à jour des poids de sortie et ceux de la couche cachée

<u>Mise à jour des poids de la couche de sortie</u> :

Conformément à l'analyse menée précédemment, la mise à jour des $L_{out} \times L_{cachée}$ poids de la couche de sortie est effectuée selon :

$$\begin{pmatrix} w_{j,0}(t+1) \\ \vdots \\ w_{j,L_{out}-1}(t+1) \end{pmatrix} = \begin{pmatrix} w_{j,0}(t) \\ \vdots \\ w_{j,L_{out}-1}(t) \end{pmatrix} - \mu \begin{pmatrix} \frac{\partial \mathcal{E}}{\partial w_{j,0}(t)} \\ \vdots \\ \frac{\partial \mathcal{E}}{\partial w_{j,L_{out}-1}(t)} \end{pmatrix} \quad \text{pour} \quad t = 0,1,2,\dots$$

On peut ainsi écrire (cf. figure ci-dessus) :

$\forall j_0 \in [\![0, L_{cachée}-1]\!], \forall m_0 \in [\![0, L_{out}-1]\!]$,

$$\begin{aligned}
\frac{\partial\, \mathcal{E}}{\partial w_{j_0,m_0}} &= \frac{\partial}{\partial w_{j_0,m_0}} \left(\frac{1}{L_{out}} \sum_{m=0}^{L_{out}-1} \frac{1}{2}\, e_m^2 \right) \\
&= \frac{1}{L_{out}} \sum_{m=0}^{L_{out}-1} \frac{1}{2}\, \frac{\partial}{\partial w_{j_0,m_0}} e_m^2 \\
&= \frac{1}{L_{out}} \sum_{m=0}^{L_{out}-1} e_m\, \frac{\partial\, e_m}{\partial w_{j_0,m_0}} \\
&= \frac{1}{L_{out}} \sum_{m=0}^{L_{out}-1} e_m\, \frac{\partial\, (y_m - \hat{y}_m)}{\partial w_{j_0,m_0}} \\
&= \frac{-1}{L_{out}} \sum_{m=0}^{L_{out}-1} e_m\, \frac{\partial\, \hat{y}_m}{\partial w_{j_0,m_0}} \\
&= \frac{-1}{L_{out}} \sum_{m=0}^{L_{out}-1} e_m\, \frac{\partial\, f(z_m)}{\partial w_{j_0,m_0}} \\
&= \frac{-1}{L_{out}} \sum_{m=0}^{L_{out}-1} e_m\, f'(z_m)\, \frac{\partial\, z_m}{\partial w_{j_0,m_0}}
\end{aligned}$$

$$= \frac{-1}{L_{out}} e_{m_0} f'(z_{m_0}) r_{j_0}$$

Car on a : $\frac{\partial z_m}{\partial w_{j_0,m_0}} = \frac{\partial}{\partial w_{j_0,m_0}} \left(\sum_{j=0}^{L_{cachée}-1} w_{j,m} \ \ r_j \right) = \delta_{m,m_0} \ r_{j_0}$

$$\boxed{\Rightarrow \frac{\partial \mathcal{E}}{\partial w_{j_0,m_0}} = \frac{-1}{L_{out}} e_{m_0} f'(z_{m_0}) r_{j_0}}$$

et ***donc l'équation de mise à jour des poids de la couche de sortie*** est donnée par :

En notant $\delta_m^{(out)} = e_m f'(z_m)$
on obtient l'équation :

$$w_{j,m}(t+1) = w_{j,m}(t) + \frac{\mu}{L_{out}} \delta_m^{(out)} r_j \qquad \begin{matrix} \forall j = 0,1,\dots,L_{cachée}-1 \\ \forall m = 0,1,\dots,L_{out}-1 \end{matrix} \qquad (1)$$

<u>Mise à jour des poids de la couche cachée</u> :

Conformément à l'analyse menée précédemment (cf. figure ci-dessus), la mise à jour des poids de la couche de sortie est effectuée selon :

$$\begin{pmatrix} c_{k,0}(t+1) \\ \vdots \\ c_{k,L_{cachée}-1}(t+1) \end{pmatrix} = \begin{pmatrix} c_{k,0}(t) \\ \vdots \\ c_{k,L_{cachée}-1}(t) \end{pmatrix} - \mu \begin{pmatrix} \frac{\partial \mathcal{E}}{\partial c_{k,0}(t)} \\ \vdots \\ \frac{\partial \mathcal{E}}{\partial c_{k,L_{cachée}-1}(t)} \end{pmatrix} \quad \text{pour} \quad t = 0,1,2,\dots$$

Il nous reste à évaluer l'équation de mise à jour des poids de la couche cachée et donc il nous faut calculer $\forall j_0 \in [\![0, L_{cachée}-1]\!]$ et $\forall k_0 \in [\![0, L_{in}-1]\!]$:

$$\frac{\partial \mathcal{E}}{\partial c_{k_0,j_0}} = \frac{\partial}{\partial c_{k_0,j_0}} \left(\frac{1}{L_{out}} \sum_{m=0}^{L_{out}-1} \frac{1}{2} e_m^2 \right)$$

$$= \frac{1}{L_{out}} \sum_{m=0}^{L_{out}-1} \frac{1}{2} \frac{\partial}{\partial c_{k_0,j_0}} e_m^2$$

$$= \frac{1}{L_{out}} \sum_{m=0}^{L_{out}-1} e_m \frac{\partial e_m}{\partial c_{k_0,j_0}}$$

$$= \frac{1}{L_{out}} \sum_{m=0}^{L_{out}-1} e_m \frac{\partial (y_m - \hat{y}_m)}{\partial c_{k_0,j_0}}$$

$$\Rightarrow \boxed{\frac{\partial \mathcal{E}}{\partial c_{k_0,j_0}} = \frac{-1}{L_{out}} \sum_{m=0}^{L_{out}-1} e_m \frac{\partial \hat{y}_m}{\partial c_{k_0,j_0}}}$$

Or on peut écrire (en suivant bien attentivement les différentes étapes sur la figure précédente) :

$$\frac{\partial \hat{y}_m}{\partial c_{k_0,j_0}} = \frac{\partial \hat{y}_m}{\partial z_m} \frac{\partial z_m}{\partial c_{k_0,j_0}}$$

$$= \frac{\partial f(z_m)}{\partial z_m} \frac{\partial z_m}{\partial c_{k_0,j_0}}$$

$$= f'(z_m) \frac{\partial (w_{j_0,m} \, r_{j_0})}{\partial c_{k_0,j_0}}$$

$$= f'(z_m) \ w_{j_0,m} \frac{\partial g(v_{j_0})}{\partial c_{k_0,j_0}}$$

$$= f'(z_m)\, w_{j_0,m}\, g'(\, v_{j_0}) \frac{\partial\, v_{j_0}}{\partial c_{k_0,j_0}}$$

$$\Rightarrow \boxed{\frac{\partial\, \hat{y}_m}{\partial c_{k_0,j_0}} = f'(z_m)\, w_{j_0,m}\, g'(\, v_{j_0})\, x_{k_0}}$$

Car on a : $\frac{\partial\, v_{j_0}}{\partial c_{k_0,j_0}} = \frac{\partial}{\partial c_{k_0,j_0}} \left(\sum_{k=0}^{L_{in}-1} c_{k,j}\ x_k\right) = \delta_{j,j_0}\ x_{k_0}$

Soit en contractant les 2 relations précédentes encadrées, on obtient :

$$\Rightarrow \frac{\partial\, \mathcal{E}}{\partial c_{k_0,j_0}} = \frac{-1}{L_{out}} g'(\, v_{j_0})\, x_{k_0} \left(\sum_{m=0}^{L_{out}-1} e_m\, f'(z_m)\, w_{j_0,m}\right)$$

et ***donc l'équation de mise à jour des poids de la couche cachée*** est donnée par :

En notant : $\delta_j^{(cachée)} = g'(\, v_j) \underbrace{\left(\sum_{m=0}^{L_{out}-1} \delta_m^{(out)}\, w_{j,m}\right)}_{caclul\ n°1 = à\ calculer\ à\ part}$

on obtient l'équation :

$$c_{k,j}(t+1) = c_{k,j}(t) + \frac{\mu}{L_{out}}\ x_k\ \delta_j^{(cachée)} \qquad \begin{matrix} \forall j = 0,1,\dots, L_{cachée} - 1 \\ \forall k = 0,1,\dots, L_{in} - 1 \end{matrix} \qquad (2)$$

Vous devrez donc programmer les équations (1) et (2) au cours de l'exercice ci-dessous. Si vous voulez avoir une programmation efficace sous ®matlab, vous devrez à tout prix programmer au plus haut niveau possible (vecteurs ! et mieux matrices !). Et surtout éviter les boucles imbriquées qui calculent à partir d'échantillons individuels comme l'exemple suivant où l'on met en évidence le terme multiplicatif $g'(\, v_j)\, x_k$ dans la mise à jour donnée par la relation (2)

```
>> tic
>> calcul = ...   % on fait le calcul n°1 indiqué au sein de la formule (2)
>>
>> for neurone = 1 : Lcachee
>>   for entree = 1 : Lin
>>  PoidsCaches(entree,neurone)=...*gprime(neurone)*x_input(entree)*calcul(neurone)
>>   end
>> end
>> toc
```

En utilisant directement un vecteur `x_input`, les calculs seront plus efficaces. Le vérifier avec les instructions ®matlab « `tic` » et « `toc` »

```
>> tic
>> calcul = ...   % on fait le calcul n°1 indiqué au sein de la formule (2)
>>
>> for neurone = 1 : Lcachee
>>      PoidsCaches( :,neurone)= ...*gprime(neurone)*x_input(:)
>> end
>> toc
```

Exercice 4 – Algorithme de mise à jour des poids

Ecrire l'algorithme de mise à jour des poids d'un réseau de neurones qui réalise la mise à jour de l'ensemble de l'ensemble $(L_{in}+1) \times L_{cachée} + (L_{cachée}+1) \times L_{out} = 410\,103$ poids d'un réseau de neurones comportant $L_{in} = 1 + Nfft/2 = 4097$ entrées , $L_{out} = 3$ neurones sur la couche de sortie, et $L_{cachée} = 100$ neurones sur la couche cachée. Pour cela, procéder suivant les étapes suivantes :

Réaliser une boucle qui lit séquentiellement l'ensemble des attributs de la base d'apprentissage (stockés préalablement dans le tableau bidimensionnel `Attributs` de dimension 60×4097). Pour chaque exemple (= chaque fichier « son »), un total de $1 + Nfft/2$ attributs seront présentés en entrée du réseau. Chaque exemple sera numéroté par un paramètre t :

```
>> sonsUn = 1:20;sonsDeux = 1:20; sonsTrois = 1 :20 ;
>> t = zeros(3*20,1) ;
>> t(1 :3 :length(t)) = sonsUn ;
>> t(2 :3 :length(t)) = sonsDeux ;
>> t(3 :3 :length(t)) = sonsTrois ;
```

(1) Ecrire l'algorithme de *backpropagation* décrit par les équations (1) et (2) vues précédemment. On choisira des fonctions d'activation de type « produit scalaire » et des fonctions de transition de type « sigmoïde » pour $f(.)$ et $g(.)$. On montrera que la dérivée de ces fonctions vérifie la relation :

$$f(v_j) = \frac{1}{1-\exp(-v_j)} \Rightarrow \boxed{f'(v_j) = f(v_j) \times \left(1 - f(v_j)\right)}$$

(2) On utilisera deux tableaux contenant les poids du réseau de neurones : le premier tableau sera nommé `PoidsCaches` de dimension $(1 + L_{in}) \times L_{cachée}$ et le second tableau sera nommé `PoidsSortie` de dimension $(1 + L_{cachée}) \times L_{out}$. Les poids de ces 2 tableaux seront initialisés par tirage aléatoire dans une loi uniforme de $\mathcal{U}\left(-\frac{1}{2}, +\frac{1}{2}\right)$. Il ne faudra pas oublier de prendre en compte l'unité de biais intégrée à chaque neurone.

(3) Pour chaque exemple n°t de la BDD d'apprentissage présenté en entrée du réseau, vous devrez calculer les valeurs $\left(\hat{y}_m(t)\right)_{m=0,1,2}$ sur la couche de sortie du réseau, puis ensuite calculer l'erreur $e_m(t) = y_m(t) - \hat{y}_m(t)$. La supervision $\left(y_m(t)\right)_{m=0,1,2}$ sera alors apportée par un triplet $(1 \quad 0 \quad 0), (0 \quad 1 \quad 0)$ ou $(0 \quad 0 \quad 1)$ selon que le paramètre `typeson` vaut 1, 2, ou 3 (il suffit de calculer le reste de la division par 3 du numéro t du fichier à l'aide de l'instruction « `rem` » de ®matlab). Pour chaque exemple n°t, on calculera alors la somme des carrés des erreurs :

$$\mathcal{E}(t) = \frac{1}{L_{out}} \sum_{m=0}^{L_{out}} \frac{1}{2}\, e_m^2(t)$$

(4) Une fois que tous les exemples de la base d'apprentissage ont été présentés à l'entrée de la base d'apprentissage, l'erreur quadratique moyenne sera évaluée sur l'ensemble de la base d'apprentissage : $EQM = \frac{1}{20\times 3} \sum_{t=1}^{20\times 3} \mathcal{E}(t)$ et l'on affichera cette valeur dans la console.

(5) continuer l'apprentissage du réseau jusqu'à ce que le critère d'arrêt $EQM < \epsilon$ soit atteint (on choisira ϵ de l'ordre de 10^{-4}). On devra alors alors voir l'affichage de l'EQM diminuer à chaque fois que l'on aura présenté l'ensemble des exemples de la BDD d'apprentissage.

(6) L'objectif de cette exercice n°4 est simplement de **faire converger** l'algorithme de mise à jour des pondérations du réseau de neurones, c'est-à-dire d'observer que la valeur affichée de l'EQM diminue au fur et à mesure de la présentation des exemples de la base d'apprentissage.

Note : pour un réseau de neurones plus important comportant plusieurs couches cachées numérotées de $2,3,\dots,N$ (ici on a une unique couche cachée), l'algorithme de mise des poids de chaque couche cachée $n°l$ est donné par les relations suivantes :

En notant : $$\delta_j^{(couche\ l)} = g'(v_j)\underbrace{\left(\sum_{m=0}^{L^{(couche\ l+1)}-1} \delta_m^{(couche\ l+1)}\, w_{j,m}\right)}_{caclul\ n°1 = à\ calculer\ à\ part}$$

on obtient l'équation :

$$c_{k,j}(t+1) = c_{k,j}(t) + \frac{\mu}{L_{out}}\, x_k\ \delta_j^{(cachée)} \qquad \forall j = 0,1,\dots,L^{(couche\ l)}-1 \quad \forall k = 0,1,\dots,L^{(couche\ l-1)}-1$$

4. Quelques règles à suivre…si on veut que cela converge !

Ces règles sont déduites de l'excellent article scientifique disponible sur le Net:

> « Efficient BackProp » , Y . Le Cun, L. Bottou, G.B. Orr and K.R. Müller, originally published in *Neural Networks; tricks and the Trade*, Springer 1998.

L'un de ces auteurs, Yann Le Cun, est l'un des spécialistes mondiaux de l'apprentissage profond, il a notamment dirigé le laboratoire d'Intelligence Artificielle (IA) de Facebook aux Etats-Unis. Voir ses nombreuses conférences à ce sujet sur les vidéos de *You Tube*. Il est le titulaire pour l'année 2016 de la chaire "Informatique et Sciences Numériques" du Collège de France.

- ✓ **Choose examples with maximal Information content**
  - o Shuffle the training set so that successive training examples never (rarely) belong to the same class
  - o Present input examples with nearly equal frequencies per classes (ça c'est fait !)
  - o Entropie : Present input examples that produce a large error more frequently than examples that produce a small error. However, one must be careful when perturbating the normal frequencies of input samples

Exercice 5 – Mélanger l'ordre dans lequel on présente les exemples

A chaque fois que l'on représente au réseau de neurones l'ensemble des exemples de la base d'apprentissage, nous allons cette fois-ci les présenter dans un ordre différent (mais toujours un « un », suivi d'un « deux », suivi d'un « trois »). Utiliser l'instruction « `randperm` » sous ®Matlab pour effectuer des permutations au sein des vecteurs `sonsUn`, `sonsDeux` et `sonsTrois`.

- ✓ **Normalize the inputs**
  - o The average of each input variable over the training set should be close to zero
  - o Scale input variables so that their covariances are about the same
  - o Input variable should be uncorrelated if possible

Exercice 6 – Normaliser les données à l'entrée du réseau

Une fois que le tableau des `Attributs` de dimension 60×4097) est disponible (a) calculer la valeur moyenne et la variance sur chaque attribut (donc effectuer ce calcul sur des statistiques temporelles calculées sur 60 valeurs disponibles), puis (b) normaliser l'ensemble

du tableau `Attributs` de manière à obtenir une v.a. centrée réduite sur chaque dimension en entrée. Il faudra également normaliser de la même façon les données de la base de test.

- ✓ **Initializing the weights**
  - Assuming that (i) the training set has been normalized and (ii) the symmetric sigmoid has been used then…
  - …. weights should be randomly drawn from a uniform distribution with mean zero and standard deviation $\sigma = 1/\sqrt{m}$ where m is the fan-in (the number of connections feeding *into* the node)

Exercice 7 – Initialisation les poids

Prendre en compte l'information ci-dessus afin d'optimiser la valeur d'initialisation des poids des neurones suivant qu'ils se situent sur la couche cachée ou la couche de sortie.

- ✓ **Choosing learning rate**
  - Many (empirical) schemes have been proposed in the literature to automatically adjust the learning rate
  - Of the general form $\mu(t) = \frac{\mu_0}{1+\alpha\, t}$ or computed from the Hessian of the error surface $\mu(t) = \frac{\mu_0}{\epsilon + \langle \frac{\partial^2 \quad \varepsilon}{\partial \; c_{k_0,j_0}^2} \rangle}$

- ✓ **Equalize the learning speeds**
  - Give each weight its own learning rate
  - Learning rates should be proportional to the square root of the number of the inputs to the unit
  - Weights in lower layers should typically be larger than in the higher layers

Exercice 8 – Valeur du pas d'adaptation

- ✓ Prendre en compte l'information ci-dessus afin d'optimiser la valeur du pas d'adaptation des poids des neurones : (1) en établissant un pas d'adaptation $\mu(t) = \frac{\mu_0}{1+\alpha\, t}$ variable au cours du temps (on choisira pour la couche cachée $\mu_o = 0.5$ et on fixera le paramètre α de manière à atteindre une valeur contrôlée du pas d'adaptation lors de la présentation des derniers fichiers de la base de données d'apprentissage) et (2) en établissant un pas d'adaptation différent entre les neurones qui appartiennent à la couche de sortie ou à la couche cachée

- ✓ **Momentum can increase speed of convergence**
  - $\boldsymbol{w}_j(t+1) = \alpha\, \boldsymbol{w}_j(t) + \Delta \boldsymbol{w}_j$ avec $0 < \alpha \leq 1$. Generally α is set to 0.5 until the initial learning stabilizes and then is increased to 0.9 or higher
  - Give each weight its own learning rate

5. Eviter le sur-apprentissage : les méthodes !

- Sur l'apprentissage…il faut que vous regardiez ces 2 points :
  - La ***Validation croisée*** est une technique qui permet de tester la précision prédictive du modèle (i.e. du réseau de neurones). On trouve plusieurs méthodes dont les plus

pertinentes sont « testset validation », « holdout method », et « leave-one-out cross-validation » (LOOCV) .

- Le ***Surapprentissage*** : méthode de régularisation, méthode d'élagage

- Les réseaux actuels…Deep-Neural Networks (DNN)

- Deep-learning = Convolutional Neural Network (CNN ou ConvNet) + Restricted Boltzmann machine (RBM)
- Voir magasine « La Recherche » n°498

« Le Deep Learning pas à pas : les concepts (1/2) » dans http://www.technologies-ebusiness.com/enjeux-et-tendances/le-deep-learning-pas-a-pas

« Le Deep Learning pas à pas : l'implémentation (2/2) » dans http://www.technologies-ebusiness.com/langages/le-deep-learning-pas-a-pas-limplementation-22

Et bien sûr le site de Geoffrey E. Hinton (http://www.cs.toronto.edu/~hinton/) qui a popularisé ces techniques à partir de 2006 avec ses travaux de recherche menés avec Y. Le Cun et A. Krizhevsky.