Université Mohamed BOUDIAF de M'SILA
Faculté des Mathématiques et de l'Informatique
Département de l'Informatique

Matière : Intelligence artificielle- 3SI

Cours Les **réseaux de neurones**

Principe :

Les chercheurs se sont assez rapidement intéressés **au fonctionnement du cerveau pour le reproduire**. C'est ainsi que **les premiers neurones artificiels** ont été définis par Mac Culloch et Pitts en 1943 .

Aujourd'hui, on cherche à avoir des systèmes pouvant résoudre certains problèmes complexes sur lesquels les systèmes classiques sont limités. C'est ainsi que sont nés les **réseaux de neurones artificiels**.

On sait depuis longtemps que la réflexion se fait grâce au cerveau.

Les cellules les plus importantes du cortex cérébral sont les neurones. Ceux-ci sont très nombreux, vu que l'on en compte presque une centaine de milliards chez l'être humain.

On sait que les neurones communiquent entre eux via des impulsions électriques.

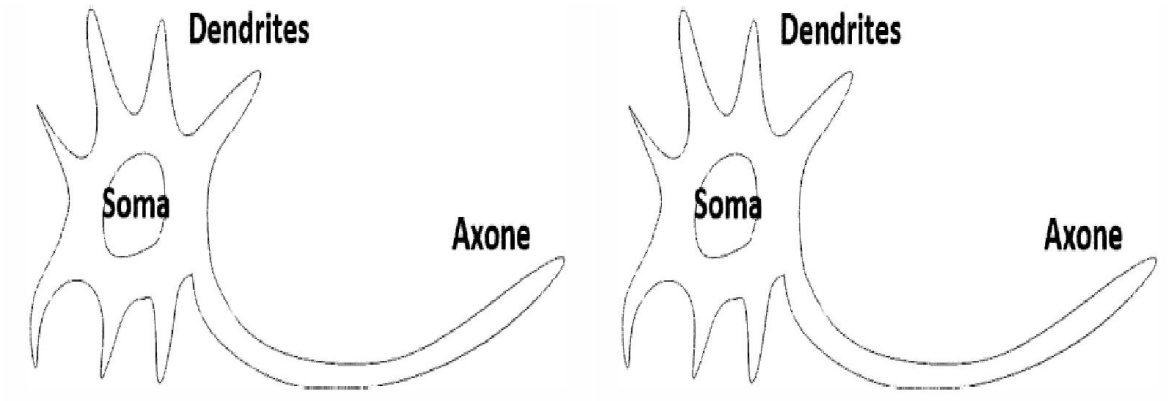
En effet, les "capteurs" (œil, oreille, peau …) envoient des impulsions électriques aux neurones, qu'ils traitent et transmettent ou non aux autres cellules.

Chaque neurone possède donc autour de son cœur (nommé soma) :

- **Des dendrites**, qui sont ses entrées.
- **Un long axone** lui servant de sortie.

Les signaux électriques arrivent donc au soma en suivant les dendrites, puis sont traités : selon l'intensité et la somme des impulsions reçues, le neurone envoie ou non une impulsion le long de son axone. Celui-ci est relié à des dendrites d'autres neurones .

Un neurone peut donc être schématisé comme suit:



Chaque neurone est donc une entité très simple, faisant simplement un travail sur les impulsions reçues pour choisir ou non d'en envoyer une en sortie.

La **puissance du cerveau** se situe en fait dans le nombre de neurones et les **nombreuses interconnexions** entre eux.

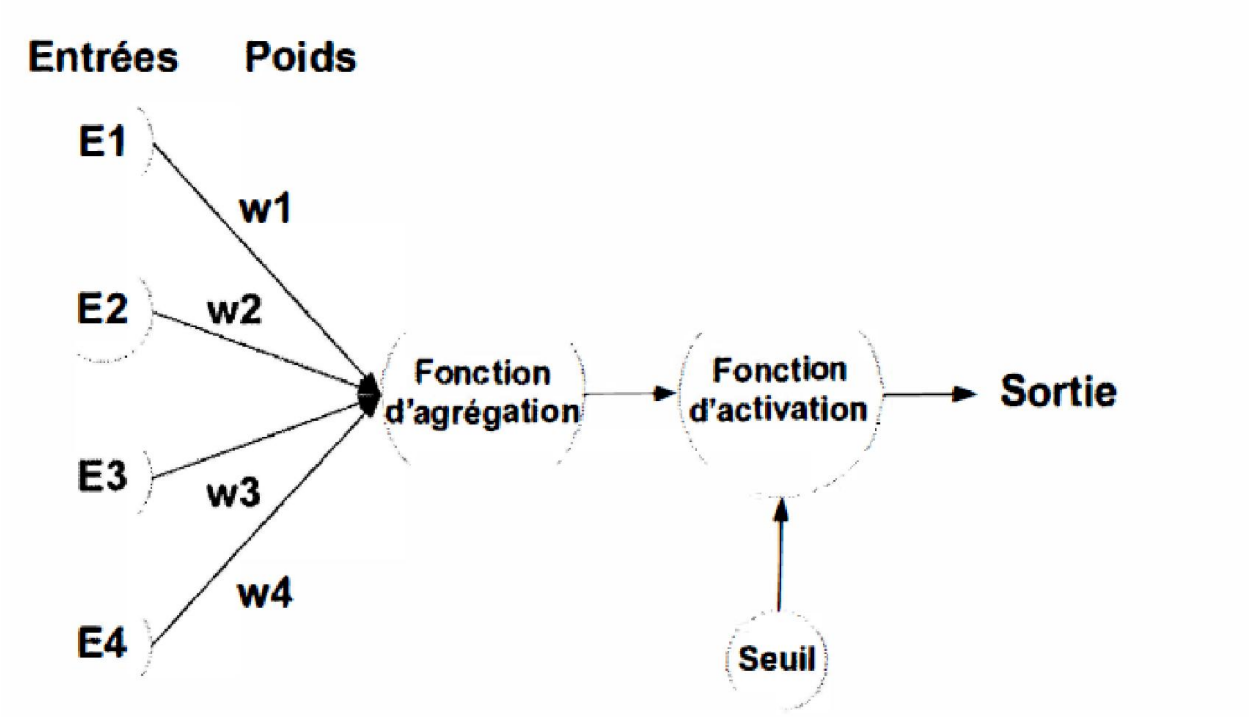
2. Le neurone formel

Le neurone artificiel, aussi appelé neurone formel, reprend le fonctionnement du neurone biologique.

2.1. Fonctionnement général

Un neurone reçoit des entrées et fournit une sortie, grâce à différentes caractéristiques :
- Des **poids** accordés à chacune des entrées, permettant de modifier l'importance de certaines par rapport aux autres.
- Une **fonction d'agrégation**, qui permet de calculer une unique valeur à partir des entrées et des poids correspondants.
- Un **seuil** (ou biais), permettant d'indiquer quand le neurone doit agir.
- Une **fonction d'activation**, qui associe à chaque valeur agrégée une unique valeur de sortie dépendant du seuil.

Le neurone formel peut donc se résumer sous la forme suivante :



2.2. Fonctions d'agrégation

Il est possible d'imaginer plusieurs fonctions d'agrégation. Les deux plus courantes sont :
- La somme pondérée.
- Le calcul de distance .

Dans le cas de la somme pondérée, on va simplement faire la somme de toutes les entrées multipliées par leur poids . Mathématiquement cela s'exprime sous la forme :

$$\sum_{i=1}^{n} E_i * w_i$$

Dans le deuxième cas, celui du calcul des distances, on va comparer les entrées aux poids (qui sont les entrées attendues par le neurone), et calculer la distance entre les deux.

Pour rappel, la distance est la racine de la somme des différences au carré, ce qui s'exprime donc :

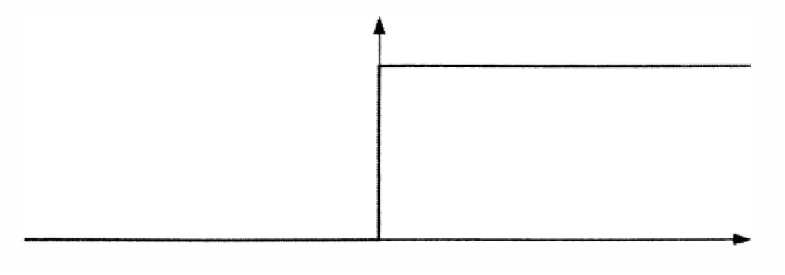
$$\sqrt{\sum_{i=1}^{n}(E_i - w_i)^2}$$

2.3. Fonctions d'activation

Une fois une valeur unique calculée, le neurone compare cette valeur à un seuil et en décide la sortie. Pour cela, plusieurs fonctions peuvent être utilisées. Les trois plus utilisées sont ici présentées.

2.3.1 Fonction "heavyside"

La fonction signe, ou heavyside en anglais, est une fonction très simple : elle renvoie + 1 ou 0. Ainsi, si la valeur agrégée calculée est plus grande que le seuil, elle renvoie + 1 , sinon 0 (ou - 1 selon les applications) .



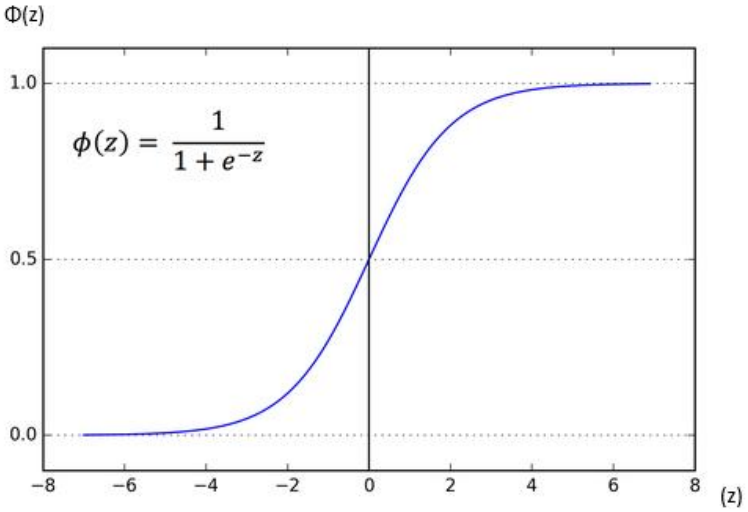
Cette fonction permet par exemple la classification, en indiquant qu'un objet est ou non dans une classe donnée.

2.3.2 Fonction sigmoïde

La fonction sigmoïde utilise une exponentielle. Elle est définie par :

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

Elle est comprise **entre 0 et + 1 , avec une valeur de 0.5 en 0.**
Dans le neurone, la méthode est appelée avec x = valeur agrégée - seuil. Ainsi, on a une sortie supérieure 0.5 si la valeur agrégée est plus grande que le seuil, inférieure à 0.5 sinon.

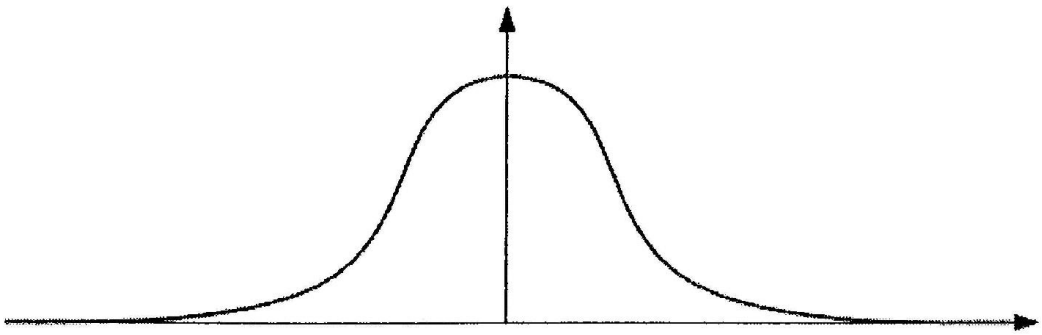


Cette fonction permet un meilleur apprentissage, grâce à sa pente. En effet, il est plus facile de savoir vers quelle direction aller pour améliorer les résultats.

2.3.3 Fonction gaussienne

La dernière fonction très utilisée est la fonction gaussienne. Celle-ci, aussi appelée "courbe en cloche", est symétrique, avec un maximum obtenu en 0.
Son expression est plus complexe que pour la fonction sigmoïde, mais elle peut se simplifier sous la forme suivante, avec k et k' des constantes dépendant de l'écart-type voulu :

$$f(x) = k.e^{-x^2/k'}$$



Là encore, on utilisera la différence entre la valeur agrégée et le seuil comme abscisse.
Cette fonction étant aussi dérivable, elle permet un bon apprentissage.

2.4 Poids et apprentissage

Les neurones formels sont tous identiques. Ce qui va les différencier, ce sont les seuils de chacun ainsi que les poids les liant à leurs entrées.
Sur des fonctions simples, il est possible de déterminer les poids et les seuils directement, cependant ce n'est jamais le cas lorsqu'un réseau de neurones est vraiment utile (donc sur des problèmes complexes) .

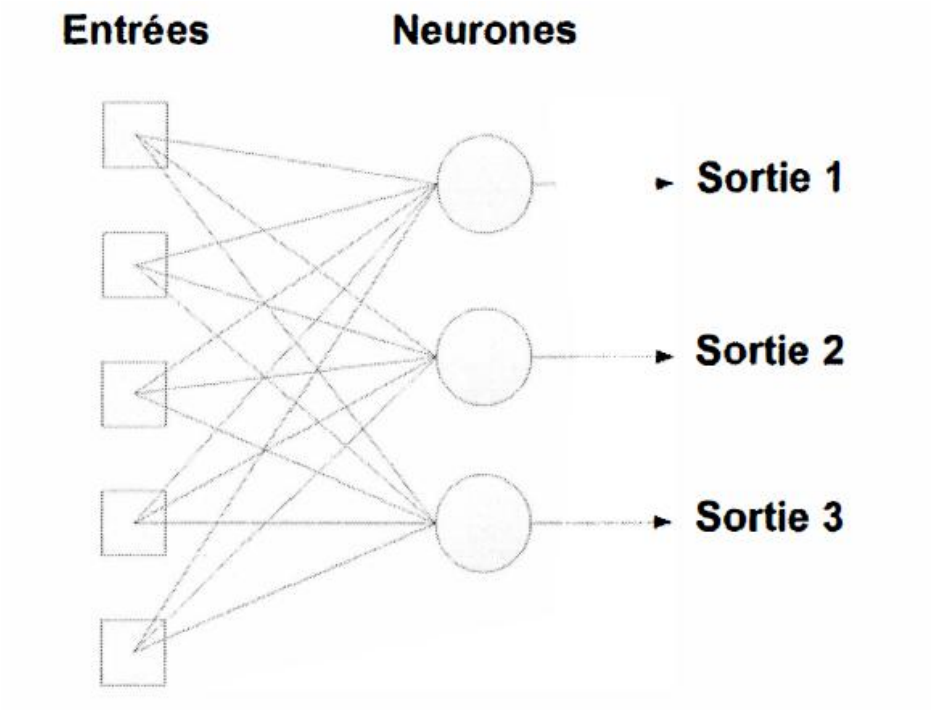
L'apprentissage va donc consister à trouver pour chaque neurone du réseau les meilleures valeurs pour obtenir la sortie attendue. Plus un neurone a d'entrées donc plus il va y avoir de poids à ajuster, et plus l'apprentissage sera complexe et/ou long.

3. Perceptron

Le perceptron est le plus simple des réseaux de neurones.

3.1 Structure

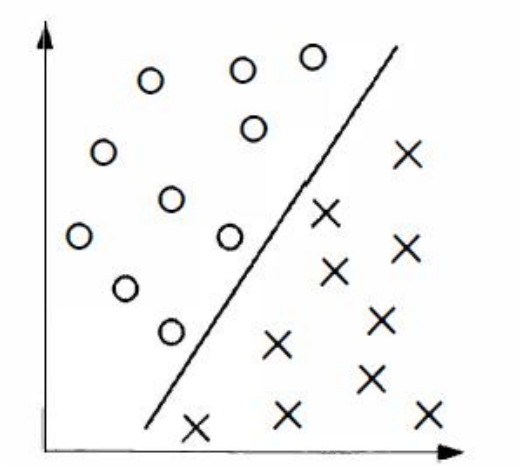
Un perceptron est un réseau contenant p neurones. Chacun est relié aux n entrées.
Ce réseau permet d'avoir p sorties. Généralement, chacune représente une décision ou une classe, et c'est la sortie ayant la plus forte valeur qui est prise en compte.
Avec 3 neurones et 5 entrées, on a donc 3 sorties . Voici la structure obtenue.



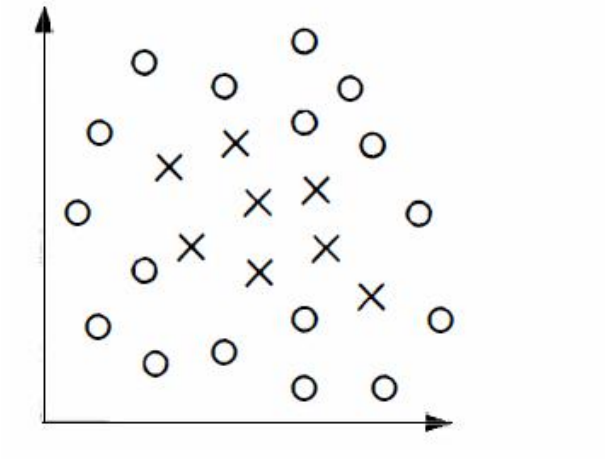
Le réseau possède alors 3 * 5 = 1 5 poids à ajuster, auxquels s'ajoutent 3 valeurs seuils (une par neurone) .

3.2 Condition de linéarité

Le perceptron est simple, et donc facile à mettre en œuvre, mais il a une limite importante : **seuls les problèmes linéairement séparables** peuvent être résolus par celui-ci.
En effet, les fonctions d'activation utilisées (généralement heavyside, plus rarement sigmoïde) présentent un seuil, séparant deux zones de l'espace.
Imaginons un problème possédant deux classes, des croix et des ronds. Si les deux classes sont disposées comme suit, alors le problème est bien linéairement séparable. Une séparation possible est indiquée.



Au contraire, si les points sont présentés comme suit, **les classes ne sont pas linéairement séparables et un réseau de type perceptron ne pourra pas résoudre ce problème**.



Il faut donc, avant d'utiliser un réseau de type perceptron, s'assurer que le problème pourra être résolu. Sinon il faudra opter pour des réseaux plus complexes.

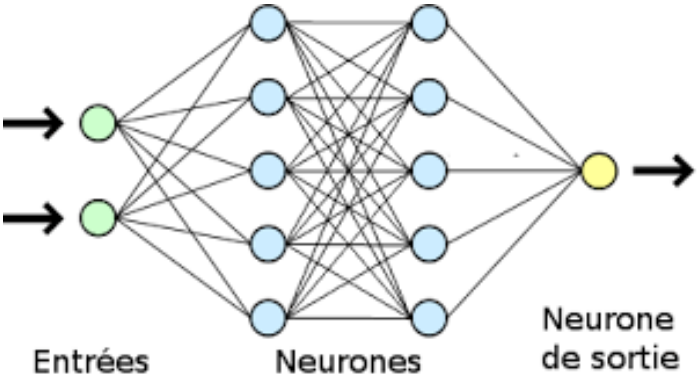
4. Réseaux feed -forward

Les réseaux de type "feed-forward" ou à couches permettent de dépasser les limites des perceptrons. En effet, ceux-ci ne sont plu s limités aux problèmes linéairement séparables.

Ils sont composés d'une ou plusieurs couches cachées de neurones, reliées aux entrées ou aux couches précédentes, et une couche de sortie, reliée aux neurones cachés. On les appelle feed-forward car l'information ne peut aller que des entrées aux sorties, sans revenir en arrière.

Il est possible de trouver des réseaux avec **plusieurs couches cachées**, cependant ces réseaux apportent plus de complexité pour des capacités équivalentes à des réseaux à une seule couche cachée. Ce sont donc ces derniers qui sont les plus utilisés.

On obtient le réseau suivant si l'on a 2 entrées et une sorties, avec 2 couches cachées.



Dans ce cas-là, il faut ajuster les poids et seuils de tous les neurones cachés ainsi que les poids et seuils des neurones de sortie. **De plus, aucune règle ne permet de connaître le nombre de neurones cachés idéal pour un problème donné**. Il est donc nécessaire de tester plusieurs valeurs et de choisir celle donnant les meilleurs résultats.
Les réseaux utilisant des neurones de type perceptron sont dits MLP pour MultiLayer Perceptron, alors que ceux utilisant des neurones à fonction d'activation gaussienne sont dit RBF (pour Radial Basis Function). Les réseaux MLP et RBF sont les plus courants.

5. Apprentissage

L'étape la plus importante dans l'utilisation d'un réseau de neurones est l'apprentissage des poids et seuils. Cependant, les choisir ou les calculer directement est impossible sur des problèmes complexes.
Il est donc nécessaire d'utiliser des algorithmes d'apprentissage. On peut les séparer dans trois grandes catégories.

5.1 Apprentissage non supervisé

L'apprentissage non supervisé est la forme **la moins courante d'apprentissage**. En effet, dans cette forme d'apprentissage, il n'y a pas de résultat attendu.
On utilise cette forme d'apprentissage pour faire du clustering : on a un ensemble de données, et on cherche à déterminer des classes de faits.
Par exemple, à partir d'une base de données de clients, on cherche à obtenir les différentes catégories, en fonction de leurs achats ou budgets. On ne sait pas a priori combien il y a de catégories ou ce qu'elles sont .

5.2 Apprentissage supervisé

L'apprentissage supervisé est **sûrement le plus courant**. Il est utilisé pour des tâches d'estimation, de prévision, de régression ou de classification.
Dans l'apprentissage supervisé, un ensemble d'exemples est fourni à l'algorithme d'apprentissage. Celui-ci va comparer la sortie obtenue par le réseau avec la sortie attendue.
Les poids sont **ensuite modifiés pour minimiser cette erreur**, jusqu'à ce que les **résultats soient satisfaisants pour tous les exemples fournis.**
Cette approche est utilisée à chaque fois que les exemples sont présentés au réseau de neurones les uns à la suite des autres, sans liens entre eux.

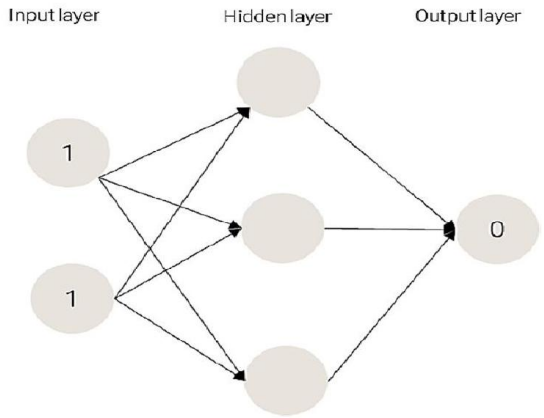
Exemple :

Forward propagation

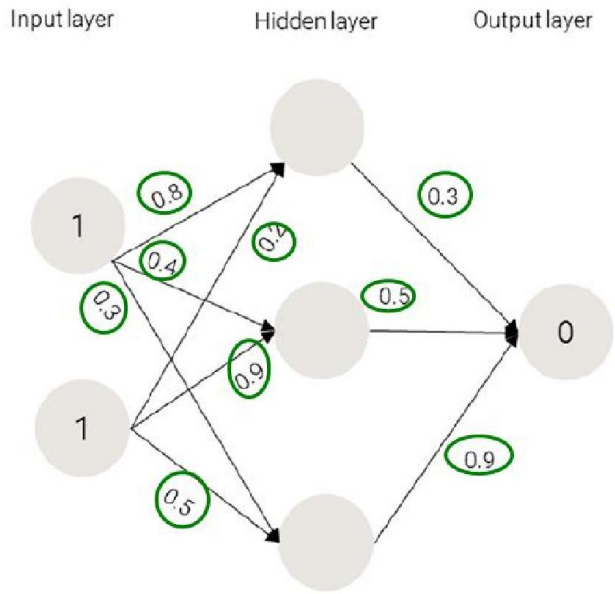
Apprentissage de la function XOR.

Input	Output
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

On prend l''entrée (1,1) et output 0 comme exemple:



Initialiser les poids (entre 0-1):

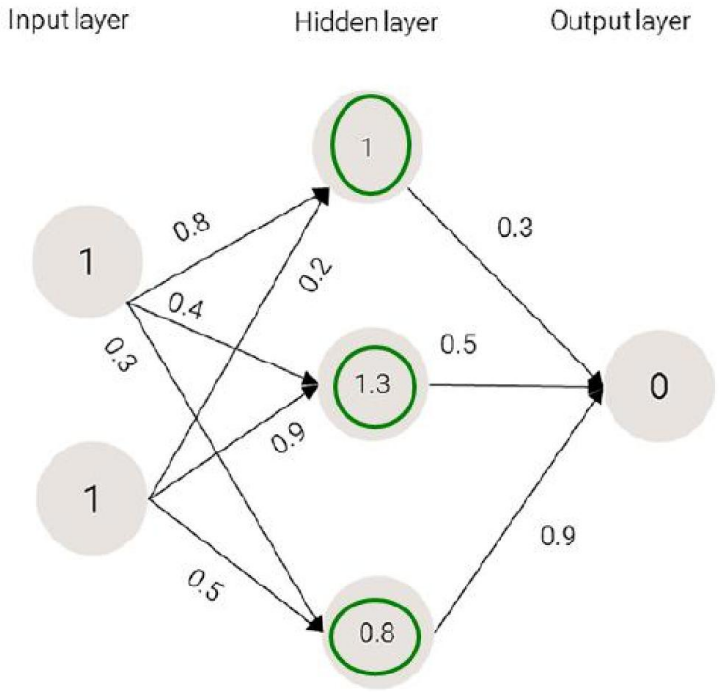


Calculer les valeurs des nœuds :

$$1 \times 0.8 + 1 \times 0.2 = 1$$

$$1 \times 0.4 + 1 \times 0.9 = 1.3$$

$$1 \times 0.3 + 1 \times 0.5 = 0.8$$



Utiliser la function d'activation :

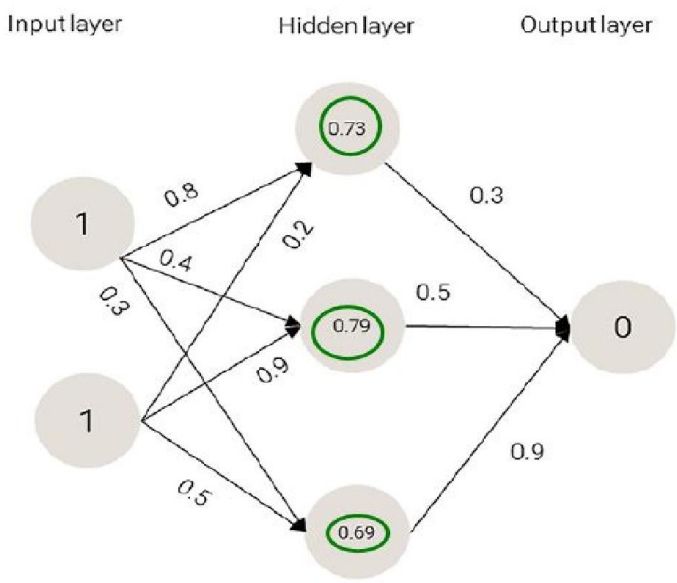
$$\text{Sigmoid} = 1/\left(1+e^{-x}\right)$$

Application sur les 3 valeurs:

$$\text{Sigmoid}(1.0) = 0.731$$

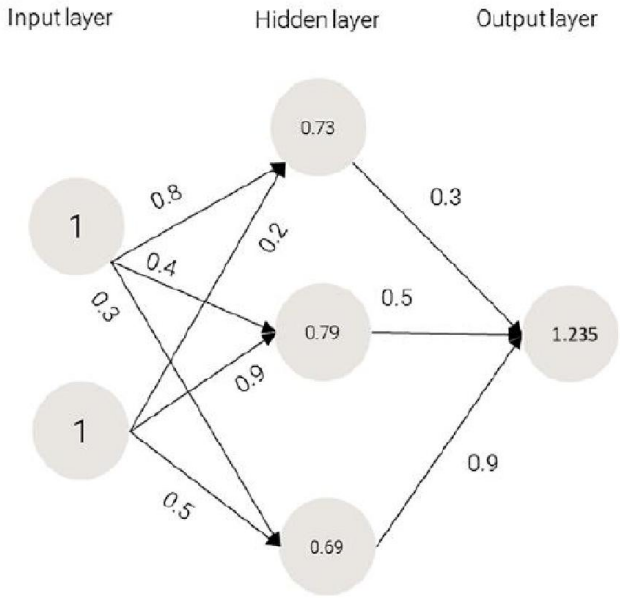
$$\text{Sigmoid}(1.3) = 0.785$$

$$\text{Sigmoid}(0.8) = 0.689$$



Calculer la valeur de la 3 ^{eme} couche (sortie, résultat) :

$$0.73 \times 0.3 + 0.79 \times 0.5 + 0.69 \times 0.9 = 1.235$$



Calculer l'erreur de prediction

On utilise la moyenne MSE Mean Squared Error:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

predicted vaue actual value

MSE = $(1.235 - 0)^2$

Backpropagation

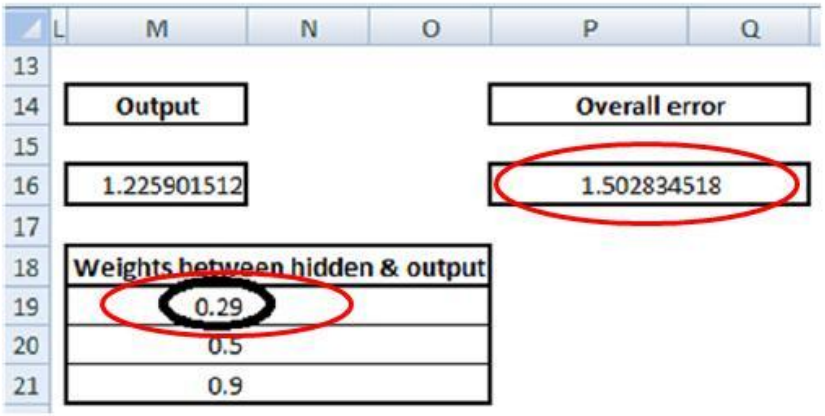
Commencer par la 1 ère couche, ajuster les poids pour minimiser l'erreur.

Changer les poids + ,- pour déterminer si l'erreur augmente ou diminue.

A partir de l'étape précédente:

MSE = $1.235^2 = 1.52$, nous changeons le poids pour voir son influence sur l'erreur.

Changer le poids de 0.3 entre la dernière couche et la couche précédente à 0.29 , on obtient une erreur de 1.50.



	L	M	N	O	P	Q
13						
14		Output			Overall error	
15						
16		1.225901512			1.502834518	
17						
18		Weights between hidden & output				
19		0.29				
20		0.5				
21		0.9				

Donc, les poids doivent être minimisés.

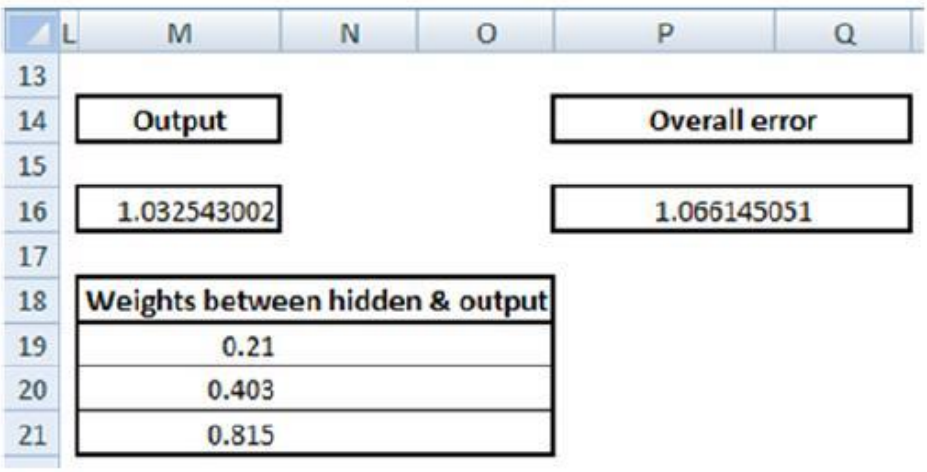
Mais, comment?

$$0.3 - 0.05 \times (\text{Reduction in error because of change in weight})$$

Ici 0.3 est la valeur initiale, et 0.05 est learning rate.

$$0.3 - 0.05 \times ((1.52 - 1.50) / 0.01) = 0.21.$$

De la même façon, calculer les autres poids de cette couche.



	L	M	N	O	P	Q
13						
14		Output			Overall error	
15						
16		1.032543002			1.066145051	
17						
18		Weights between hidden & output				
19		0.21				
20		0.403				
21		0.815				

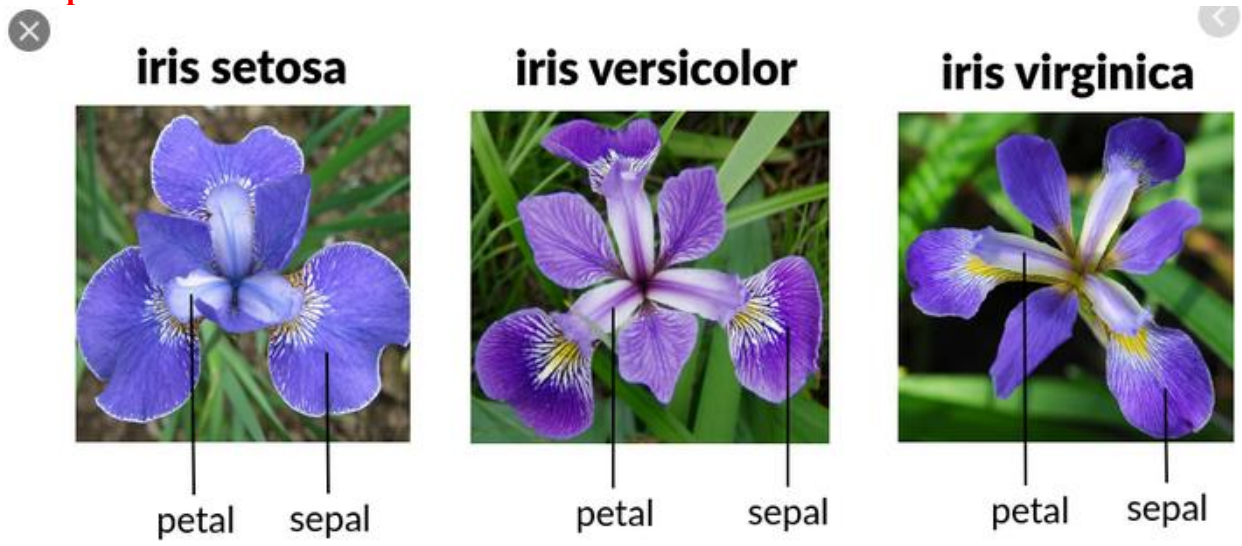
Nous trouvons que cette ajustement entre la couche cachée et la couche sortie réduise MSE de 1.52 à 1.06

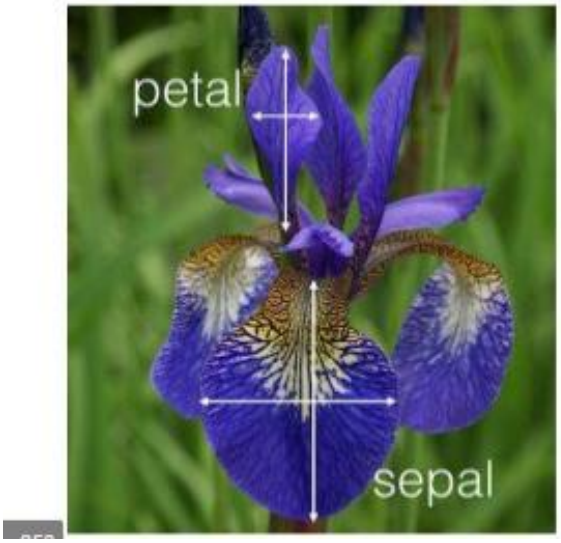
Apres, nous ajustons les poids de la couche d'entrée et la couche cachée.

Original weight	Updated weight	Decrease in error
0.8	0.7957	0.0009
0.4	0.3930	0.0014
0.3	0.2820	0.0036
0.2	0.1957	0.0009
0.9	0.8930	0.0014
0.5	0.4820	0.0036

Le processus est répété jusqu'a l'obtention d'une seul d'erreur prédéfinie par l'utilisateur.

TP : problème de classification des fleurs iris





Training / test data

Features				Labels
Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	Iris setosa
4.9	3.0	1.4	0.2	Iris setosa
7.0	3.2	4.7	1.4	Iris versicolor
6.4	3.2	4.5	1.5	Iris versicolor
6.3	3.3	6.0	2.5	Iris virginica
5.8	3.3	6.0	2.5	Iris virginica

La bd contient 150 exemples.

Accuracy = nombre de bonnes classification / taille de la bd.