Les réseaux de neurones

Houcine Senoussi

juillet 2024

Introduction

- Dans ce chapitre nous introduisons les réseaux de neurones artificiels, leur architecture et leur fonctionnement.
- Il s'agit d'une méthode d'apprentissage supervisé, utilisée aussi bien en classification qu'en régression.

Qu'est-ce qu'un réseau de neurones?

- Vu comme une boite noire (figure7), un réseau de neurones est un système ayant des entrées **numériques** et des sorties **numériques**.
- Autrement dit, du point de vue mathématique, un réseau de neurones est une fonction de $\mathbb{R}^n$ dans $\mathbb{R}^m$.
- Nous notons $(x_1, \ldots, x_n)$ les entrées du réseau et $(\hat{y}_1, \ldots, \hat{y}_m)$ ses sorties.

Qu'est-ce qu'un réseau de neurones?

Figure: Un réseau de neurones vu comme boite noire.

Qu'est-ce qu'un réseau de neurones?

- Problèmes de régression:
  1. Étant donné une instance X décrite par un n-uplet $(x_1, \ldots, x_n)$, le réseau doit calculer une valeur $f(X) \in \mathbb{R}^m$.
  2. Dans ce cas, le RN a n entrées et m sorties dont la signification est immédiate.
- Problèmes de classification:
  1. Étant donné une instance X décrite par un n-uplet $(x_1, \ldots, x_n)$, le RN doit permettre de déterminer sa classe (éventuellement ses classes dans le cas multi-labels) dans un ensemble de classes $\{C_1, ..., C_m\}$.
  2. Dans ce cas, le RN a n entrées et le nombre de sorties dépend de m de la manière qui sera expliquée plus bas.

Qu'y a-t-il à l'intérieur de la boite noire ?

- Un réseau de neurones se compose d'une **suite** de **couches** (figure3): la couche d'**entrée**, les couches **cachées** et la couche de **sortie**.
- La couche d'entrée sert juste à introduire les données dans le réseau.
- Une couche cachée reçoit ses entrées de la couche précédente, fait des calculs, et transmet les résultats à la couche suivante.
- La couche de sortie reçoit ses entrées de la couche précédente, fait des calculs, et transmet les résultats (les sorties du réseau, donc) à l'extérieur.
- Du point de vue mathématique, chaque couche cachée et la couche de sortie sont donc des fonctions de $\mathbb{R}^{n1}$ dans $\mathbb{R}^{n2}$, le réseau de neurones est la composée de ces fonctions.

Qu'y a-t-il à l'intérieur de la boite noire ?

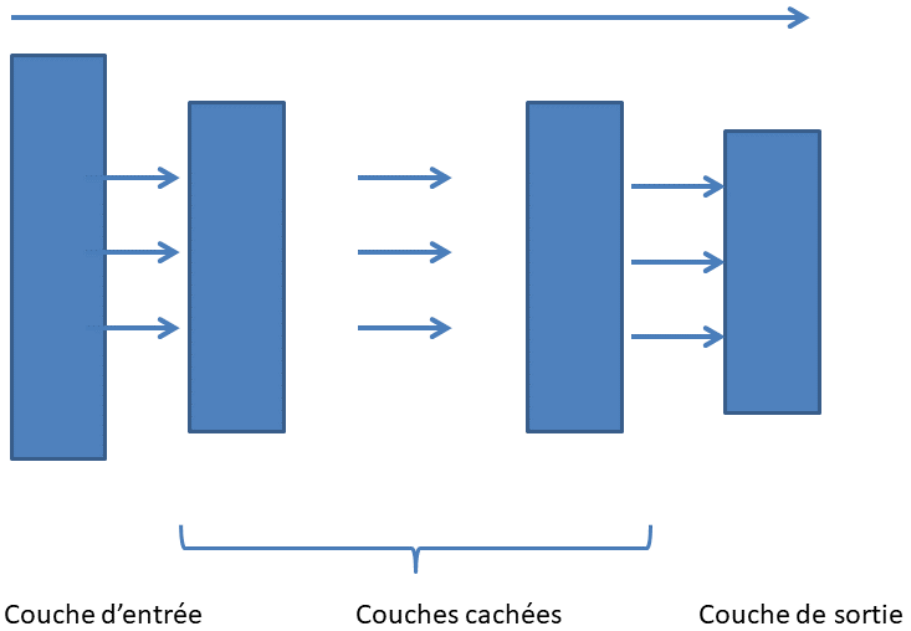


Figure: Couches d'un RN.

Qu'y a-t-il à l'intérieur de la boite noire ?

- Nous noterons L le nombre de couches du réseau, et $l = 1, \ldots, L$ l'indice de chaque couche.
- Compte tenu du sens unique de la circulation des données, le réseau est qualifié de **feedforward**.

Zoom sur une couche

- Une couche est un tableau de neurones.
- Un neurone est une unité de calcul.

Le neurone

- Le fonctionnement du neurone est décrit par la figure suivante:

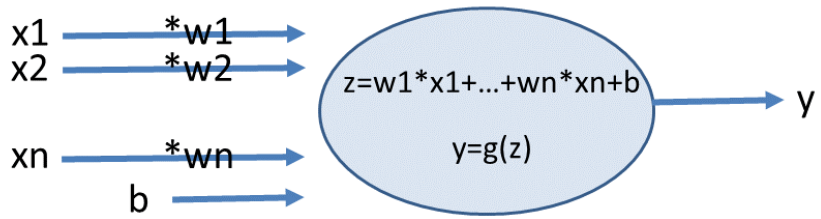


Figure: Un neurone.

Le neurone

1. Le neurone (figure ci-dessus) reçoit en entrée $n1$ valeurs x_1, ..., x_{n1} issues d'autres neurones.
2. Chaque neurone a $n1$ **poids** w_i et un **biais** b qui lui sont **propres**.
3. Le neurone calcule la valeur $z = \sum_i w_i * x_i + b$.
4. Le neurone calcule sa sortie $y = g(z)$ où g est la fonction d'**activation**. Tous les neurones d'une même couche ont la même fonction d'activation.

Zoom sur une couche

- **Chaque** neurone de la couche $l+1$ est connecté en **entrée** à **chaque** neurone de la couche l. Autrement dit, il reçoit en entrée la sortie de chacun de ces neurones.
- **Chaque** neurone de la couche l est connecté en **sortie** à **chaque** neurone de la couche $l+1$. Autrement dit, il envoie sa sortie à chacun de ces neurones.
- Le nombre de poids d'un neurone (noté $n1$ ci-dessus) de la couche l est donc égal au nombre de neurones de la couche $l-1$.
- Cette double propriété des neurones nous conduit à qualifier le RN de totalement connecté (**fully connected**).

Notations

1. Nombre de couches : L.
   - Couche $l = 1$: Couche d'entrée.
   - Couches $l \in \{2, \ldots, L-1\}$: Couches cachées.
   - Couche $l = L$: Couche de sortie.
2. Nombre de neurones de la couche l: nl.
3. La position de chaque neurone dans le réseau est définie par un couple (l, j), où l est le numéro de la couche à laquelle appartient le neurone, et $j \in \{1, \ldots nl\}$ est la position du neurone dans sa couche.
4. Nous notons w^l_{jk} le poids de la connexion entre le neurone $(l-1, k)$ et le neurone (l, j), et b^l_j le biais du neurone (l, j).
5. Nous notons $x^{(l)}_j$ la sortie du neurone (l, j).
   - Pour la couche de sortie, $x^{(L)}_j$ se confond donc avec $\hat{y}_j$.

Notations-2

1. Compte tenu des notations présentées ci-dessus, nous avons
$$x_j^{(l)} = g(\sum_{k=1}^{n_{l-1}} w_{jk}^l x_k^{(l-1)}).$$
2. La somme $\sum_{k=1}^{n_{l-1}} w_{jk} x_k^{(l-1)}$, appelée somme pondérée (**weighted input**) du neurone (l, j), sera notée $z_j^{(l)}$.
3. Nous avons donc $x_j^{(l)} = g_l(z_j^{(l)})$.

Un RN pour le OU logique

- Notre premier réseau de neurones reçoit en entrée deux valeurs binaires x_1 et x_2 et fournit en sortie la valeur $y = x_1 \vee x_2$ où $\vee$ est l'opérateur *OU* logique.
- Ce réseau aura donc deux neurones en entrée et un neurone en sortie. Compte tenu de la simplicité du problème, nous allons construire un réseau sans couche cachée.
- Ce réseau sera donc entièrement déterminé par le choix d'une fonction d'activation h et la détermination des poids $w1$ et $w2$ et le biais b du neurone de sortie.

Un RN pour le OU logique

- La sortie du réseau sera $h(w1 * x_1 + w2 * x_2 + b)$. Puisque cette valeur sera toujours égale à 0 ou 1, nous allons prendre comme fonction d'activation h la fonction de **Heaviside** définie comme suit (figure 4(a) ci-dessous):

$$\begin{cases} h(x) = 1, si\ x\ >\ 0 \\ h(x) = 0, sinon \end{cases}$$

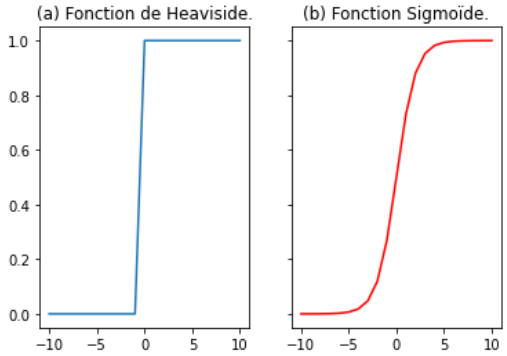


Figure: Fonctions d'activation

Un RN pour le OU logique

- Reste à déterminer les poids et le biais. C'est ce que nous allons faire à l'aide des graphiques suivants:

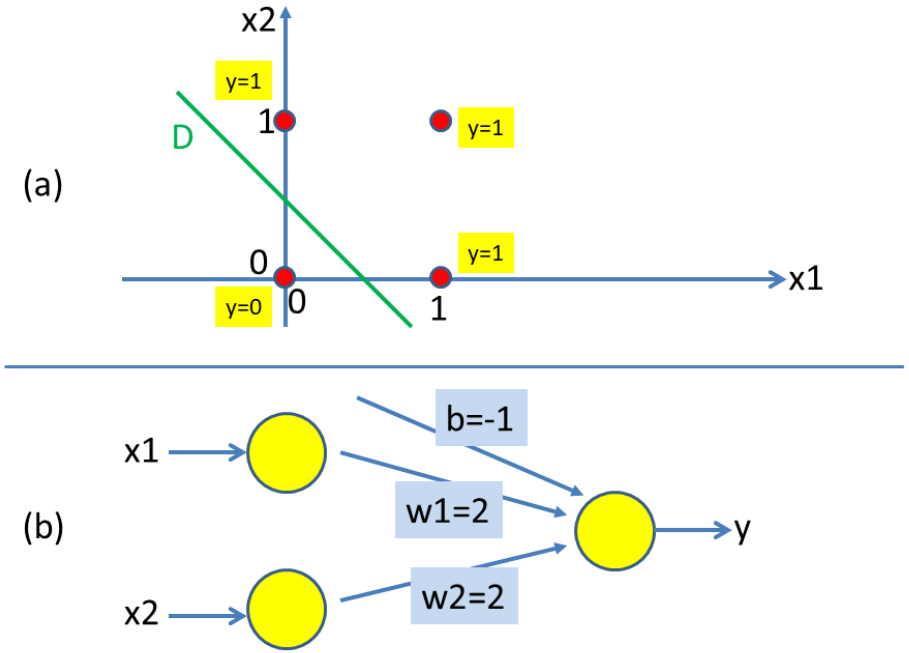


Figure: Un RN pour le OU logique.

Un RN pour le OU logique

- Nous plaçons les 4 entrées possibles dans le plan (x_1, x_2).
- Nous constatons qu'il existe une (infinité de) droite(s) qui sépare(nt) les points correspondant à $y = 0$ de ceux correspondant à $y = 1$.
- Autrement dit, il existe $w1$, $w2$ et b tels que:
$$\begin{cases} w1x1 + w2x2 + b > 0 \ lorsque \ y = 1 \\ w1x1 + w2x2 + b < 0 \ lorsque \ y = 0 \end{cases}$$
- ou encore:
$$\begin{cases} h(w1x1 + w2x2 + b) = 1 \ lorsque \ y = 1 \\ h(w1x1 + w2x2 + b) = 0 \ lorsque \ y = 0 \end{cases}$$
- Cela donne le RN de la figure 5 (b).

Perceptrons et séparabilité linéaire

Définition :

Soit f une fonction de $\mathbb{R}^n$ dans $\{0,1\}$. f est dite **linéairement séparable** s'il existe un hyperplan qui sépare les points de $\mathbb{R}^n$ dont l'image par f est égale à 1 et ceux dont elle est égale à 0.
Autrement dit, s'il existe $(a_1, \dots a_n, b) \in \mathbb{R}^{n+1}$ tel que $\forall$ $(x_1, \dots x_n) \in \mathbb{R}^n$ $f(x) = 1 \iff a_1 * x_1 + \cdots + a_n * x_n + b > 0$.

Définition :

Un **perceptron** est un réseau de neurones, sans couches cachées, dont la couche de sortie a un seul neurone et dont la fonction d'activation est la fonction de Heaviside.

Perceptrons et séparabilité linéaire(2)

Propriété :

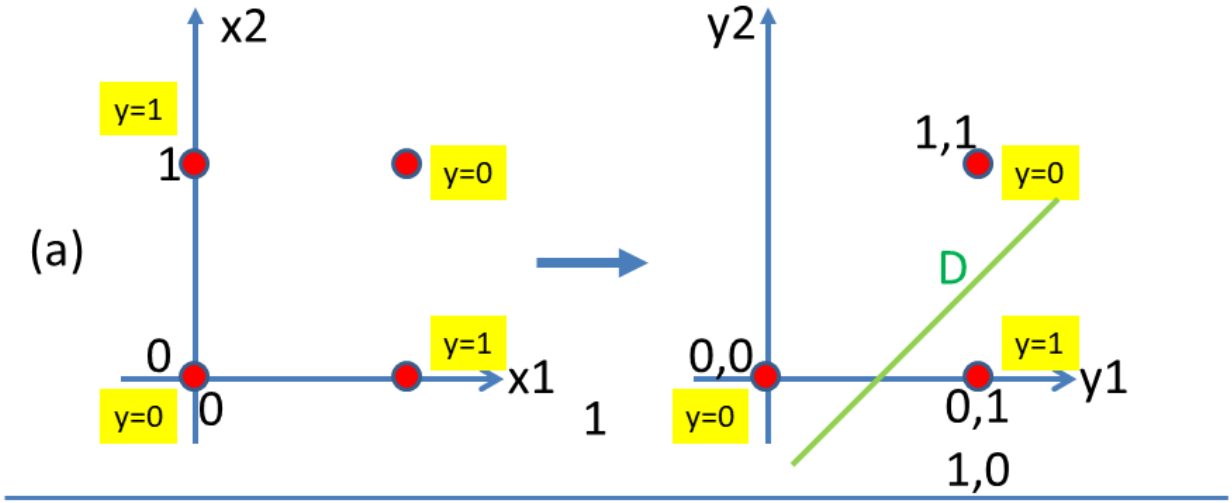
Une fonction f de $\mathbb{R}^n$ dans $\{0, 1\}$ peut être représentée par un perceptron si et seulement si elle est linéairement séparable.

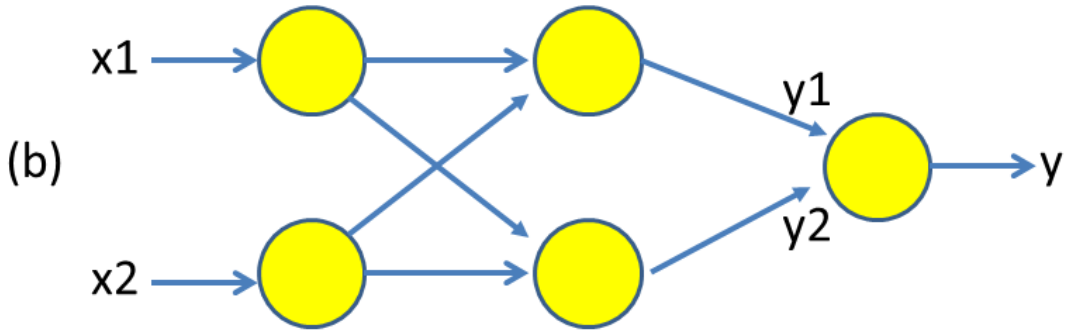
- **Exercice :** Vérifiez que la fonction *ET* logique est aussi linéairement séparable et définissez un RN permettant de la calculer.
- Intéressons-nous à présent à la fonction **OU-exclusif** (XOR).

Un RN pour le OU-exclusif (XOR)

- La figure 6(a)-gauche montre clairement que cette fonction n'est pas **linéairement séparable**. Nous allons donc utiliser un RN ayant une **couche cachée** et fixons à 2 le nombre de neurones de cette couche (figure 6(b)).
- Les deux neurones de la couche cachée transforment l'entrée $(x1, x2)$ en $(y1, y2)$. Autrement dit la couche cachée permet d'avoir une nouvelle **représentation** de $(x1, x2)$.
- Pour obtenir cette nouvelle présentation, nous avons pris $y1 = x1 \vee x2$ et $y1 = x1 \wedge x2$.
- La figure 6(a)-droite montre la nouvelle représentation. On vérifie bien que dans le nouveau plan $(y1, y2)$, il existe une (infinité de) droite(s) permettant de séparer les points pour lesquels y vaut respectivement 0 et 1. Nous utilisons une telle droite pour définir le neurone de la couche de sortie.

Un RN pour le OU-exclusif (XOR)





Paramètres et hyperparamètres d'un RN

- Pour définir et faire fonctionner un réseau de neurones, nous avons besoin (au moins) des paramètres suivants:
  - le nombre de couches L (c'est-à-dire, en réalité, le nombre de couches cachées $L-2$),
  - le nombre de neurones dans chaque couche,
  - la fonction d'activation de chaque couche,
  - les poids et les biais des connexions entre les neurones, c'est-à-dire la définition de chaque neurone caché ou de sortie.
- En nous posant la question de savoir comment on détermine ces différents paramètres, nous nous rendons compte qu'il y a une différence essentielle entre les **paramètres** (les poids et les biais) d'un coté, et les **hyperparamètres** (les autres paramètres) de l'autre.

Paramètres et hyperparamètres d'un RN

- Les hyperparamètres sont fixés par le concepteur du réseau qui peut (du moins pour certains) en essayer plusieurs valeurs et les ajuster en fonction des résultats auxquels ils conduisent.
- En revanche, le concepteur du réseau n'a pas la main sur les poids et les biais parce que leurs valeurs doivent être celles, ou faire partie de celles, qui permettent de calculer la fonction que le réseau est censé représenter. Un algorithme utilise un **ensemble d'apprentissage**, c'est-à-dire un ensemble d'**exemples** $D = \{X_i, Y_i)\}$ pour déterminer les poids et les biais de manière $\hat{Y}_i$ (sorties du réseau, valeurs **prédites** de Y) soient aussi proches que possible des Y_i (valeurs **cibles** de Y). Autrement dit, le réseau doit **apprendre** ses paramètres.

Détermination des hyperparamètres

- Il n'existe pas de véritables règles pour déterminer les hyperparamètres. En règle général, ils sont déterminés en essayant plusieurs valeurs et en prenant celles qui donnent les meilleurs performances (tout en gardant un coût raisonnable pour le réseau et en évitant le sur-apprentissage).
- Nous pouvons exclure de la liste des hyperparamètres le nombre de neurones des couches d'entrée et de sortie. Ces 'paramètres' sont en réalité déterminés par la fonction que le réseau doit représenter. Rappelons en effet que:
  1. le nombre de neurones de la couche d'**entrée** n'est rien d'autre que le nombre de variables (**indépendantes**) de cette fonction,
  2. et que le nombre de neurones de la couche de **sortie** se déduit naturellemnt du nombre de variables **dépendantes**.

Couches de sortie

- Problèmes de **régression** (à une ou plusieurs sorties) : le nombre m de neurones de la couche de sortie est tout simplement égal à celui des variables à prédire.
- Classification **binaire**: la couche de sortie aura **un seul** neurone dont la sortie donnera la **probabilité** d'une classe choisie comme classe de **référence**.
- Classification à trois classes ou plus (**multi-classes** ou **multi-étiquettes**): le nombre m de neurones de la couche de sortie est égal à celui des classes. Chaque sortie donnera la **probabilité** d'une classe.

Fonctions d'activation

- Historiquement, la première fonction d'activation à avoir été utilisée est la fonction de **Heaviside**. Sa sortie binaire correspond à la définition d'un neurone à deux états: activé (suite au calcul d'une combinaison de ses entrées) ou non activé.
- Cela limite son intérêt car l'empêche d'exprimer une probabilité ou de permettre des calculs intermédiaires dont le résultat n'est pas binaire.
- Mathématiquement, elle a aussi l'inconvénient de ne pas être continue en 0 (donc non dérivable) alors que ces bonnes propriétés sont utiles dans des algorithmes essentiels pour les RN.

La fonction sigmoïde

- La fonction **sigmoïde** (figure 4-(b)) peut être vue comme une version 'lissée' de la fonction de Heaviside. À ce titre, elle permet de distinguer les valeurs hautes (voisines de 1) et les valeurs basses (voisines de 0) des sorties des neurones.
- Elle est définie par

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

- Elle possède les 'bonnes' propriétés qu'une fonction d'activation doit avoir (e.g. continuité et dérivabilité).
- Ses valeurs étant comprises entre 0 et 1 et pouvant donc être interprétées comme des probabilités, on l'utilise dans la couche de sortie dans le cas d'une classification binaire (mais aussi d'une classification multi-labels).
- On l'utilise aussi dans les couche cachées.

Autres fonctions d'activation

- Il existe un grand nombre de fonctions d'activation. Citons notamment les deux suivantes:
  - La fonction d'activation **linéaire** définie par $f(x) = \lambda * x$ et dont la fonction identité est un cas particulier. Cette fonction a l'avantage de la simplicité, mais son usage est limité à des situations particulières, notamment la couche de sortie dans le cas d'un problème de régression.
  - La fonction **tangente hyperbolique** définie par $th(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, a les mêmes propriétés que la fonction sigmoïde, mais en diffère par le fait qu'elle a des valeurs négatives et qu'elle est centrée autour de 0. Elle est utilisée dans les couches cachées.

Cas particulier de la fonction Softmax

- Dans le cas d'un problème de multi-classification à m classes on utilise la fonction **softmax** comme fonction d'activation de la couche de sortie.
- Pour alléger les notations, notons $x_1, ..., x_n$ les entrées de la couche de sortie (c'est-à-dire les sorties de la couche $L-1$. Quelle est la notation rigoureuse pour ces valeurs?).
- Toujours pour alléger (légèrement) les notations, notons w_{jk} et b_j les poids et le biais du j^{eme} neurone de la couche de sortie.
- Chaque neurone j commence par calculer la moyenne pondérée:
  - $z_j = \sum_{k=1}^{n} w_{jk} * x_k + b_j$
- Ensuite il calcule sa sortie comme suit :
  - $\hat{y}_j = \frac{exp(z_j)}{\sum_{j=1}^{m} exp(z_i)}$

Cas particulier de la fonction Softmax

- Les deux propriétés principales de cette fonction, qui la rendent utile ici sont:
  - La sortie de chaque neurone est comprise entre 0 et 1.
  - La somme des sorties est égale à 1.
- Autrement dit, ce qui rend utile la fonction Softmax est que ses valeurs peuvent être interprétées comme m probabilités, ce qui correspond à notre besoin dans le cas d'un problème de multi-classification.
- La particularité de cette fonction est qu'elle suppose que chaque neurone a accès aux valeurs z_i des autres neurones. On parle de couche Softmax plutôt que de fonction d'activation seulement.

Exemple:Un RN pour le dataset Iris

- Problème de multi-classification.
- Variables indépendantes? modalités de la classe?
- Architecture du RN.
- Fonctions d'activation.

Calcul des paramètres d'un RN

- Une fois les hyperparamètres déterminés, il nous reste à calculer les paramètres, c'est-à-dire les poids et les biais de chaque neurone.
- Pour ce faire, nous disposons d'un **ensemble d'apprentissage** $D = \{(X_k, Y_k)\ k = 1, \ldots, K\}$.
- Il s'agit donc de trouver des poids et des biais qui permettent au réseau, lorsqu'il reçoit l'entrée X_k, de produire une sortie aussi proche que possible de Y_k.
- Le principe des algorithmes utilisés pour cela est celui de la **correction des erreurs**: calculer une **erreur** par **comparaison** des sorties voulues et des sorties produites par le réseau, puis **ajuster** les paramètres dans un sens qui permet de **réduire** cette erreur.

Apprentissage par correction des erreurs

Le schéma général des algorithmes d'apprentissage par correction des erreurs est le suivant:

Algorithme: Calcul des poids par correction des erreurs.

---

1. Initialiser les poids et les biais de manière aléatoire.
2. Tant que (critère d'arrêt n'est pas atteint)
   1. Pour chaque exemple $(X, Y) \in D$, calculer la sortie $\hat{Y}$ correspondant à l'entrée X.
   2. Calculer l'erreur en comparant les valeurs Y et $\hat{Y}$.
   3. Ajuster les poids et les biais de manière à réduire l'erreur.

Apprentissage par correction des erreurs

Pour passer de ce schéma général aux algorithmes, plusieurs choix doivent être faits:

1. Comment le réseau doit-il calculer l'erreur? Cela nous amènera à introduire les fonctions de **perte** (**loss** functions).
2. Comment ajuster les poids et les biais de manière à **minimiser** l'erreur? Là, il s'agit donc de choisir un algorithme d'**optimisation**.
3. Quel critère d'arrêt appliquer? Un tel critère est inclus dans l'algorithme d'optimisation. Il est lié plus particulièrement à la **convergence** de ce dernier.

Dans la suite, nous allons présenter deux algorithmes d'apprentissage: d'abord l'algorithme simple appelé 'règle du Delta' (Delta rule), ensuite l'algorithme le plus général (la rétropropagation).

Règle du Delta

Considérons un réseau de neurones sans couches cachées, dont la couche de sortie comporte un nombre quelconque m de neurones et dont la fonction d'activation est **linéaire**. L'ensemble d'apprentissage est donc comme suit:

- $D = \{(X_k, Y_k)\ i = 1, \ldots, K\}$.
- Pour $k = 1, \ldots, K$ $X_k = (x_{k1}, \ldots, x_{kn})$ et $Y_k = (y_{k1}, \ldots, y_{km})$.

Pour chaque exemple (X_k, Y_k), la sortie $\hat{Y}_k$ calculée par le réseau à chaque itération est un m-uplet $(\hat{y}_{k1}, \ldots, \hat{y}_{km})$ tel que:

- Pour $j = 1, \ldots, m$ $\hat{y}_{kj} = \sum_{i=1}^{n} w_{ji} * x_{ki} + b_j$

Règle du Delta-Définition de l'erreur

Pour chaque exemple, l'écart entre la sortie voulue et la sortie calculée par le réseau à chaque itération est calculé de la manière suivante:

$$\begin{aligned} E_k &= \frac{1}{2}\sum_{j=1}^{m}(y_{kj} - \hat{y}_{kj})^2 \\ &= \frac{1}{2}\| Y_k - \hat{Y}_k \|^2 \end{aligned}$$

L'erreur globale est donc:

$$\begin{aligned} E &= \sum_{k=1}^{K} E_k \\ &= \frac{1}{2}\sum_{k=1}^{K} \| Y_k - \hat{Y}_k \|^2 \\ &= \frac{1}{2}\sum_{k=1}^{K}\sum_{j=1}^{m}(y_{kj} - \hat{y}_{kj})^2 \\ &= \frac{1}{2}\sum_{k=1}^{K}\sum_{j=1}^{m}(y_{kj} - w_{j1} * x_{k1} - \cdots - w_{jn} * x_{kn} - b_j)^2 \end{aligned}$$

Règle du Delta-Définition de l'erreur

L'erreur E est donc une fonction **quadratique** des poids $w_{11}, \ldots, w_{mn}$ et des biais $b_1, \ldots, b_m$. Pour la minimiser, nous allons utiliser une **descente du gradient**. Soit, en notant W le vecteur contenant les poids et les biais, et t le numéro d'une itération:

$$W_{t+1} = W_t - \eta \nabla E(W_t) \quad (1)$$

où η est une constante réelle appelée **taux d'apprentissage** (learning rate) et $\nabla E()$ est le gradient de E.

Règle du Delta-Définition de l'erreur

L'équation (1) peut être réecrite de la manière suivante:

$$\begin{cases} \text{Pour } i = 1, \ldots, n,\ j = 1, \ldots, m & \Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}(W) \quad (2a) \\ \text{Pour } j = 1, \ldots, m & \Delta b_j = -\eta \frac{\partial E}{\partial b_j}(W) \quad (2b) \end{cases}$$

Ce qui donne, compte tenu de la définition de E:

$$\begin{cases} \Delta w_{ji} = \eta \sum_{k=1}^{K} x_{ki} \left(y_{kj} - \hat{y}_{kj}\right) & (3a) \\ \Delta b_j = \eta \sum_{k=1}^{K} \left(y_{kj} - \hat{y}_{kj}\right) & (3b) \end{cases}$$

Règle du Delta-Définition de l'erreur

- Nous constatons la présence d'un terme commun dans les équations (3a) et (3b). C'est précisément ce terme propre à chaque neurone qu'on appelle δ^j. Nous avons:

$$\delta_j \quad = \quad (y_j - \hat{y}_j).$$

- Nous retrouverons ce terme plus bas, dans le cas le plus général.

Règle du Delta

- Reste à définir un critère d'arrêt. Pour cela nous avons plusieurs possibilités.Cela peut être par exemple $E < E_{max}$, deux valeurs successives de W et B sont quasi-identiques (utilisation d'un paramètre *epsilon*), idem pour deux valeurs de l'erreur (ou ce qui revient au même de $\hat{Y}_k$). Ces critères peuvent être combinés avec un nombre d'itérations maximal.
- La **convergence** de l'algorithme est assurée par le fait que l'erreur (c'est-à-dire la fonction à minimiser) est quadratique. Dans ce cas, cette fonction possède un **minimum unique**.

Rétropropagation

- Nous allons à présent retourner au cas général d'un réseau de neurones **quelconque** et nous inspirer de la règle du Delta pour avoir un nouvel algorithme d'apprentissage.
- Il s'agit donc de **généraliser** la règle du Delta de 3 manières: nombre de couches quelconque, fonction d'activation quelconque, fonction de perte quelconque.

Fonctions de perte

- Étant donnée l'ensemble d'apprentissage $D = \{(X_k, Y_k), k = 1, \ldots, n\}$, la fonction de perte (**loss** function) mesure l'écart entre les valeurs de Y_k et celles des sorties $\hat{Y}_k$ que le réseau de neurones calcule lorsqu'on lui soumet X_k en entrée.
- L'objectif de l'apprentissage est donc de minimiser la valeur de cette fonction.

Fonctions de perte

- Les principales fonctions de perte utilisées dans les RN sont:
  1. L'**erreur quadratique**, c'est-à-dire la méthode des **moindres carrés**.
  2. L'entropie croisée binaire (**binary cross-entropy**), utilisée notamment dans les problèmes de classification binaire. Elle est définie comme suit:
  $$BCE(\hat{y}, y) = -(y log(\hat{y}) + (1 - y) log(1 - \hat{y})).$$
  3. L'entropie croisée catégorielle (**categorical cross-entropy**) utilisée dans les problèmes de multi-classification. En utilisant pour Y la représentation one-hot, notre fonction de perte peut s'écrire de la manière suivante:
  $$CCE(\hat{y}, y) = -\sum_{j=1}^{m} y_j log(\hat{y}_j).$$

Rétropropagation

- Le principe de l'algorithme reste identique: utiliser une descente du gradient pour mettre à jour à chaque itération les poids et les biais.
- L'application de la descente du gradient se fait d'une manière qui donne son nom à l'algorithme : les mises à jour des poids et des biais se font **couche par couche, de droite à gauche**.
- Derrière cette rétropropagation il y a le fait que chaque neurone (l, r), pour mettre à jour ses poids et ses biais, a besoin de calculer une valeur qu'on note δ_r^l qui dépend des valeurs δ_s^{l+1} calculées par les neurones de la couche suivante.

Rétropropagation

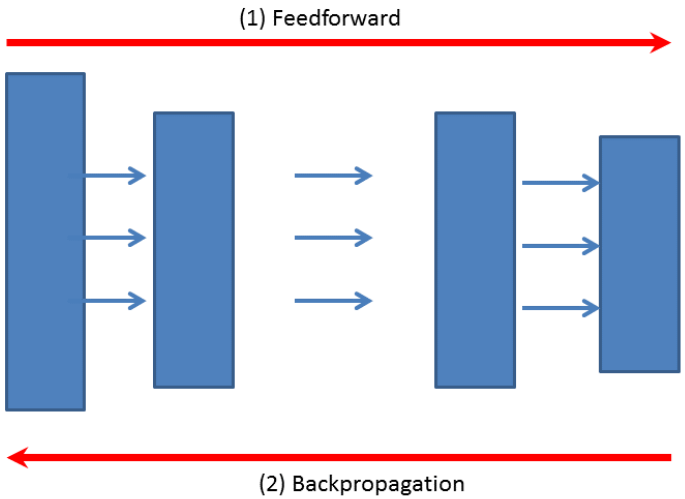


Figure: Étapes de l'apprentissage.

Rétropropagation

Algorithme: Rétropropagation.

- Entrées : l'ensemble d'apprentissage
  $D = \{(X_k, Y_k), k = 1, ..., n\}$

- Initialiser W and B.
- Tant que (critère d'arrêt n'est pas atteint)
  1. Pour k=1, ..., n
     1. Mettre X_k à l'entrée du RN.
     2. Pour $l = 2, ..., L$ calculer les sorties de la couche l. **(Feedforward)**
     3. La sortie de la couche L est $\hat{Y}_k$.
     4. Utiliser les valeurs Y_k et $\hat{Y}_k$ pour calculer l'erreur E.
     5. Pour $l = L, L-1, ..., 1$ **(Rétropropagation)**
        Calculer les valeurs δ_j^l qui dépend des valeurs δ_i^{l+1}.
        Calculer les valeurs $\frac{\partial E}{\partial w_{jk}^l}$ and $\frac{\partial E}{\partial b_j^l}$
        Mettre à jour les valeurs de w_{jk}^l and b_j^l.

La rétropropagation dans la pratique

Tel que nous l'avons présenté ci-dessus, cet algorithme met à jour les poids et les biais à chaque exemple. Dans la pratique, ce n'est pas toujours le cas. De ce point de vue, il y a en effet 3 versions de cet algorithme:

- La version **batch** learning: à chaque itération, l'algorithme calcule une erreur globale sur tous les exemples avant de mettre à jour les poids et les biais.
- La version descente du gradient **stochastique**: à chaque itération, un exemple est choisi **aléatoirement**, et les calculs puis les mises à jour sont effectués.
- La version **mini-batch** learning: La mise à jour se fait tous les *mb* exemples, *mb* étant la taille du mini-batch.

Conclusion

Dans ce premier chapitre consacré aux réseaux de neurones, nous avons présenté les réseaux de neurones 'standards', leur architecture et leur fonctionnement.