Deep Learning : Réseau de Neurones

Abdoul Kader KABORE

abdoulkader.kabore@protonmail.com

Objectifs de ce module

- **Aborder** une approche radicalement différente des autres modes de programmation.
- **Comprendre** les principaux concepts de simulation informatique d'un neurone.
- **Maîtriser** deux algorithmes majeurs d' "apprentissage".
- **Evaluer** les performances et les limites d'un réseau neuronal.
- **Structurer et tester** des "mini-réseaux" (ex: OCR).

Plan du module

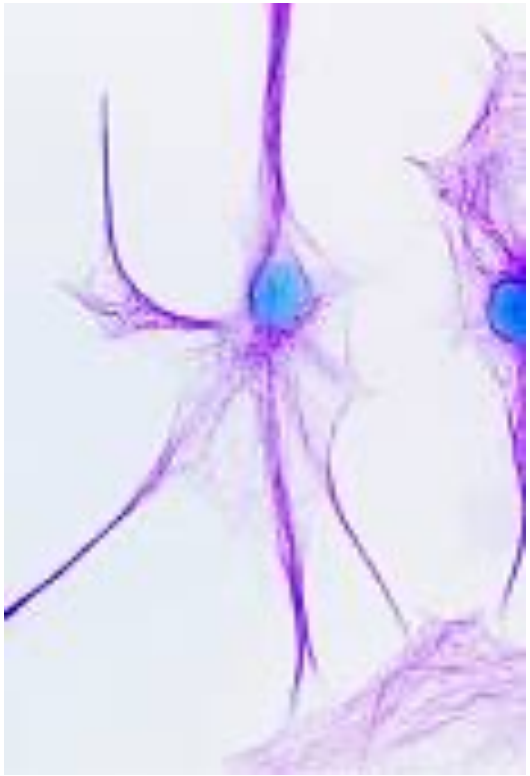
Voici les parties que nous allons aborder:

- **Notions de base**
- **Le perceptron**
- **Réseaux multicouches à rétro-propagation de l'erreur**

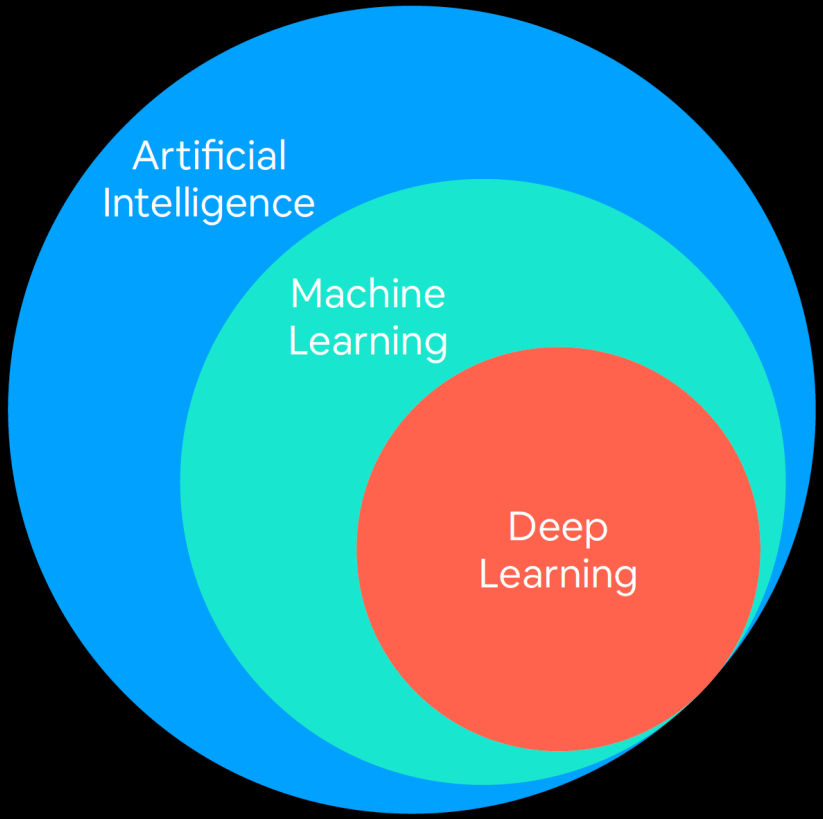
Plan de la partie

Voici les chapitres que nous allons aborder:

- Historique et développements
- Modélisation du neurone
- Types d’apprentissage

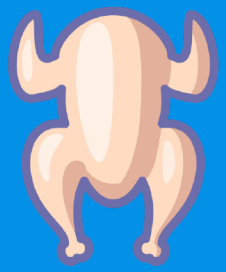


Machine Learning vs. Deep Learning



Traditional programming

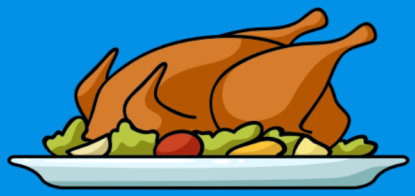
Inputs



Rules

1. Cut vegetables
2. Season chicken
3. Preheat oven
4. Cook chicken for 30-minutes
5. Add vegetables

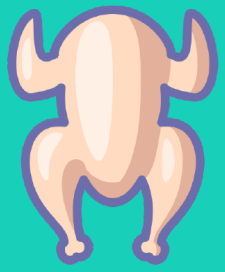
Output



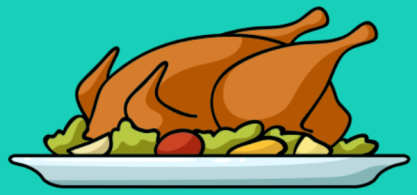
Starts with **Makes**

Machine learning algorithm

Inputs



Output



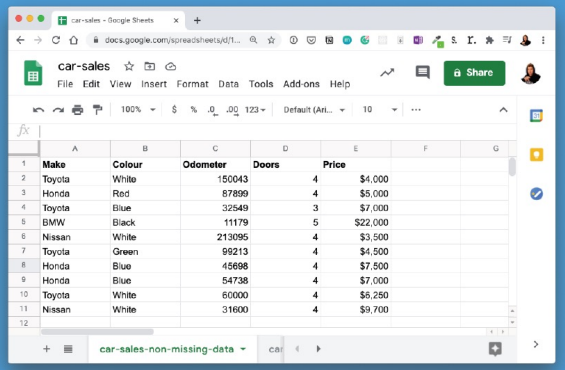
Rules

1. Cut vegetables
2. Season chicken
3. Preheat oven
4. Cook chicken for 30-minutes
5. Add vegetables

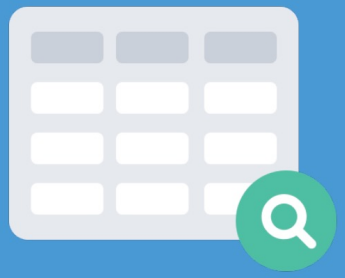
Starts with **Figures out**

Machine Learning vs. Deep Learning

Machine Learning

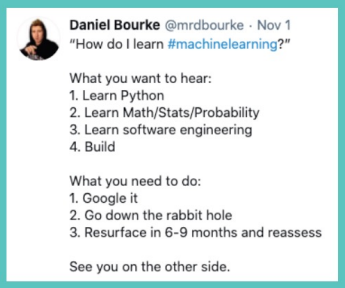


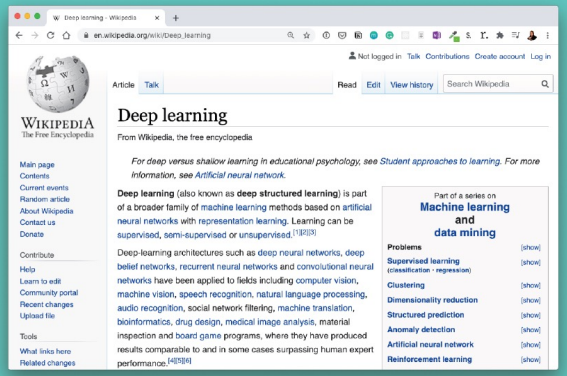
	Make	Colour	Odometer	Doors	Price
2	Toyota	White	150043	4	$4,000
3	Honda	Red	87899	4	$5,000
4	Toyota	Blue	32549	3	$7,000
5	BMW	Black	11179	5	$22,000
6	Nissan	White	213095	4	$3,500
7	Toyota	Green	99213	4	$4,500
8	Honda	Blue	45698	4	$7,500
9	Honda	Blue	54738	4	$7,000
10	Toyota	White	60000	4	$6,250
11	Nissan	White	31600	4	$9,700



Structured data

Deep Learning





Unstructured data

Historique et développements

Approches informatiques pour résoudre un problème :

- Approche algorithmique (programmation complète)
- Création des « moteurs d'inférence » (programme qui raisonne ; règles SI..ALORS.. ; système expert)
- Approche connexionniste : le réseau s'organise par apprentissage (pas de programmation)

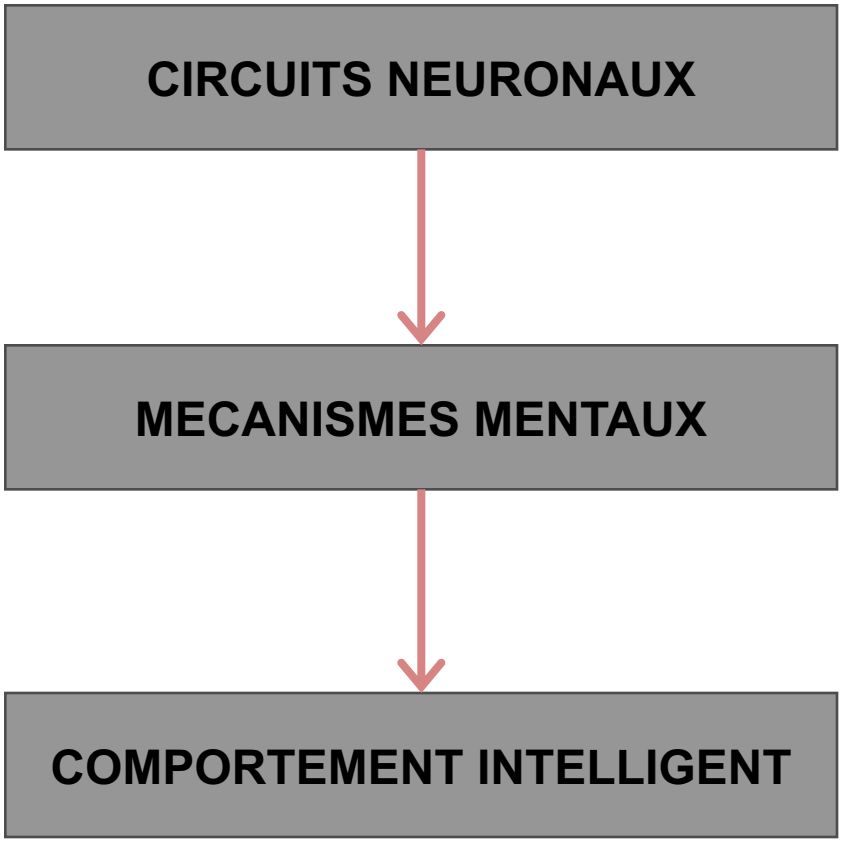
Historique et développements

Caractéristiques de l'approche connexionniste (réseaux neuronaux) :

- Calcul non-algorithmique
- Information et mémoire distribuée dans le réseau
- Architecture massivement parallèle (processeurs élémentaires interconnectés)
- Apprentissage par entraînement sur des exemples
- Inspiré du fonctionnement du cerveau

Historique et développements

Niveaux de modélisation :



Historique et développements

Historique

- 1943 J.Mc Culloch et W.Pitts établissent le "modèle logique" du neurone qui ouvre la voie à des modèles techniques.
- 1949 D.Hebb élabore une théorie formelle de l'apprentissage biologique par modifications des connexions neuronales.
- 1957 F.Rosenblatt réalise le Perceptron, le premier modèle technique basé sur la modification des poids.
- 1960 B.Widrow réalise Adaline (Adaptive Linear Element), un réseau adaptatif de type perceptron.
- 1969 M.Minsky et S.Papert émettent des critiques et démontrent les limites des modèles neuronaux de type perceptron.
- La recherche s'arrête durant un peu plus d'une dizaine d'années.
- 1982 J.Hopfield (physicien) propose une nouvelle approche des réseaux neuronaux basée sur l'analogie avec les milieux à grand nombre de particules. Cela relance l'intérêt pour les réseaux neuronaux
- depuis 1984 : développement croissant du domaine connexionniste aussi bien en IA qu'en informatique.

Historique et développements

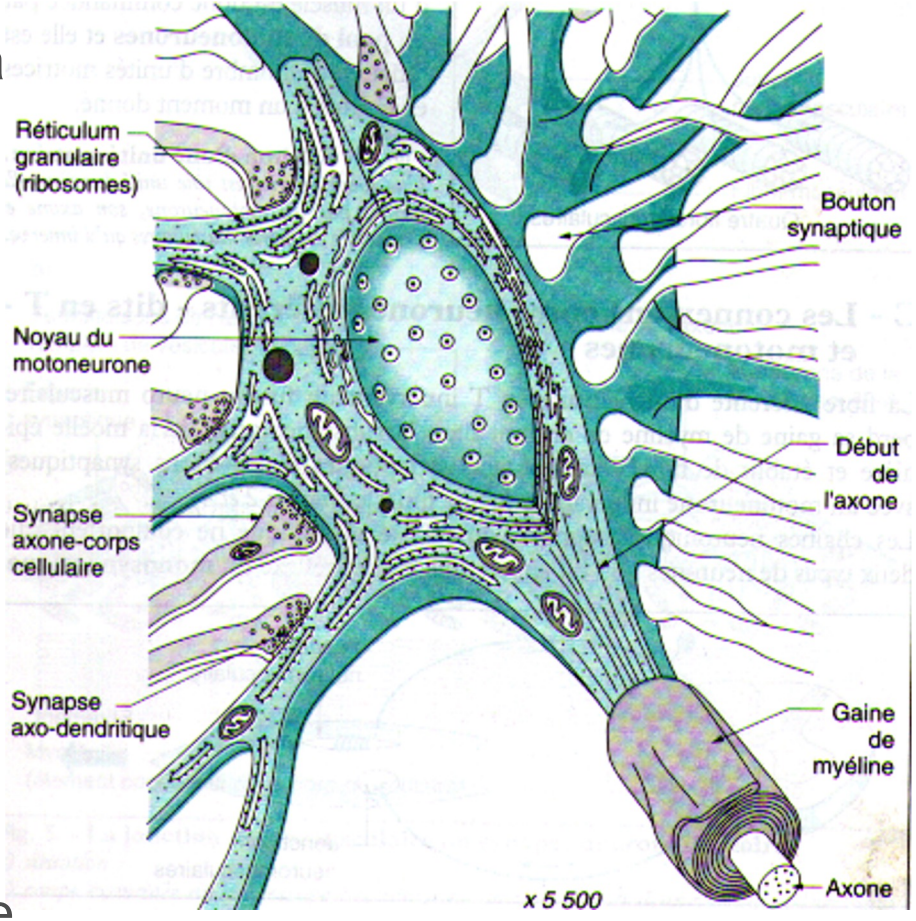
Applications

- Traitement des images
- Identification des signatures
- Reconnaissance des caractères (dactylos ou manuscrits)
- Reconnaissance de la parole
- Reconnaissance de signaux acoustiques (bruits sous-marins, …)
- Extraction d'un signal du bruit
- Contrôle de systèmes asservis non-linéaires (non modélisables)
- Robotique (apprentissage de tâches)
- Aide à la décision (domaine médical, bancaire, management, …)

Modélisation du neurone

Le neurone réel

- Dans un cerveau, il y a 10^12 neurones avec 10^3 à 10^4 connexions par neurone.
- Dendrite : récepteur des messages
- Corps : génère le potentiel d'action (la réponse)
- Axone : transmet le signal aux cellules suivantes
- Synapse : jonction axone - dendrite (plus ou moins passante)
- Neurone : élément autonome dépourvu d'intelligence



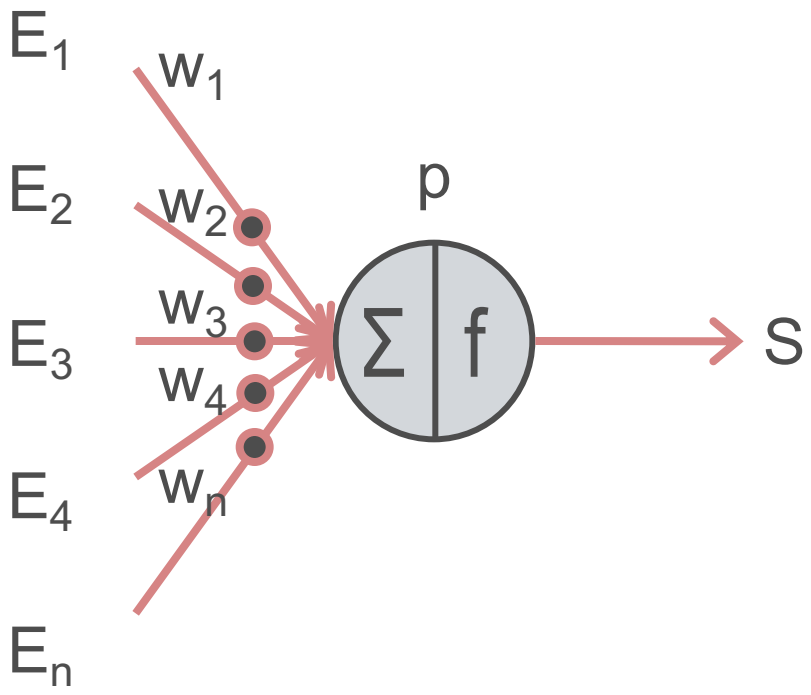
Modélisation du neurone

La modélisation du système nerveux biologique repose sur la correspondance suivante

Système nerveux	Système de calcul
Neurone	Processeur
Dendrite	Fonction de combinaison
Corps du neurone	Fonction de transfert
Axone	Elément de sortie
Synapse	Poids

Modélisation du neurone

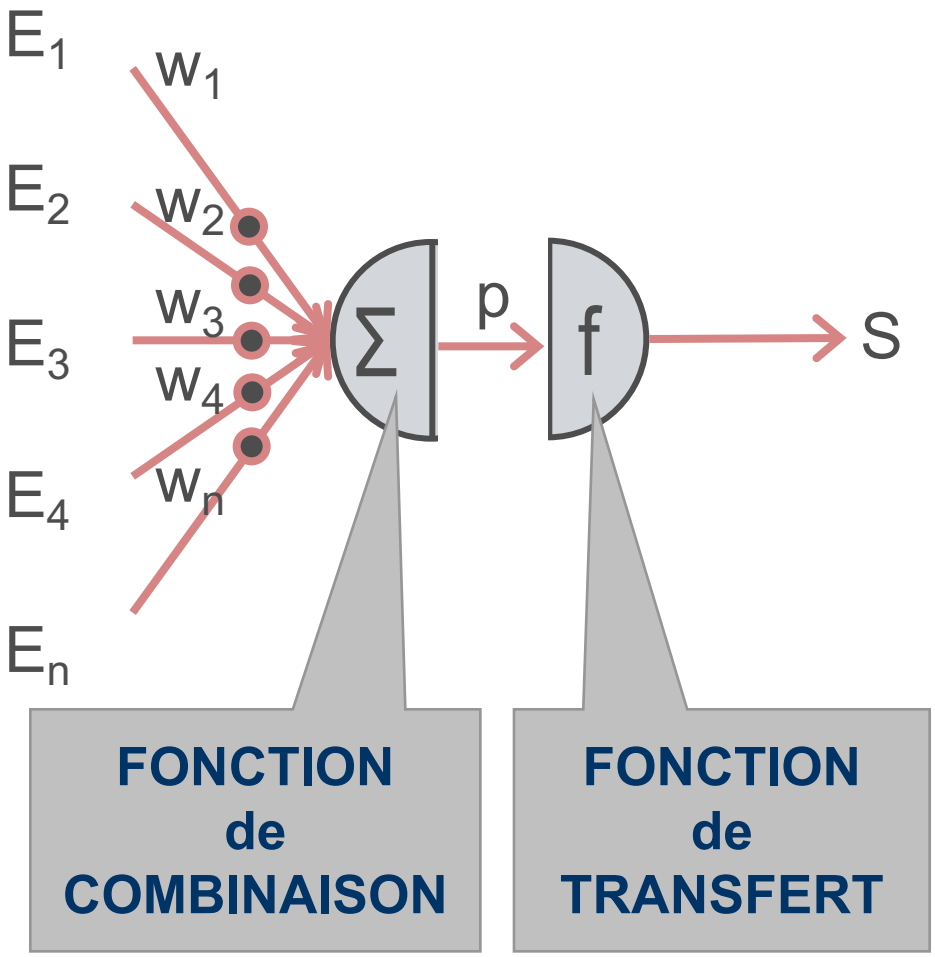
La représentation graphique (conventionnelle) d'un neurone formel modélisé par Mc Culloch et Pitts.



Modélisation du neurone

Les éléments constitutifs du neurone artificiel

- Les entrées "**E**" du neurone proviennent soit d'autres éléments "processeurs", soit de l'environnement.
- Les poids "**W**" déterminent l'influence de chaque entrée.
- La fonction de combinaison "**p**" combine les entrées et les poids.
- La fonction de transfert calcule la sortie "**S**" du neurone en fonction de la combinaison en entrée.

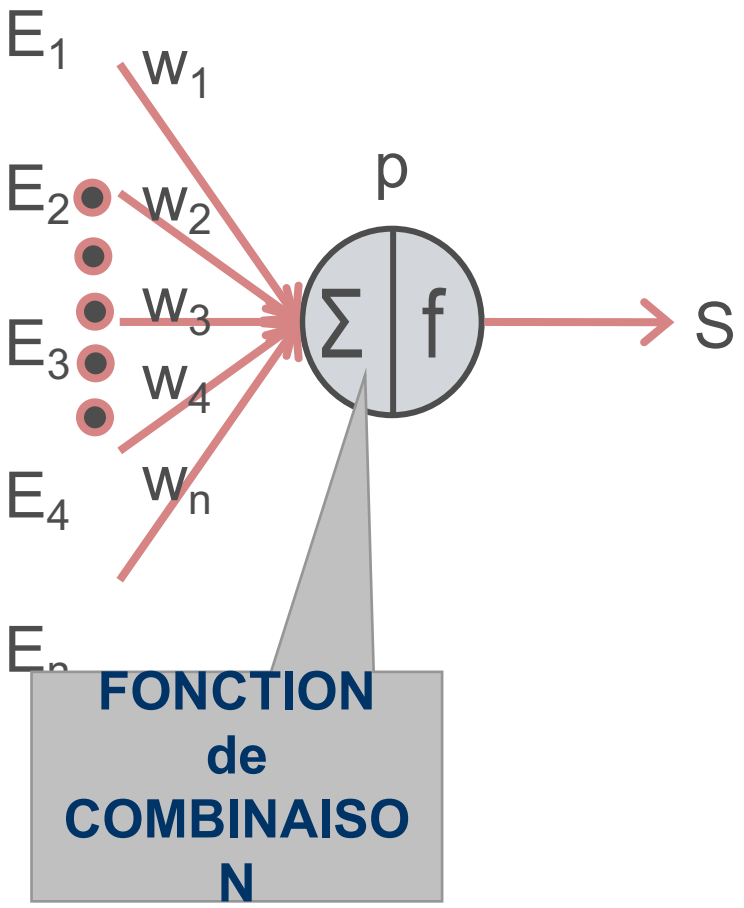


Modélisation du neurone

La Fonction de Combinaison calcule l'influence de chaque entrée en tenant compte de son poids. Elle fait la somme des entrées pondérées :

$$p = \sum W_i E_i$$

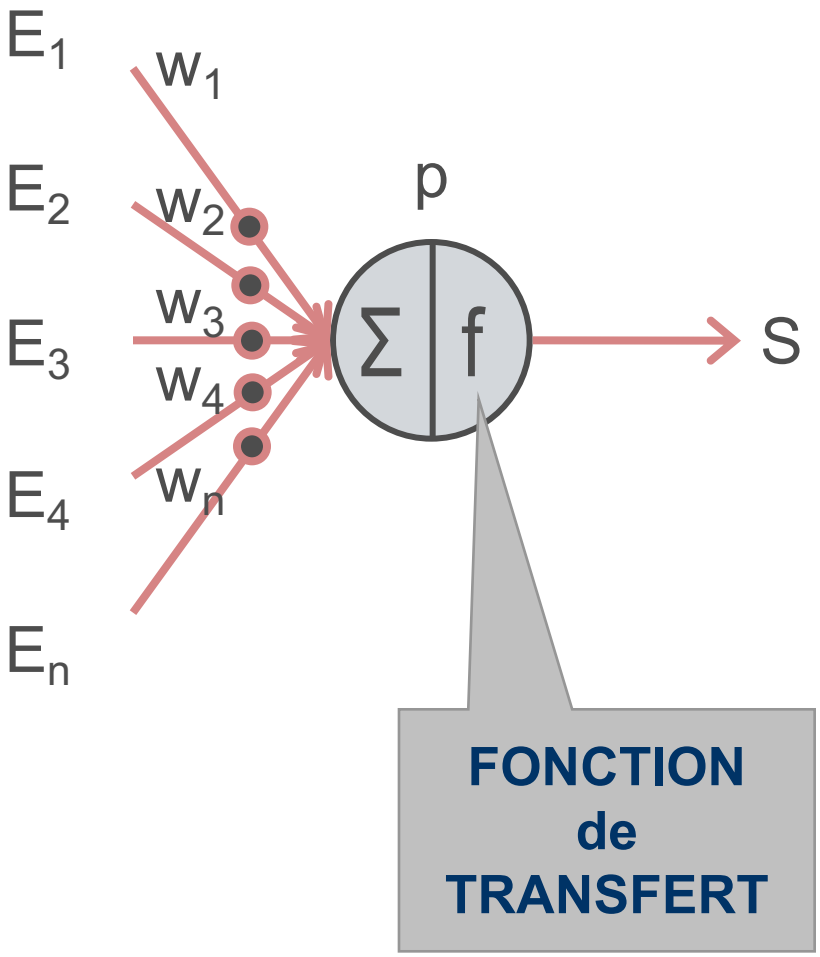
- W_i :
- Poids de la connexion à l'entrée i.
- E_i :
- Signal de l'entrée i.



Modélisation du neurone

La Fonction de Transfert détermine l'état du neurone (en sortie)

- Calcul de la sortie :
- ***S = f(p)***
- ou encore :
- ***S = f(∑ W_i E_i)***
- La fonction de transfert "**f**" peut avoir plusieurs formes.



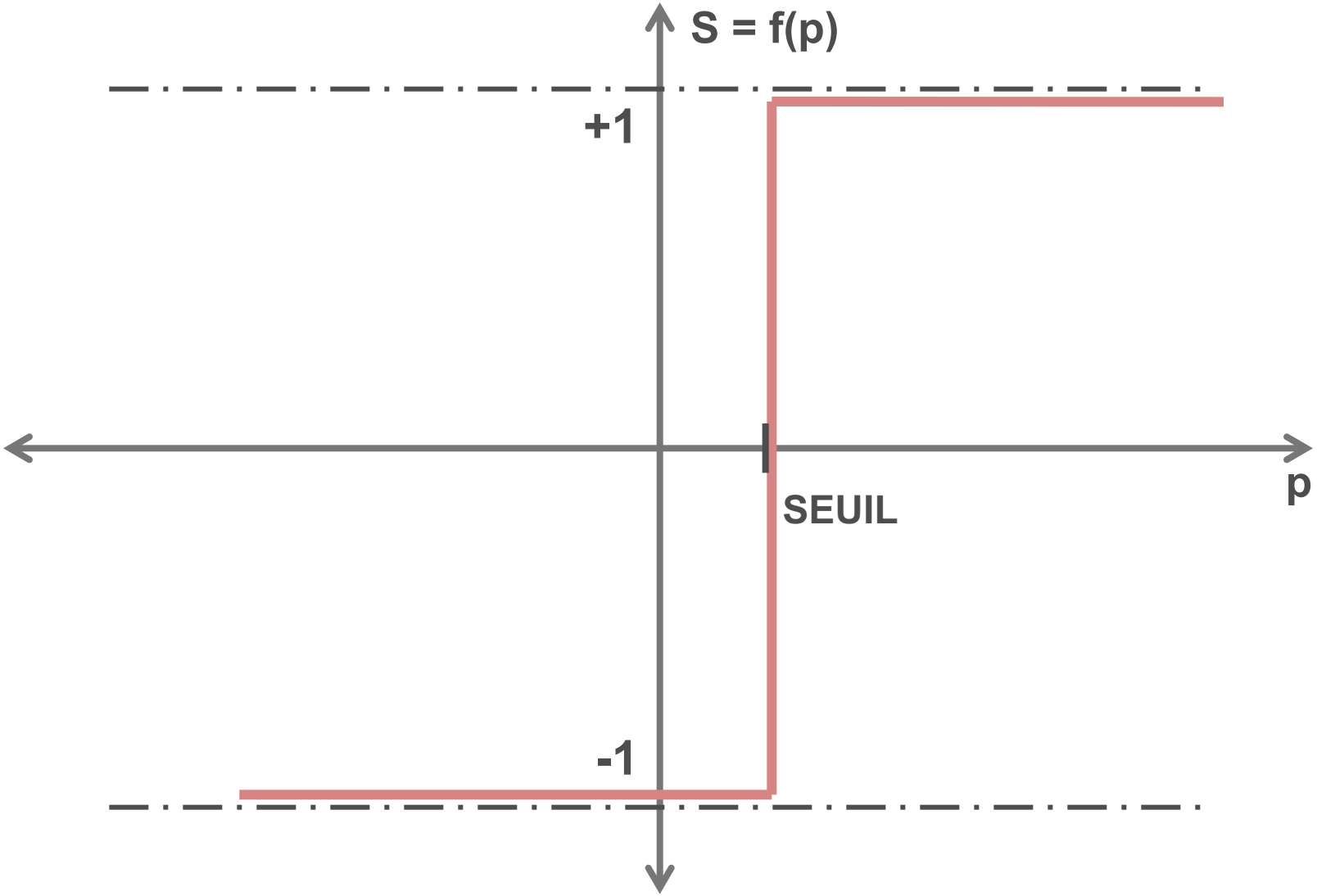
Modélisation du neurone

La fonction ‘f’ peut être de la forme :

- Fonction en échelon.
- Fonction linéaire par morceaux.
- Fonction dérivable (sigmoïde).

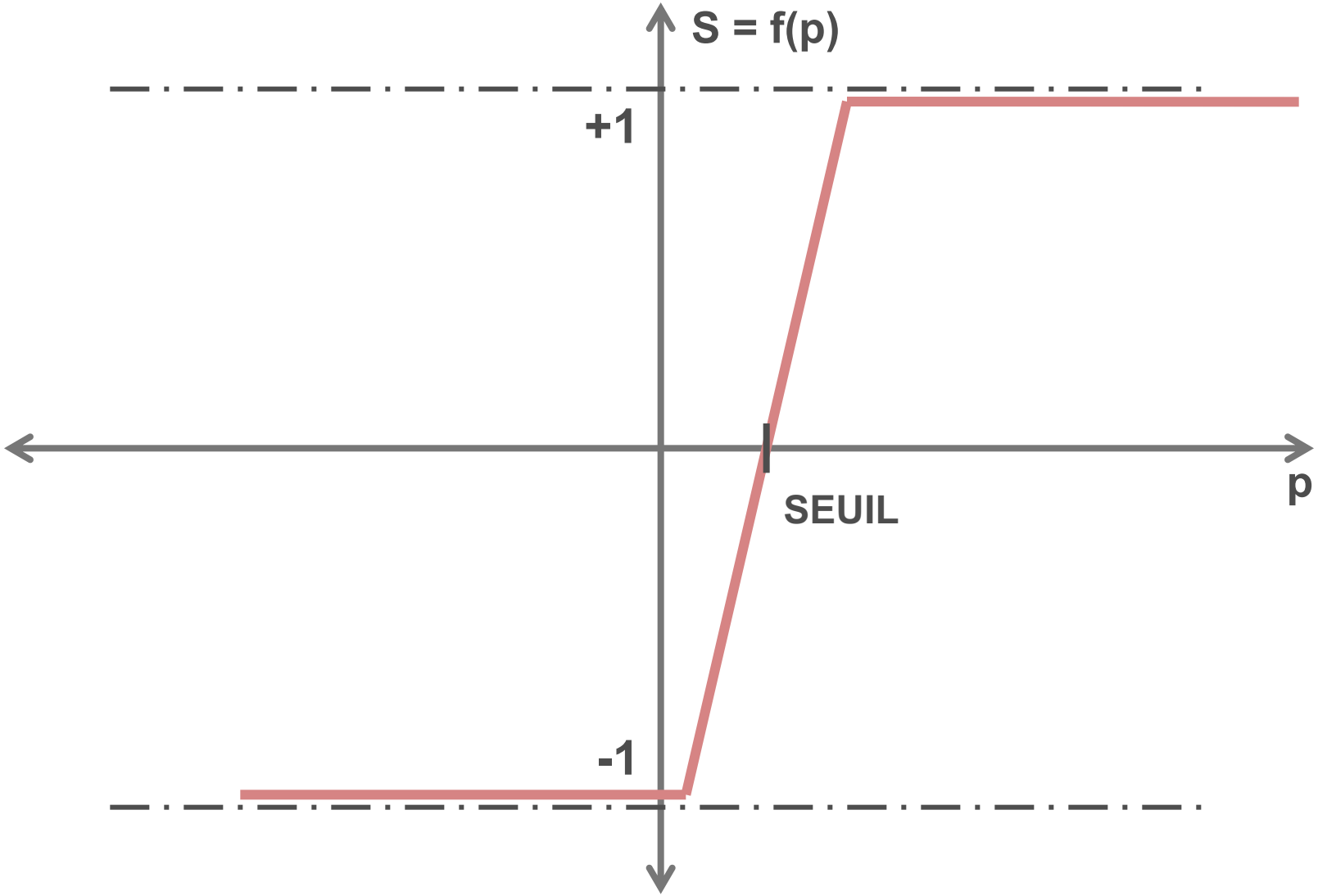
Modélisation du neurone

Fonction de transfert en échelon :



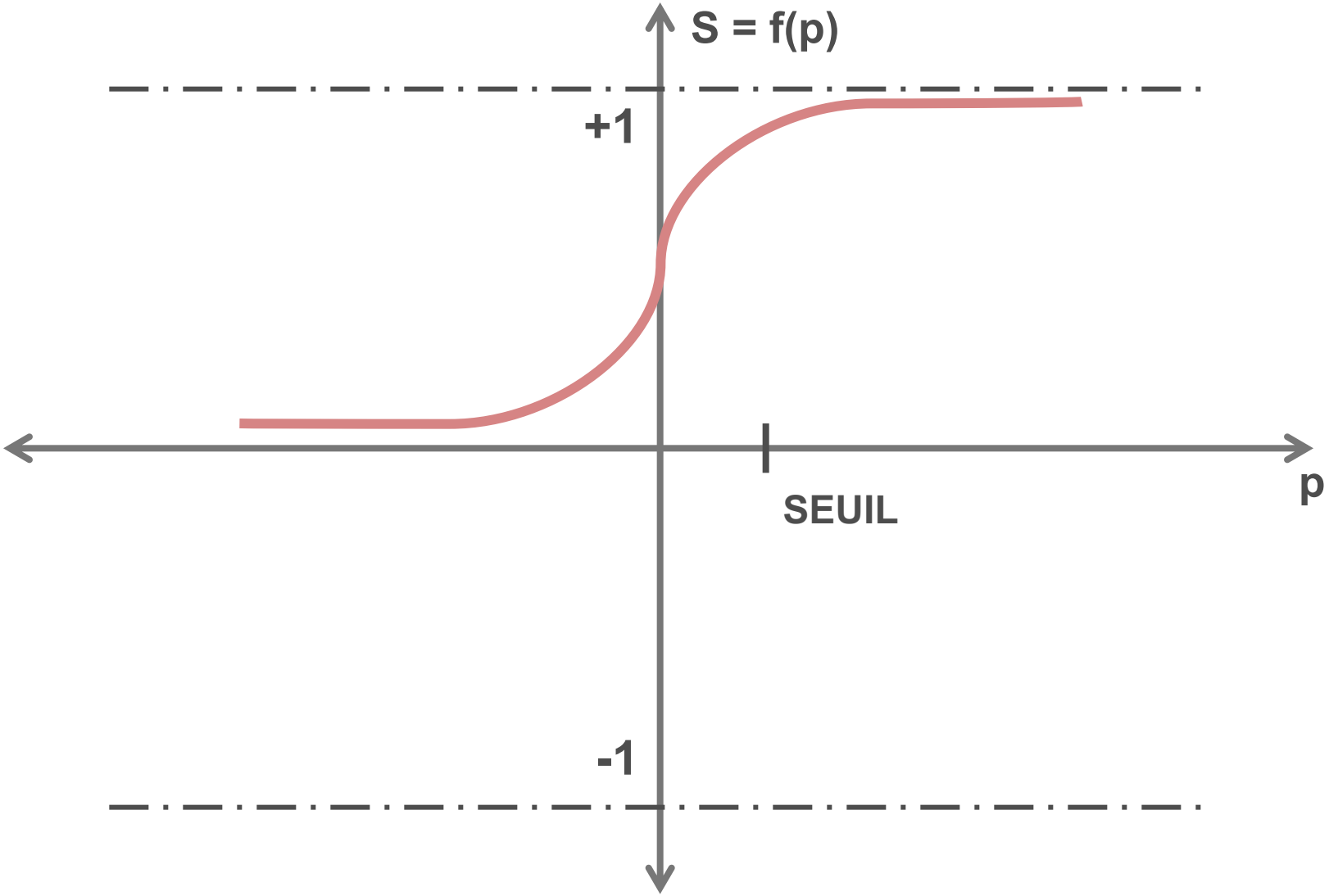
Modélisation du neurone

Fonction de transfert linéaire par morceaux :



Modélisation du neurone

Fonction de transfert dérivable (sigmoïde) :



Types d'apprentissage

Le but des réseaux neuronaux est d'apprendre à répondre correctement à différentes entrées.
<u>Moyen :</u> modification des poids par apprentissage supervisé, ou non supervisé.

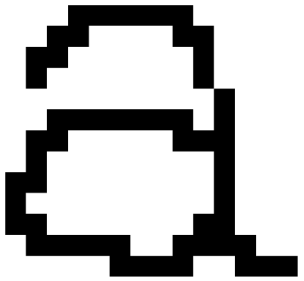
- **Apprentissage supervisé**: un système "instructeur" corrige les réponses éronnées.

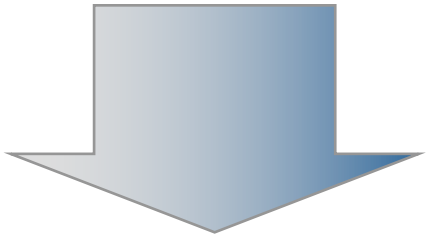
- **Apprentissage non supervisé:** le système neuronal apprend tout seul en formant des classes d'entrées à réponses communes.

Types d'apprentissage

Apprentissage supervisé (ex: OCR)

- Association imposée entre un vecteur d'entrée (forme multidimensionnelle) et un vecteur de sortie (la réponse désirée).
- L'erreur est calculée à chaque essai afin de corriger les poids.
- Les poids sont modifiés jusqu'à l'erreur minimale, voire aucune erreur.



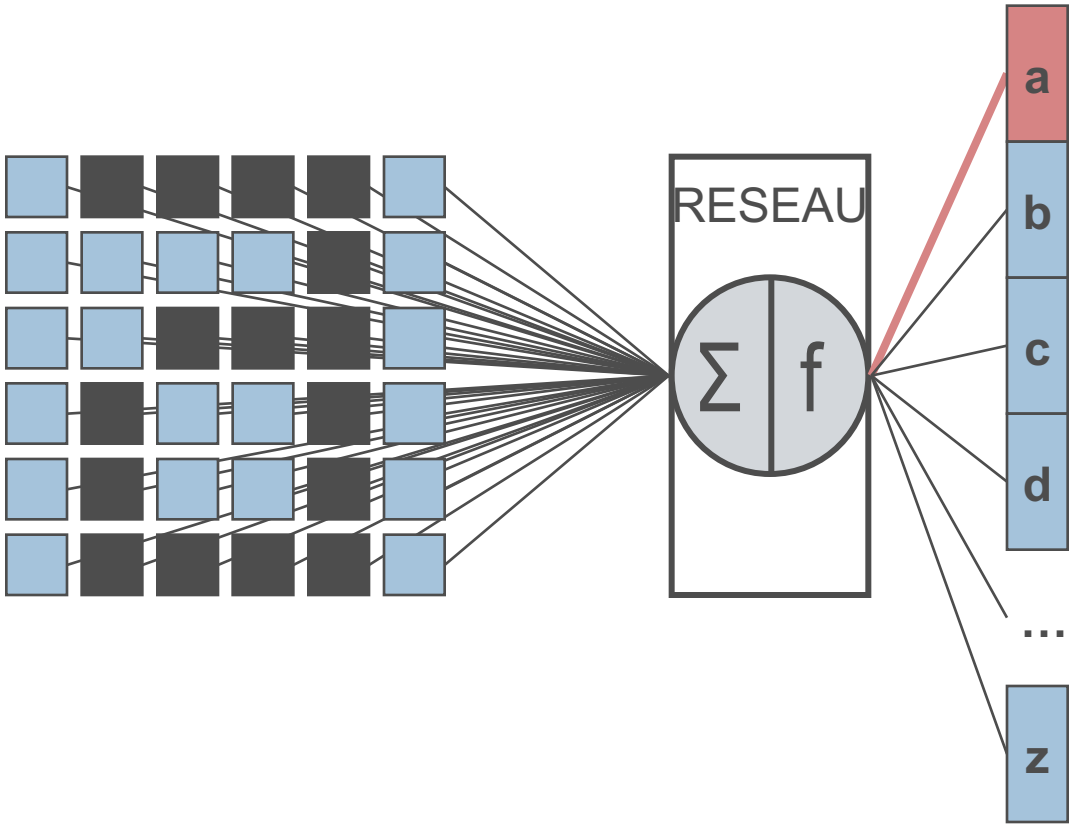


a

Types d'apprentissage

Apprentissage supervisé (ex: OCR)

- La réponse attendue est le "a". Un seul et unique vecteur de sortie doit être activé.



Types d'apprentissage

Apprentissage non supervisé

- Pas d'indication sur les erreurs.
- Le réseau détecte les caractéristiques communes des différentes entrées.
- Il tente ainsi de former des « classes » de façon autonome.
- Il apprend à donner des réponses aux classes

Pause-réflexion sur cette 1ère partie

Avez-vous des questions ?

Le perceptron

Plan de la partie

Voici les chapitres que nous allons aborder:

- Les phases apprentissage-utilisation
- L'algorithme d'apprentissage
- Apprentissage des fonctions logiques
- Les limites du perceptron

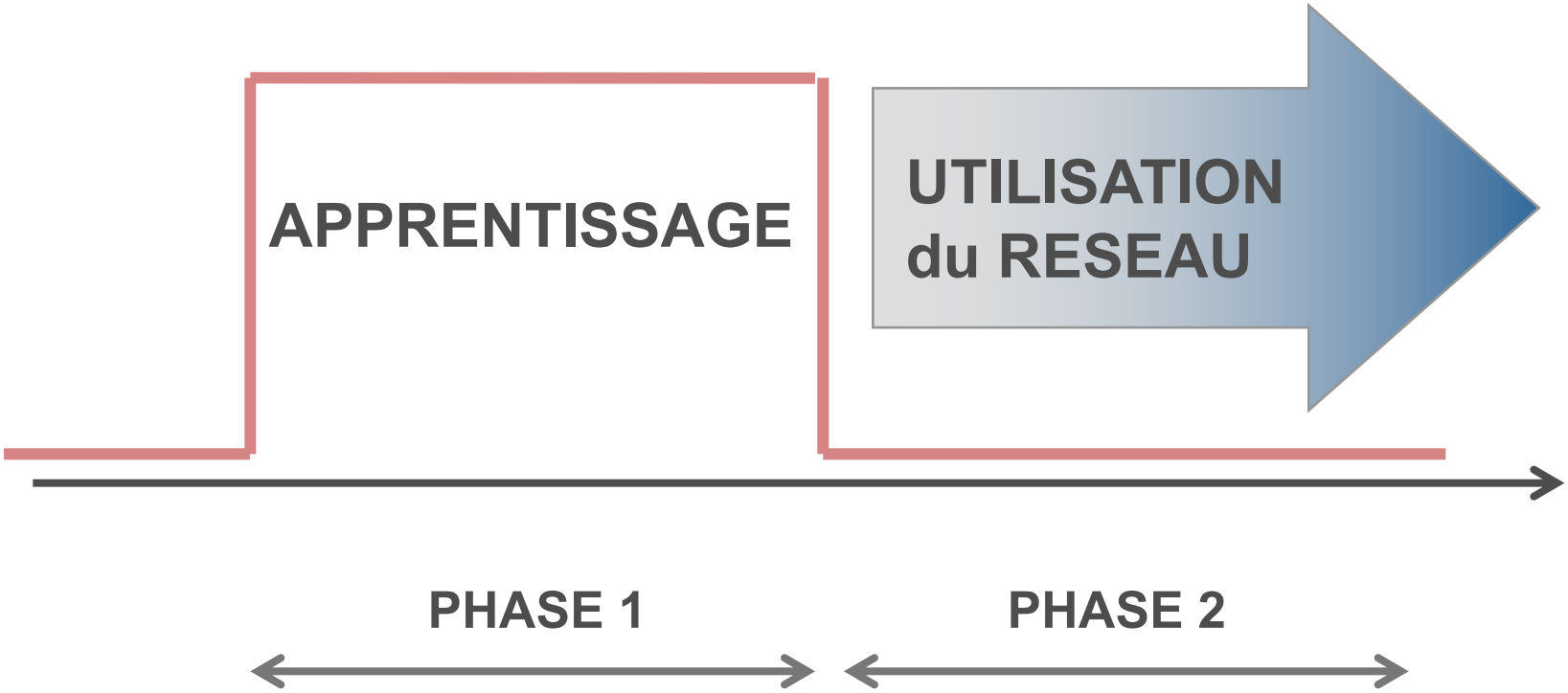
Les phases apprentissage-utilisation

- Phase 1: **APPRENTISSAGE**, le concept du Perceptron est basé sur un algorithme d'apprentissage dont l'objectif est de corriger les poids de pondération des entrées afin de fournir une activité (en sortie) en adéquation avec les éléments à apprendre.

- Phase 2: **UTILISATION**, une fois les exemples appris, chaque neurone active ou non sa sortie (en correspondance avec le domaine acquis), en fonction des éléments appliqués en entrée.

Les phases apprentissage-utilisation

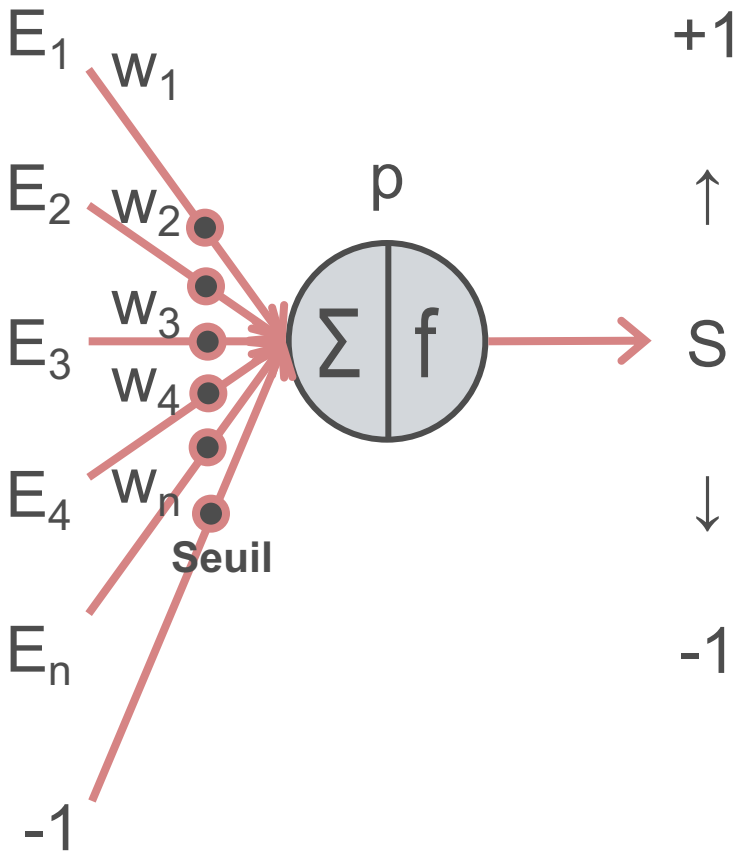
Toute utilisation du réseau doit être précédée d'une phase d'apprentissage durant laquelle on lui présente des exemples type.



L'algorithme d'apprentissage

Classes et séparabilité linéaire

- Le perceptron est un **classificateur linéaire**
- Il réalise une partition de son espace d'entrée (E1,…,En) en deux, ou plusieurs classes C1, …, Cm. séparables linéairement
- On considère deux classes :
  - C1 (S = +1)
  - C2 (S = -1)



L'algorithme d'apprentissage

Algorithme du perceptron (algorithme de principe)

1: **INITIALISATION:** W1 et W2:[-1;+1], SEUIL et PAS:[0;+1]
   Base d'apprentissage: (E1;E2)-> "Sortie correcte"
   ET logique:(+1;+1)->+1;(-1;+1)->-1;(-1;-1)->-1;(+1;-1)->-1
2: **REPETER**
3: **POUR** chaque exemple de la base:**(E1;E2)-> "S_correcte"**
4: Calcul de la sortie "S_calculée" pour chaque exemple:
4a: Fonction de combinaison:(sa sortie"p": potentiel)
     "p" = (W1 x E1) + (W2 x E2) - SEUIL
4b: Fonction d'activation:
     SI("p")>= 0 ALORS "S_calculée" = +1
     SINON "S_calculée" = -1
5: **SI** la sortie **"S_calculée" est différente** de la sortie
      "S_correcte" (**ERREUR = "S_correcte" - "S_calculée"**)
     ALORS Modification des poids:
5a: **W1(t+1)= W1(t) + (E1 x PAS x ERREUR)**
5b: **W2(t+1)= W2(t) + (E2 x PAS x ERREUR)**
6: **Revenir à 4 pour recalculer la sortie**
7: **TANTQUE** une **ERREUR** subsiste revenir à 4

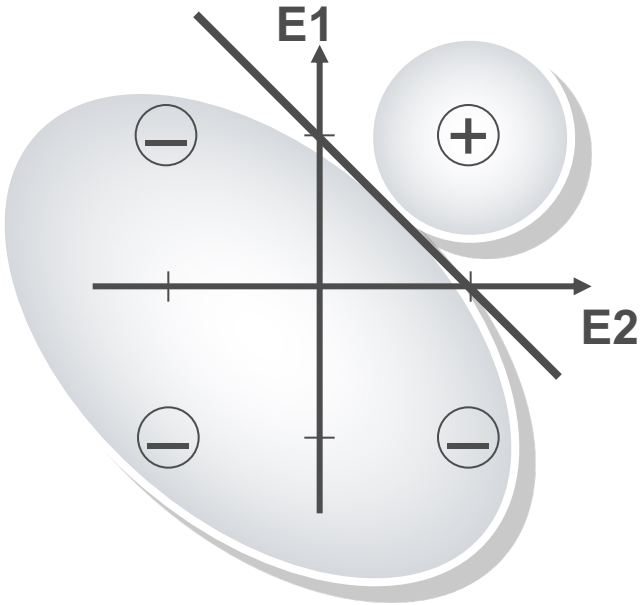
Apprentissage des fonctions logiques

Simulation de la fonction ET logique par apprentissage:

- Dans le cas de la fonction ET la séparation des deux classes se fait par une ligne droite:
  - W1xE1 + W2xE2 - SEUIL = 0

  Deux classes (deux réponses):
  - C1 (S = +1):
    - Pour les entrées: (+1;+1)
  - C2 (S = -1):
    - Pour les entrées:

    {(-1;-1); (+1;-1); (-1;+1)}



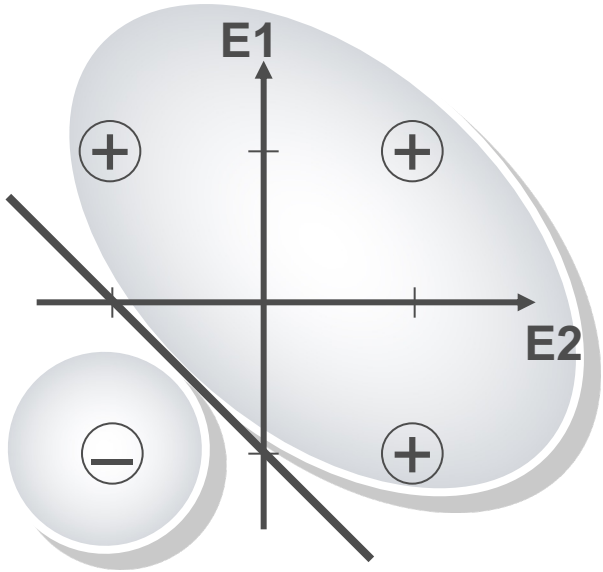
« ET » logique

Apprentissage des fonctions logiques

Simulation de la fonction OU logique par apprentissage:

- Dans le cas de la fonction OU, une droite permet toujours la séparation des deux classes.

- Pour deux classes :
  - C1 (S = +1):
    - {(+1;+1); (+1;-1);(-1;+1)}
  - C2 (S = -1):
    - (-1;-1)

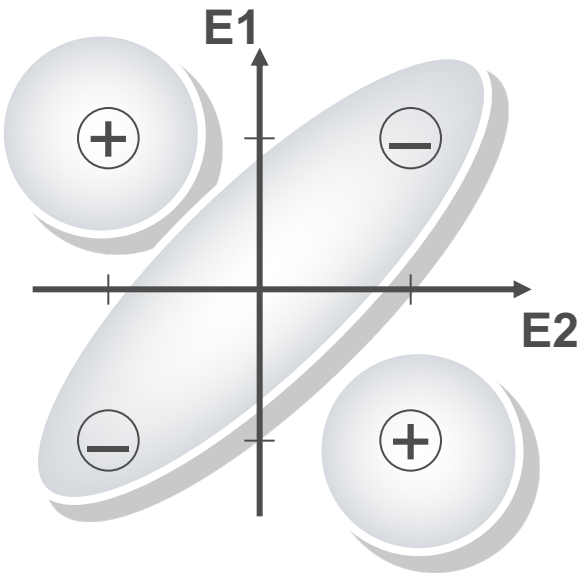


« OU » logique

Les limites du perceptron

Les limites du perceptron : la fonction logique OU exclusif.

- Dans le cas de la fonction **OU-EXCLUSIF**, la séparation des deux classes ne peut se faire par une droite mais par une courbe.
  - C1 (S = +1):
    - {(-1;+1); (+1;-1)}
  - C2 (S = -1):
    - {(+1;+1); (-1;-1)}



« OU-Exc » logique

Les limites du perceptron

- Le perceptron est un classificateur linéaire.
- Il ne peut traiter les problèmes non linéaires du type OU EXCLUSIF, COMPARATEUR (et bien d'autres…).
- La structuration d'un réseau neuronal (constitué de plusieurs couches), et l'utilisation conjointe d'un algorithme d'apprentissage approprié vont permettre de pallier les limites identifiées ici.

Pause-réflexion sur cette 2ème partie

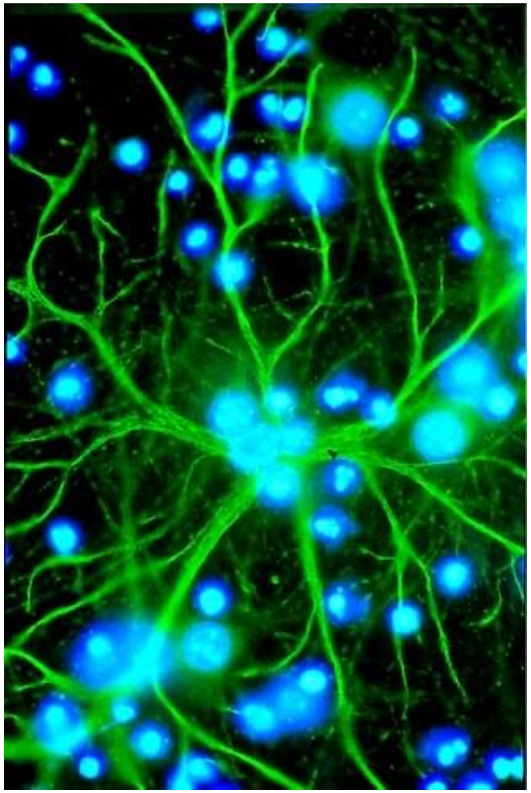
Avez-vous des questions ?

Réseaux multicouches à rétro-propagation de l'erreur

Plan de la partie

Voici les chapitres que nous allons aborder:

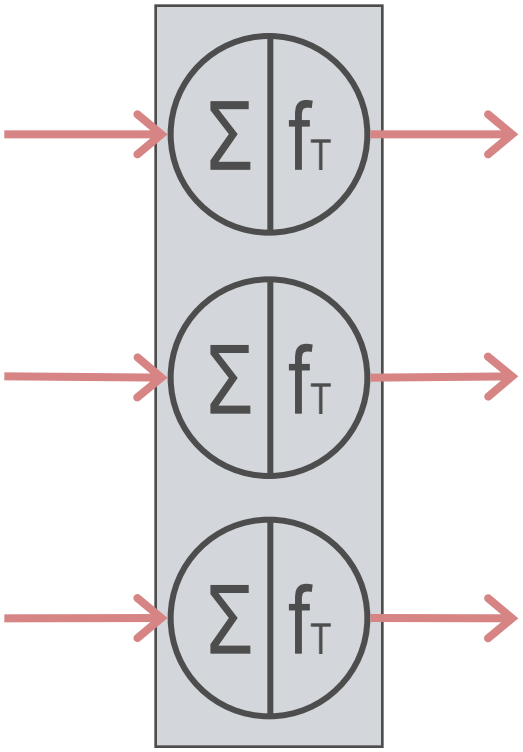
- Réseaux de neurones formels
- Apprentissage d'un réseau multicouche
- L'algorithme d'apprentissage



Réseaux de neurones formels

Réseau à une couche de neurones

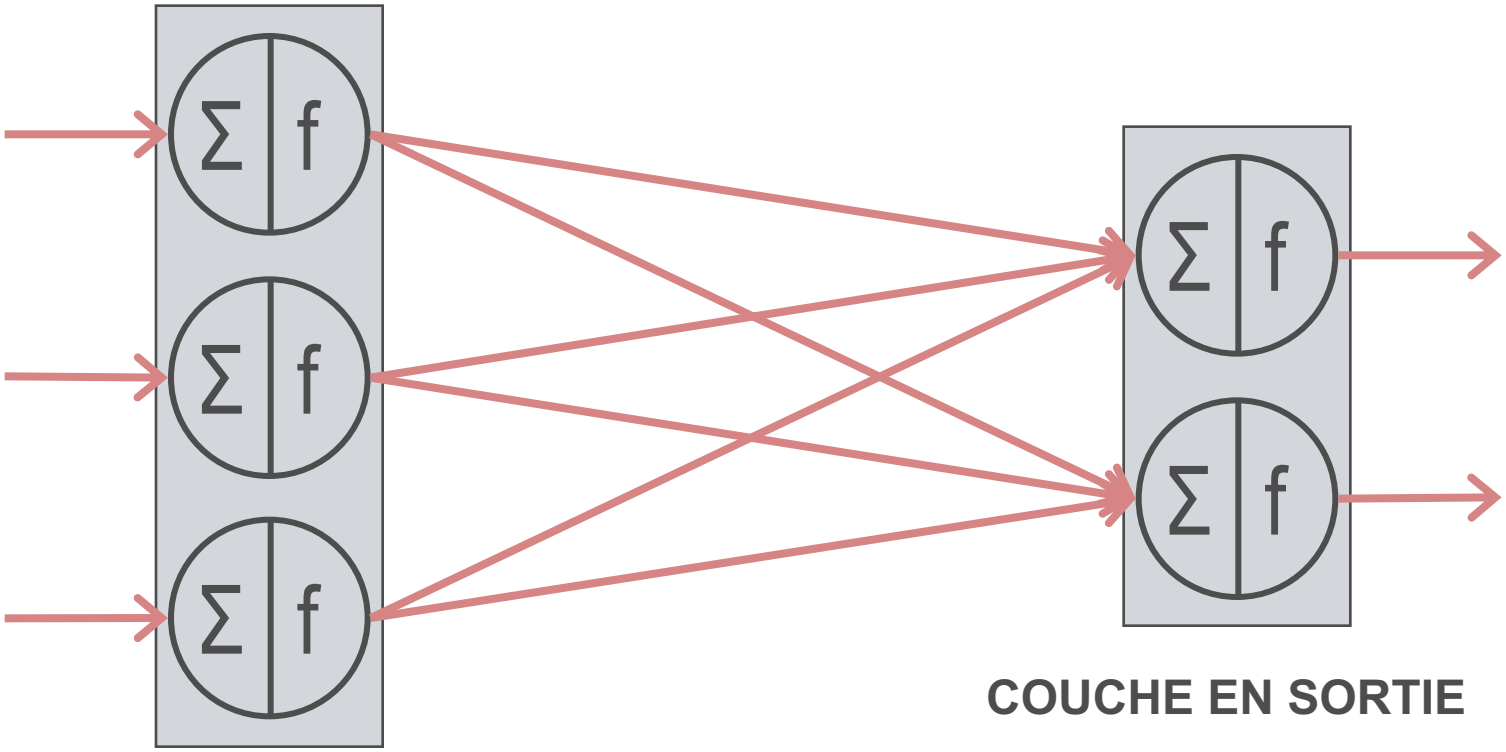
COUCHE UNIQUE*



*Les entrées des réseaux sont soit des sorties d'autres neurones, soit des entrées directes dans le réseau (par exemple des pixels).

Réseaux de neurones formels

Réseau à deux couches de neurones



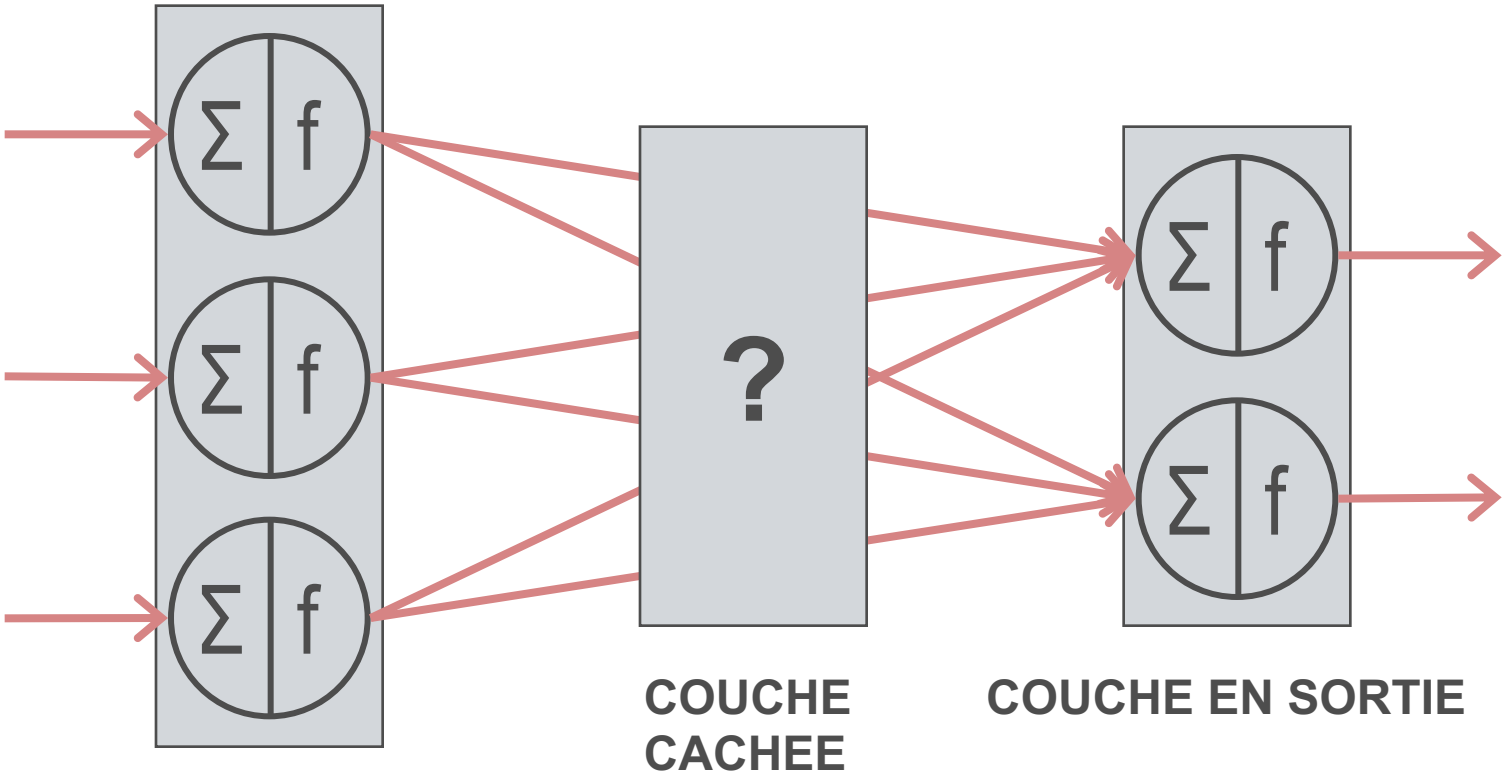
COUCHE EN SORTIE

COUCHE EN ENTREE*

*Les entrées des réseaux sont soit des sorties d'autres neurones, soit des entrées directes dans le réseau (par exemple des pixels).

Réseaux de neurones formels

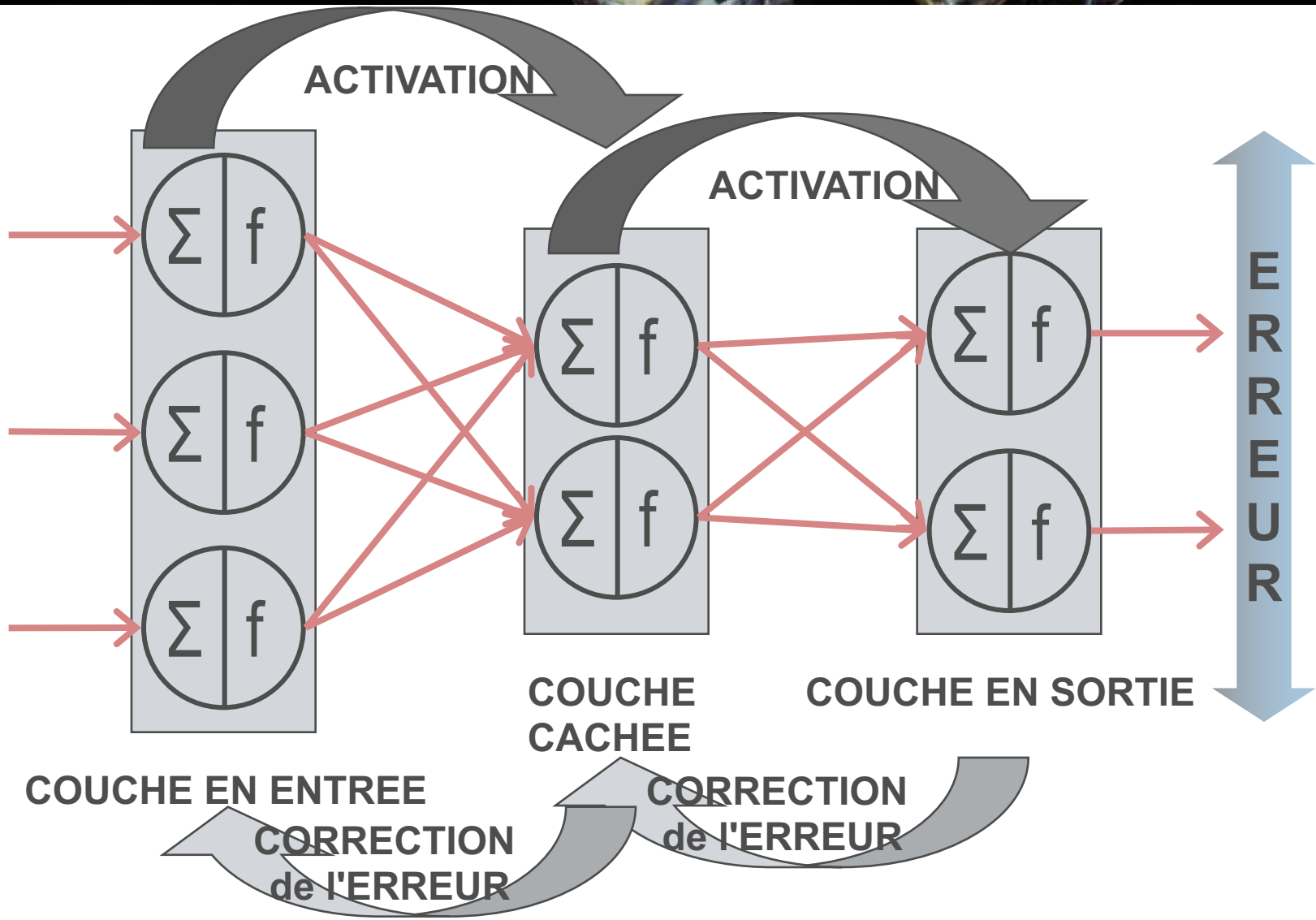
Réseau avec une couche cachée (3 couches de neurones).



*Les entrées des réseaux sont soit des sorties d'autres neurones, soit des entrées directes dans le réseau (par exemple des pixels).

Apprentissage d'un réseau multicouche

Principe de fonctionnement général.



Apprentissage d'un réseau multicouche

Notations :

- *x1, x2, x3, … , xk* : les formes présentées en entrée.
- *xk* : vecteur à I éléments.
- *X* : matrice I × K des K formes à apprendre.

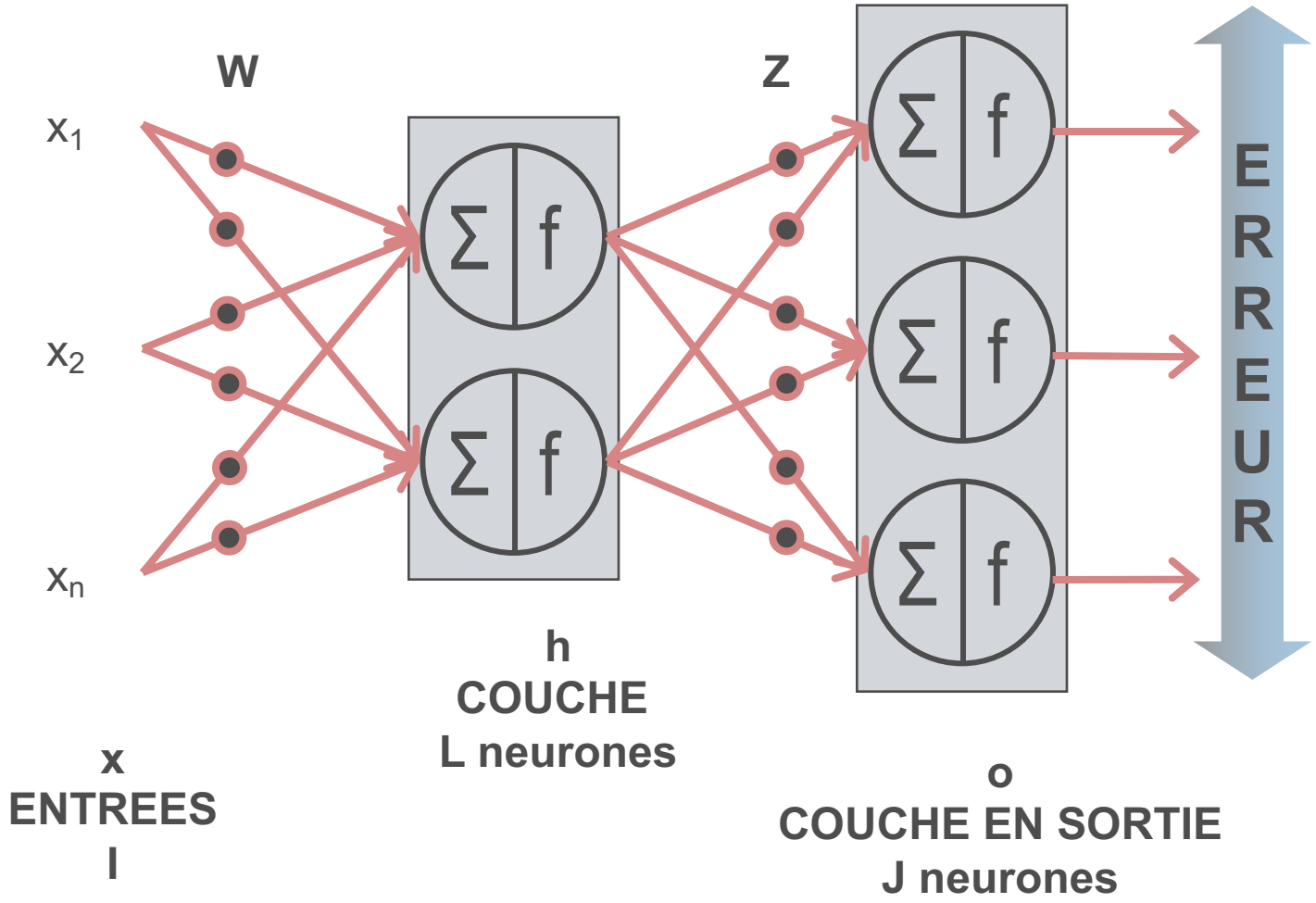
$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1I} \\ x_{21} & x_{22} & \cdots & x_{2I} \\ \vdots & \vdots & \ddots & \vdots \\ x_{K1} & x_{K2} & \cdots & x_{KI} \end{bmatrix}$$

Chaque couche fournit un « vecteur réponse »:

- *hk* : vecteur à L éléments, réponse de la couche cachée à la kième forme.
- *ok* : vecteur à J éléments, réponse de la couche de sortie à la kième forme.
- *tk* : vecteur à J éléments, réponse désirée (théorique) de la couche de sortie à la kième forme.
- *T* : matrice J × K des réponses désirées (théoriques).
- *W* : matrice L × I des poids de connexions entre les entrées et la couche cachée (W l,i : connexion entrée i - neurone l caché).
- *Z* : matrice J × L des poids de connexions entre la couche cachée et la couche de sortie (Z j,l : connexion neurone l caché - neurone j sortie).

Apprentissage d'un réseau multicouche

Les synapses modifiables (et leur matrice W et Z).



Apprentissage d'un réseau multicouche

Principe d'activation non linéaire.

- Soit un neurone *n* (d'une couche cachée ou de sortie) et son potentiel nommé *a* ; sa sortie *o* sera de la forme :

$$o_n = f(a_n)$$

- avec *f* : sa fonction de transfert (non linéaire, dérivable).

Exemples courants de fonction de transfert :

- La fonction logistique (ou sigmoïde) :

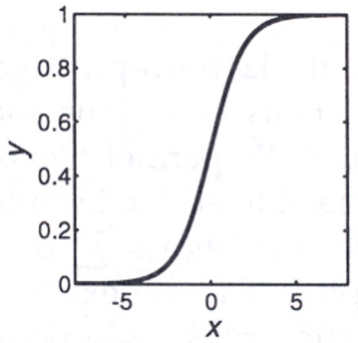
$$f(x) = \frac{1}{1 + e^{-x}}$$

Apprentissage d'un réseau multicouche

Représentation graphique des fonctions de transfert.

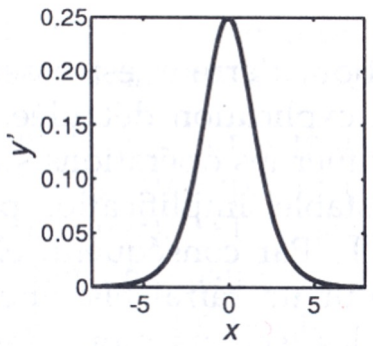
- La fonction logistique (ou sigmoïde) :

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



- et sa dérivée :

$$f'(x) = \frac{e^{-x}}{\left(1+e^{-x}\right)^2} = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} = f(x)\left[1-f(x)\right]$$



L'algorithme d'apprentissage

Algorithme de rétro-propagation de l'erreur,

Etape 1 : Transmission du signal entre l'entrée et la sortie via la couche cachée

- Soit le vecteur xk à l'entrée (forme k).
- La réponse de la cellule cachée est le vecteur :

$$h_k = f(Wx_k)$$

- La réponse des cellules de la couche de sortie est le vecteur :

$$o_k = f(Zh_k)$$

- Les matrices des poids, W et Z, déterminent le comportement du réseau.

L'algorithme d'apprentissage

Algorithme de rétro-propagation de l'erreur,

Etape 2 : Calcul de l'erreur en sortie

- On compare la réponse donnée (vecteur ok) à la réponse théorique (vecteur tk).
- Erreur pour la kième forme : $e_k = (t_k - o_k)$
- Le signal d'erreur résulte en pondérant l'erreur ek par l'état d'activation de chaque cellule (une activation forte est plus nocive qu'une activation faible).

$$\delta_{sortie,k} = f'(Zh_k) * e_k = o_k * (1 - o_k) * (t_k - o_k)$$

$*$: produit (de Hadamar) de deux matrices

$f'(Zh_k)$: intensité de l'activation des cellules de sortie

L'algorithme d'apprentissage

Algorithme de rétro-propagation de l'erreur,

Etape 3 : Correction des poids des connexions "cachée/sortie"

- La matrice des connexions Z est corrigée par des itérations successives.

$$Z_{t+1} = Z_t + \eta.\delta_{sortie,k}.h_k^T = Z_t + \Delta_t Z$$

$$\eta : \text{nombre réel positif (le pas d'apprentissage)}$$

L'algorithme d'apprentissage

Algorithme de rétro-propagation de l'erreur,

Etape 4 : Calcul de l'erreur en sortie des couches cachées

- Le problème est d'estimer l'erreur de l'activité (inconnue) des cellules cachées (pour une réponse attendue et connue en sortie uniquement !)
- Il n'y a pas de réponse idéale disponible.
- On l'estime à partir :
  - du signal d'erreur : $\delta_{sortie,k}$
  - de l'activation des cellules cachées.

L'algorithme d'apprentissage

Algorithme de rétro-propagation de l'erreur,

Etape 4 (suite) : Calcul de l'erreur en sortie des couches cachées

- L'erreur $\delta_{sortie,k}$ se propage en sens inverse (back-propagation) à travers les connexions Z.
- Elle est pondérée par l'activation $f'(Wx_k)$ des cellules cachées.
- Le signal d'erreur :

$$\delta_{cachée,k} = f'(Wx_k) * \left(Z_t^T \delta_{sortie,k}\right) = h_k * (1 - h_k) * \left(Z_t^T \delta_{sortie,k}\right)$$

L'algorithme d'apprentissage

Algorithme de rétro-propagation de l'erreur,

Etape 5 : Correction des poids des connexions "entrée/cachée"

- Calcul des poids des neurones d'entrée :

$$W_{t+1} = W_t + \eta.\delta_{cachée,k}.x_k^T = W_t + \Delta_t W$$

Pause-réflexion sur cette 3ème partie

Avez-vous des questions ?

Résumé du module

Perceptron : principe de séparabilité linéaire

NEURONE = Combinaison + Transfert

Rétro-propagation de l'erreur : estimation de l'erreur en couches cachées

Fonction Combinaison :

$$\sum Wi\ Ei$$

Fonction Transfert : en échelon, ou linéaire par morceaux, ou dérivable