HMIN233 - Algorithmes d'exploration et de mouvement

Neurocontrôleurs

Suro François

Université de Montpellier
Laboratoire d'informatique, de robotique
et de microélectronique de Montpellier

Janvier 2021

Comportement réactif

Un comportement réactif produit toujours les mêmes actions dans des situations identiques.

Comme il n'y a pas de planification ou de système de mémoire, l'agent réagit uniquement aux perceptions.

On peut comprendre un comportement réactif comme une fonction qui associe les perceptions aux actions.

$$f(\mathit{perception}) = \mathit{action}$$

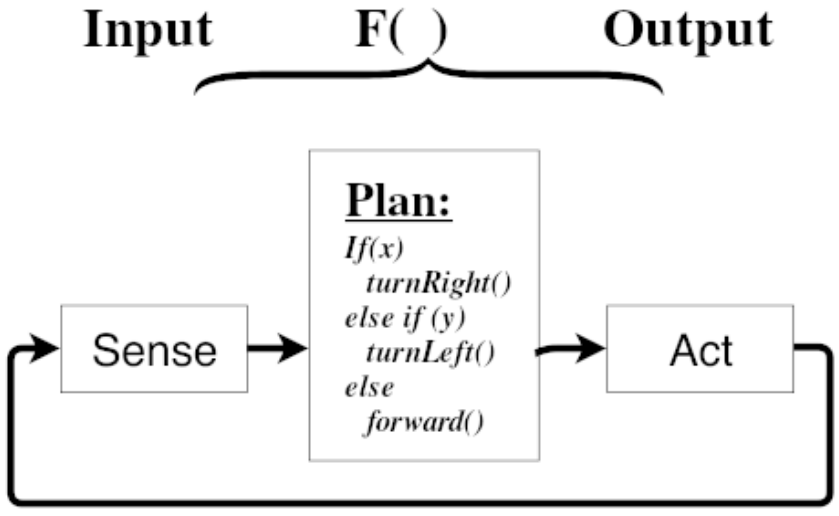
Comment contrôler un agent réactif ?

à un niveau de contrôle fin, de type sensorimoteur, nous avons vu 2 solutions :

- L'exécution de plans sous conditions: techniques "naïves", par exécution de sous-fonctions, chaque plan est un élément simple indivisible (tourner à droite, reculer ...)
- Les techniques vectorielles: les différentes informations perceptibles permettent de calculer une action.

Comment contrôler un agent réactif ?

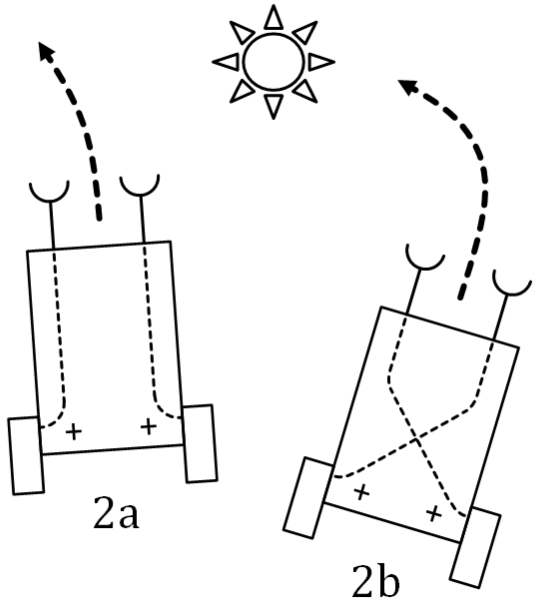
Par programmation,
sous-fonctions,
actions atomiques
(indivisibles)

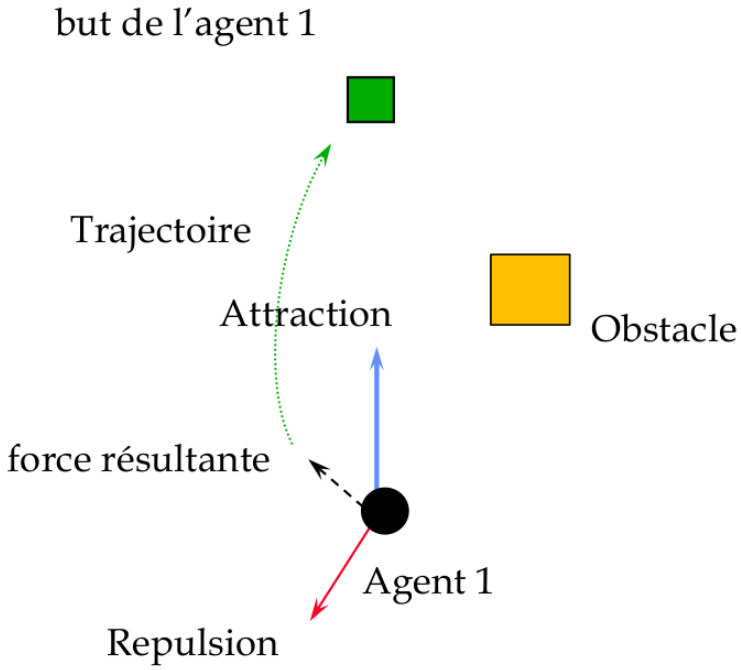



```
1  A <- agents a portee
2  a <- agent a portee le plus proche
3
4  Si Distance(a) < distance evitement
5     eviter()
6  Sinon Si DistanceCentre(A) > distance cohesion
7     cohesion()
8  Sinon
9     aligner()
```

Comment contrôler un agent réactif ?

Par vecteurs, par signal.





Vehicles: Experiments in Synthetic Psychology [Braitenberg, 1984]

Apprentissage

Au lieu de créer une fonction différente pour chaque comportement.

$$f_{chasseur}(perception) = action$$
$$f_{artisan}(perception) = action$$

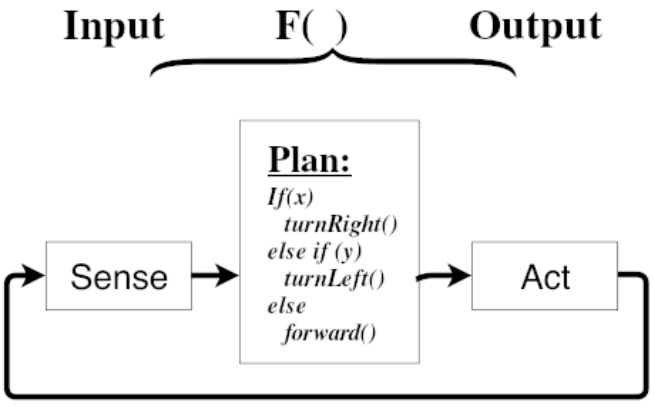
On souhaite avoir une fonction paramétrable, dont on peut changer le comportement.

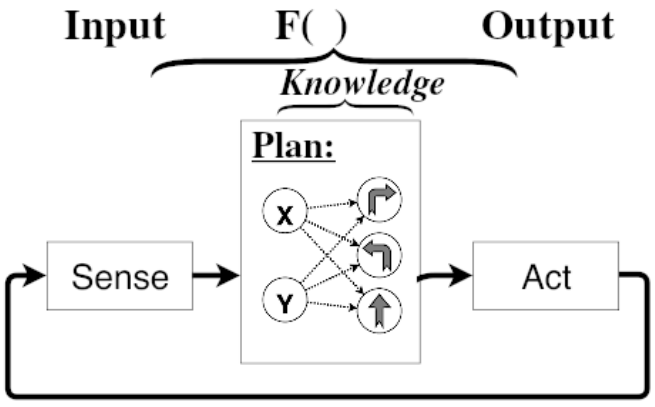
$$f_{humain}(perception, chasseur) = action$$
$$f_{humain}(perception, artisan) = action$$

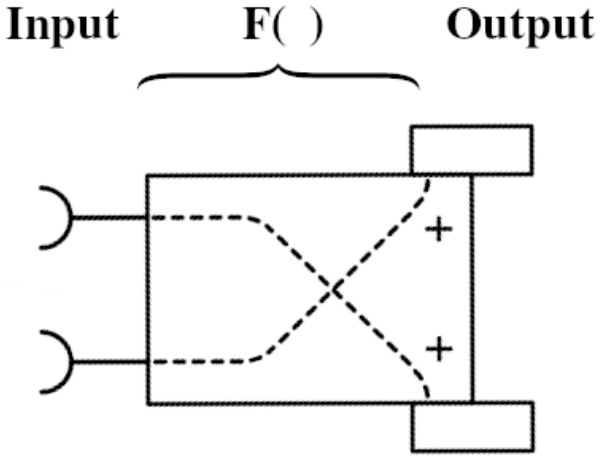
$$f_{agent}(perception, connaissance) = action$$

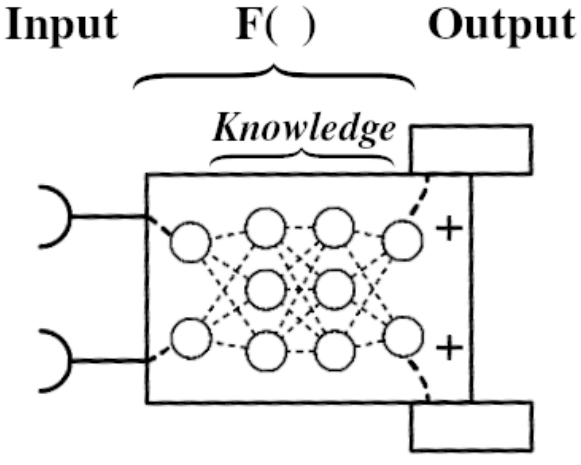
Un algorithme d'apprentissage va chercher les bonnes valeurs de ce paramètre de connaissance (savoir-faire) afin d'obtenir le comportement souhaité.

Apprentissage sensorimoteur





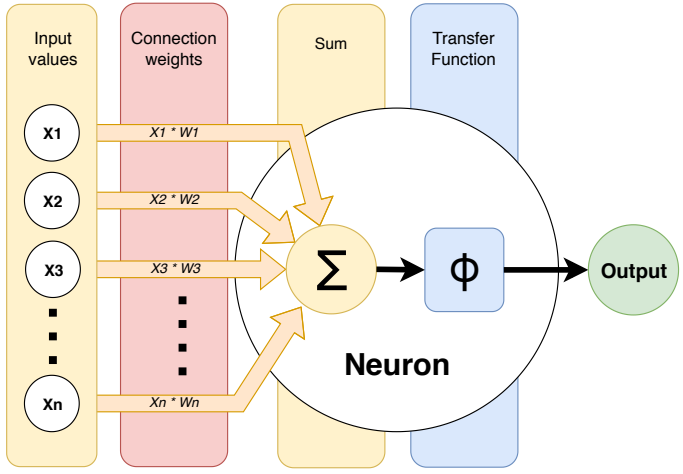




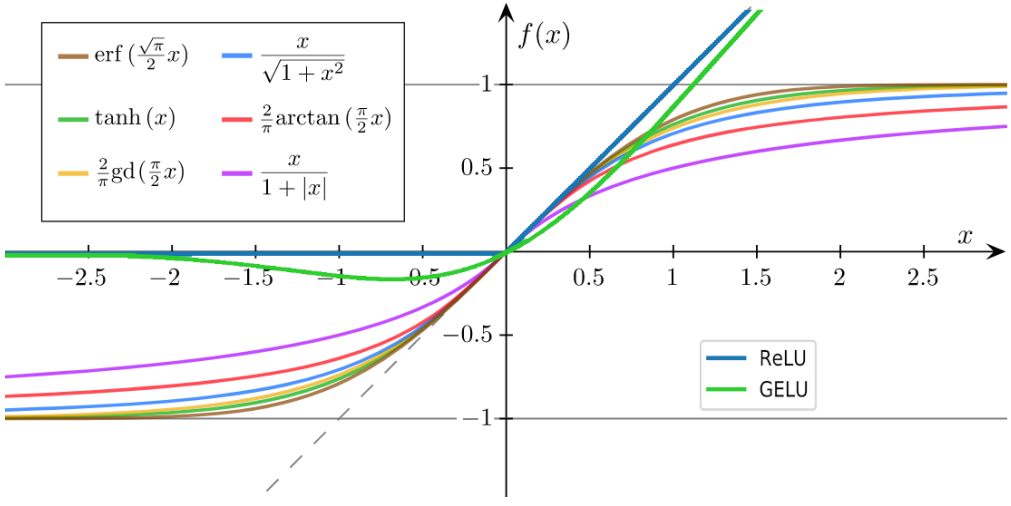
Approche neuronale: le perceptron

Perceptron

- Les valeurs d'entrée sont multipliés par le poids de leurs connexions (des "synapses").
- Le résultat est sommé dans le neurone, une fonction de transfert est appliqué (sigmoïde, Relu ...).

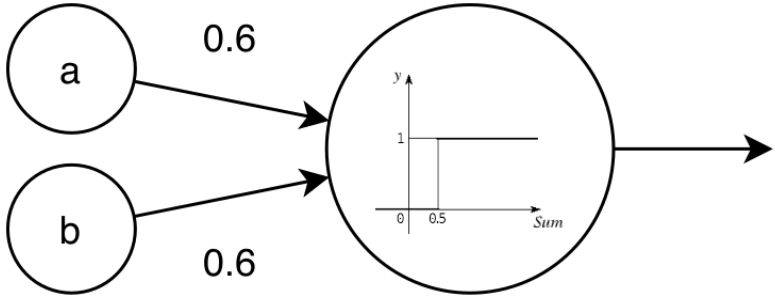


Fonction de transfert



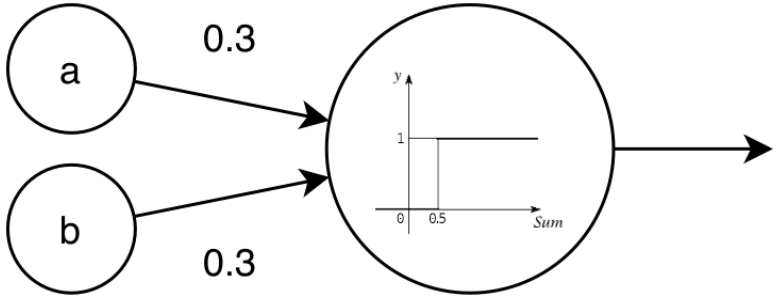
Perceptron : exemple

Fonction OU



a	b	Sum	Result
0	0	0	0
0	1	0.6	1
1	0	0.6	1
1	1	1.2	1

Fonction ET



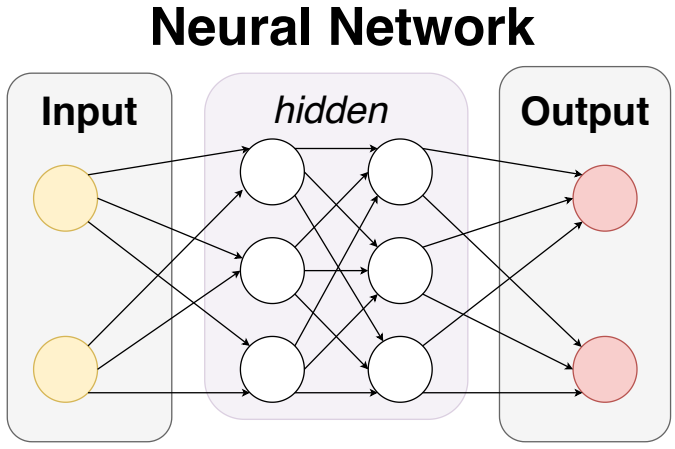
a	b	Sum	Result
0	0	0	0
0	1	0.3	0
1	0	0.3	0
1	1	0.6	1

La fonction représentée dépends donc de la configuration des **poids** du perceptron.

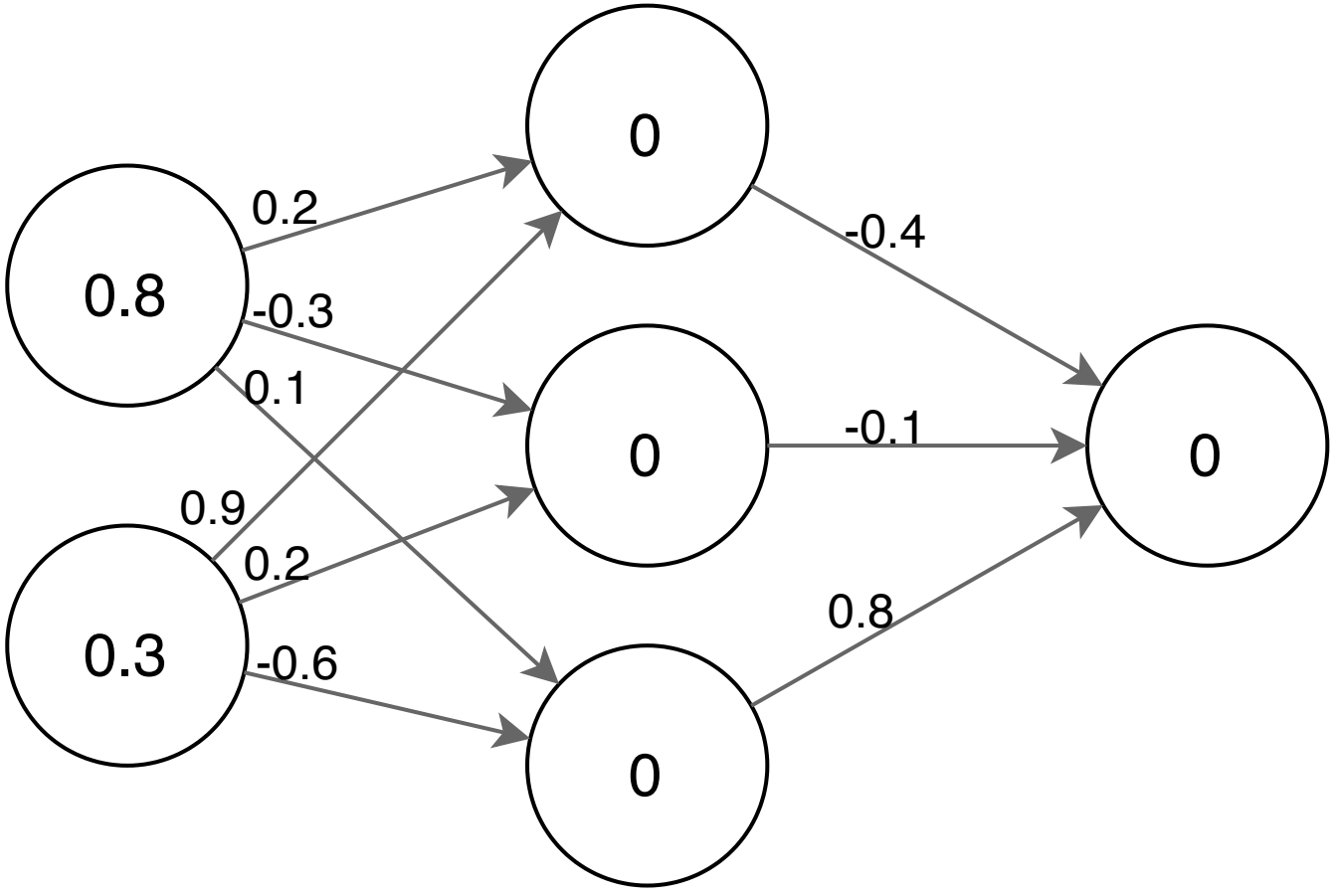
Réseaux neuronaux

Réseau de neurones: perceptron multi couches

- Plusieurs couches connectées en succession.
- Couche : ensemble de neurones non connectés entre eux
- Signal propagé de l'entrée vers la sortie

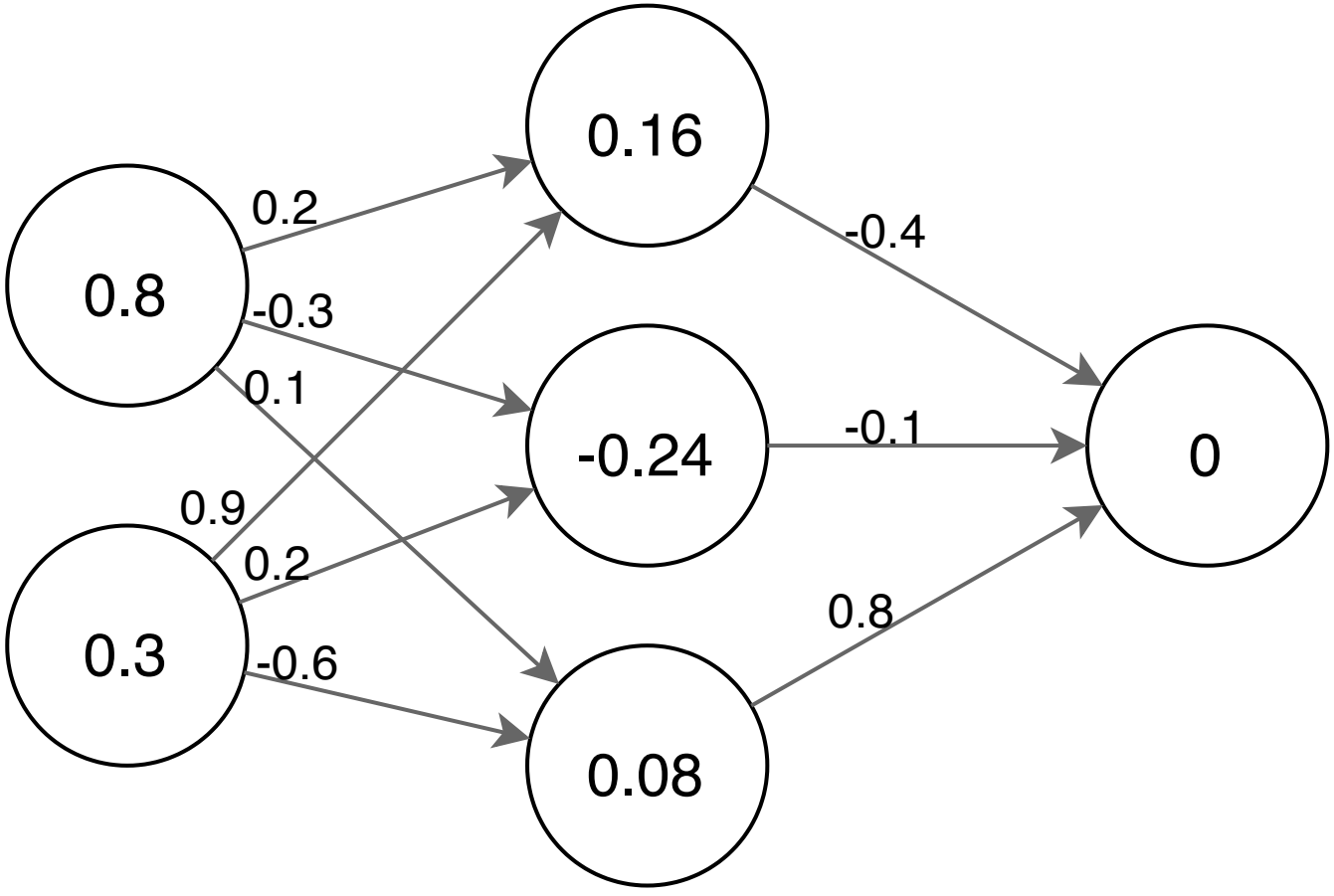


Fonctionnement



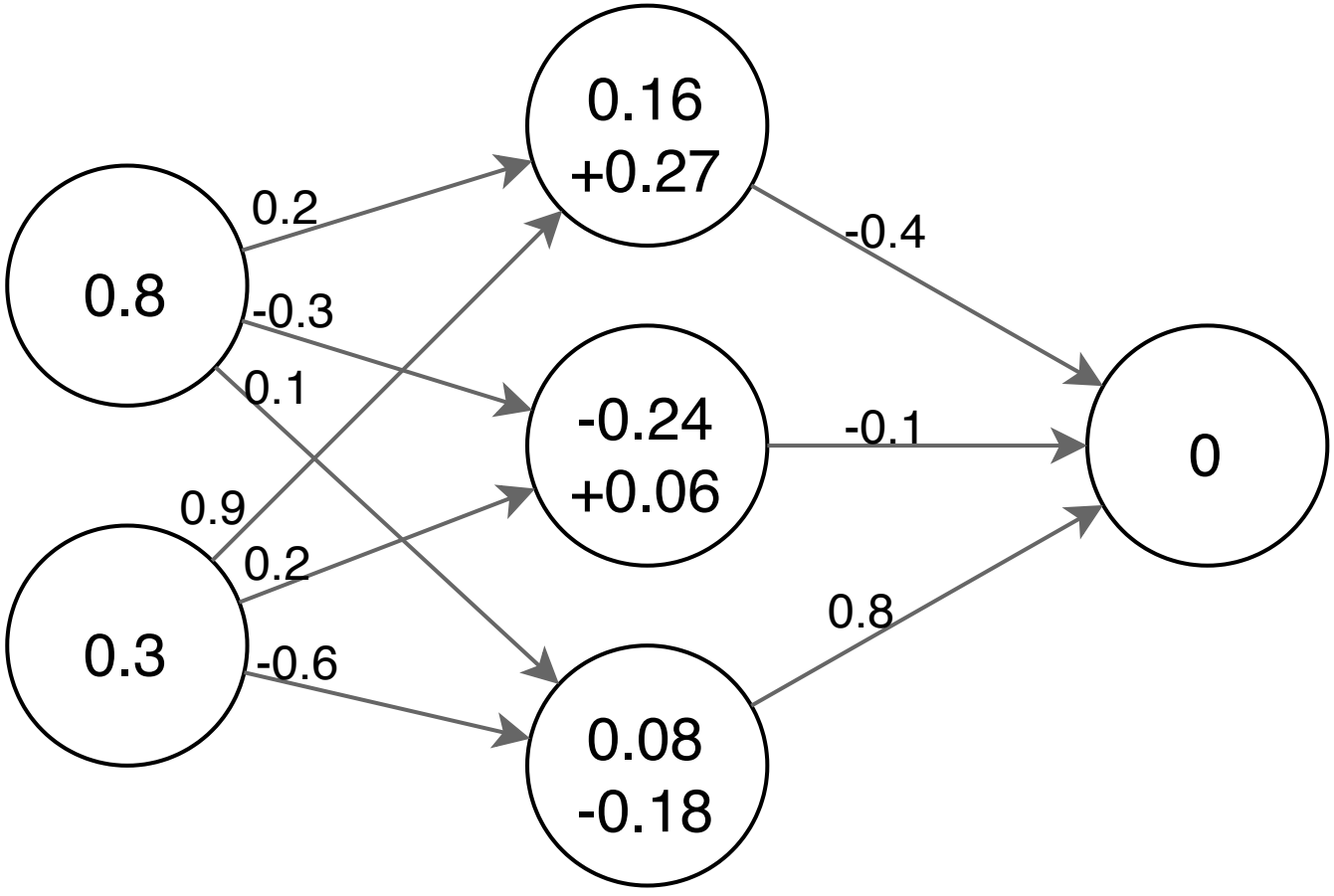
Weights : 0.2 -0.3 0.1 0.9 0.2 -0.6 -0.4 -0.1 0.8

Fonctionnement



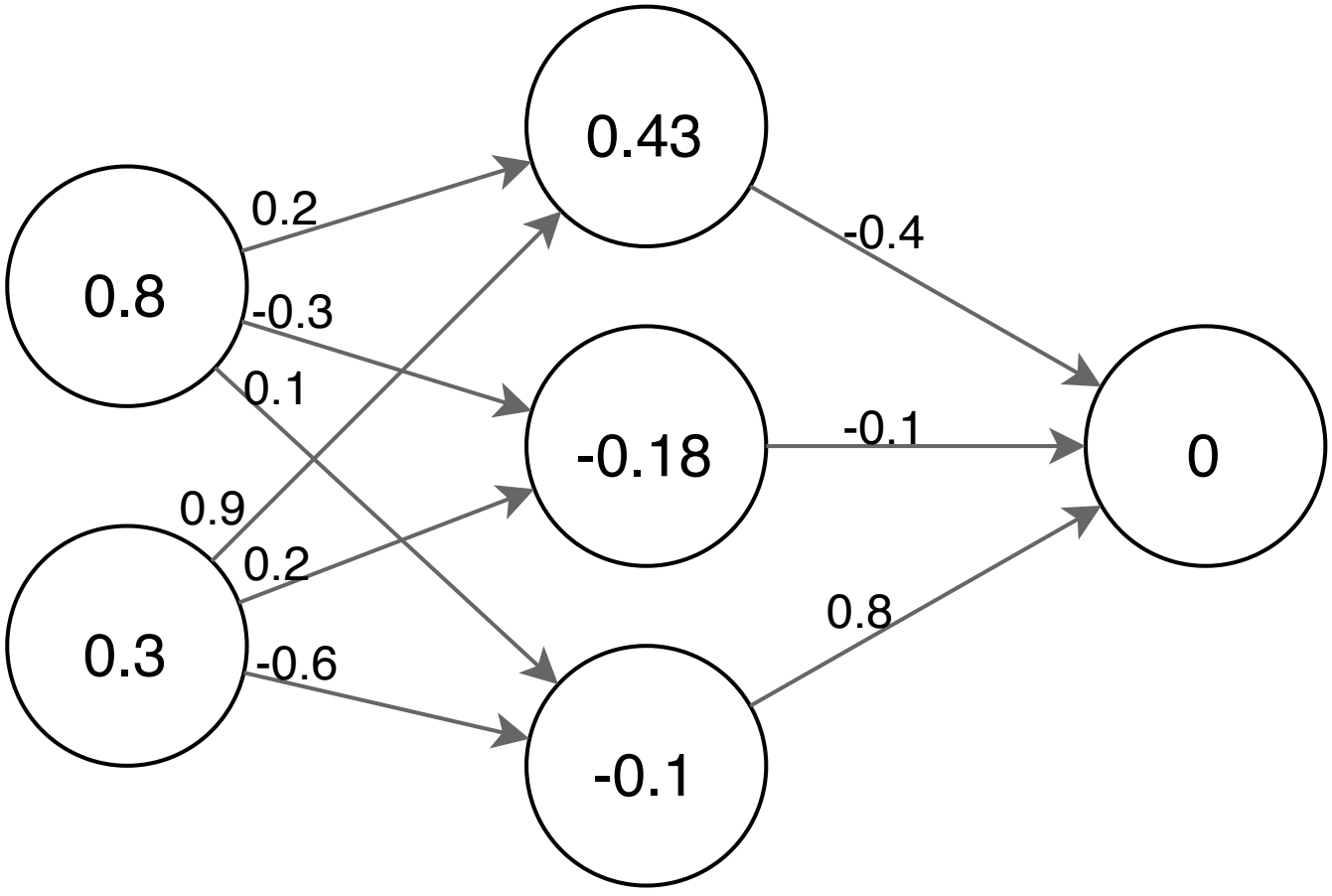
Weights : 0.2 -0.3 0.1 0.9 0.2 -0.6 -0.4 -0.1 0.8

Fonctionnement



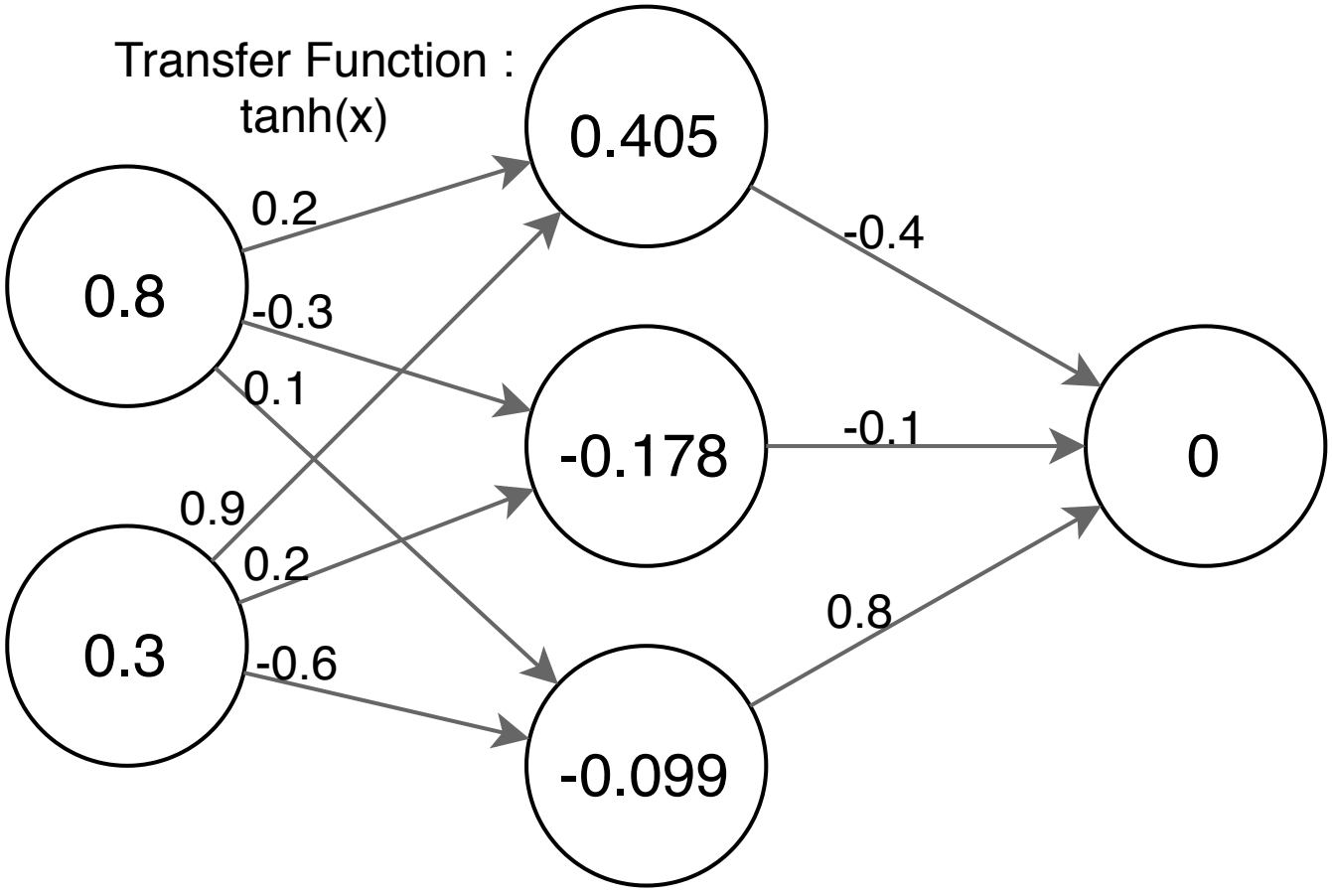
Weights : 0.2 -0.3 0.1 0.9 0.2 -0.6 -0.4 -0.1 0.8

Fonctionnement



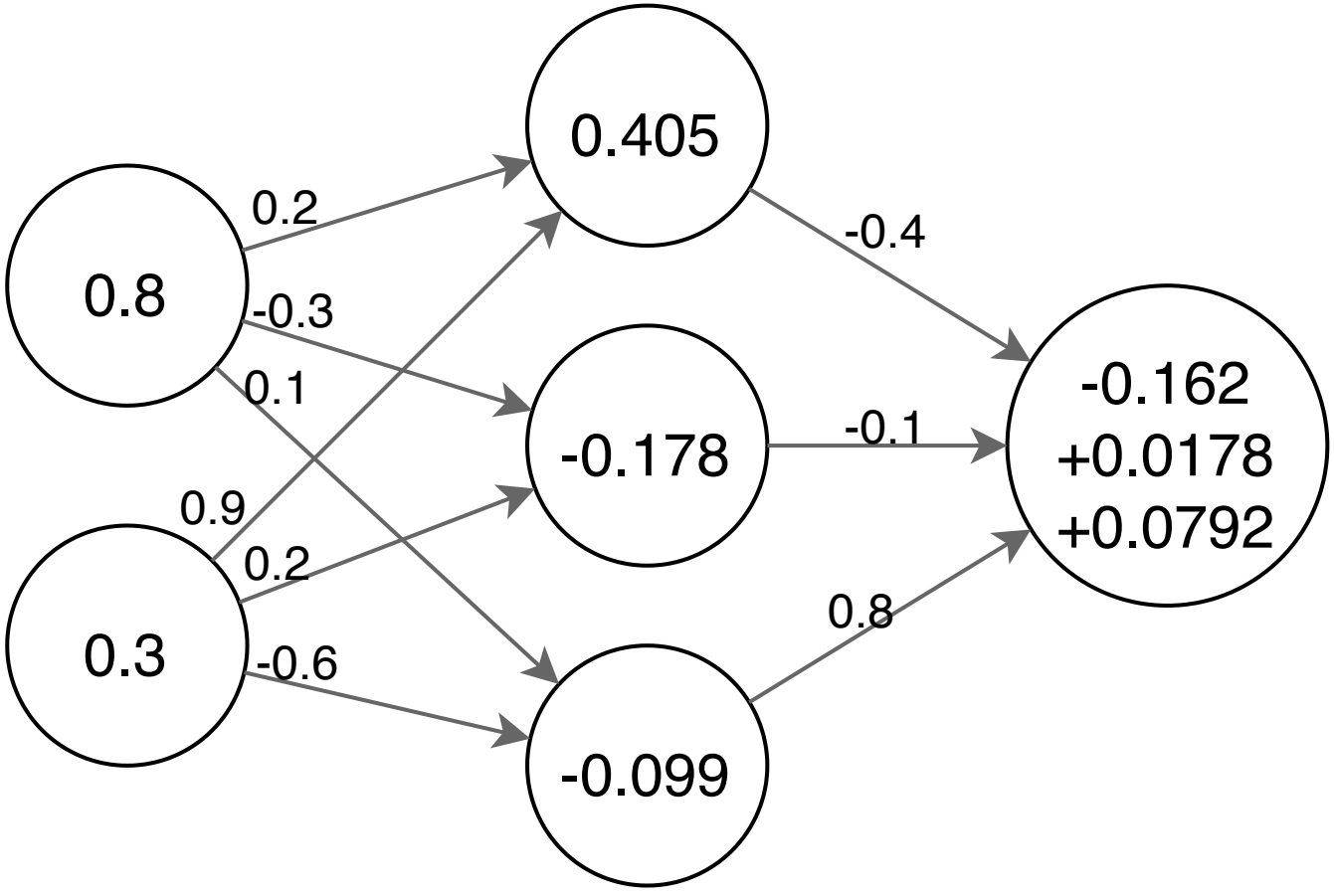
Weights : 0.2 -0.3 0.1 0.9 0.2 -0.6 -0.4 -0.1 0.8

Fonctionnement



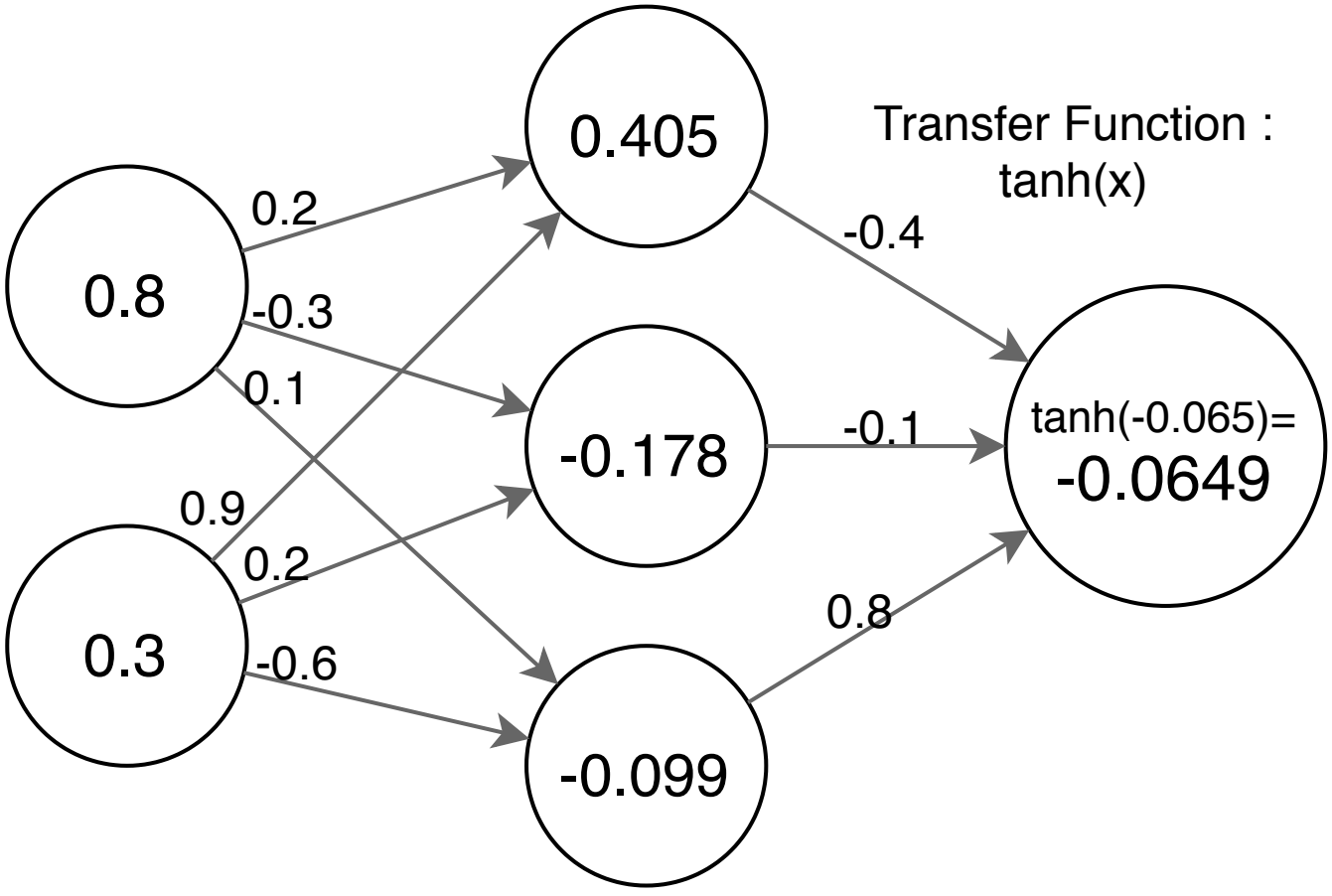
Weights : 0.2 -0.3 0.1 0.9 0.2 -0.6 -0.4 -0.1 0.8

Fonctionnement



Weights : 0.2 -0.3 0.1 0.9 0.2 -0.6 -0.4 -0.1 0.8

Fonctionnement

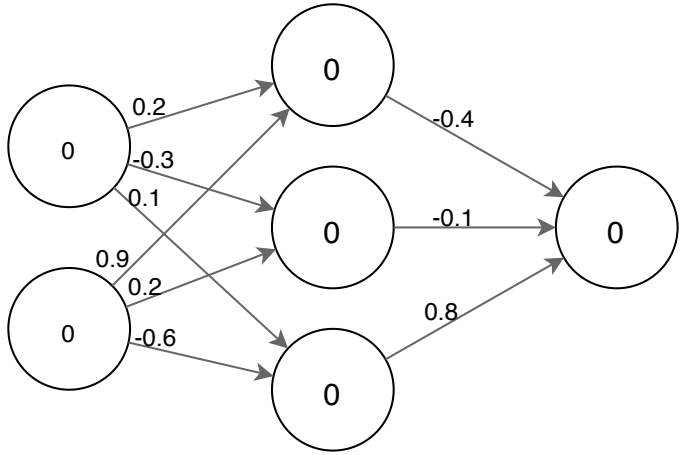


Weights : 0.2 -0.3 0.1 0.9 0.2 -0.6 -0.4 -0.1 0.8

Biais

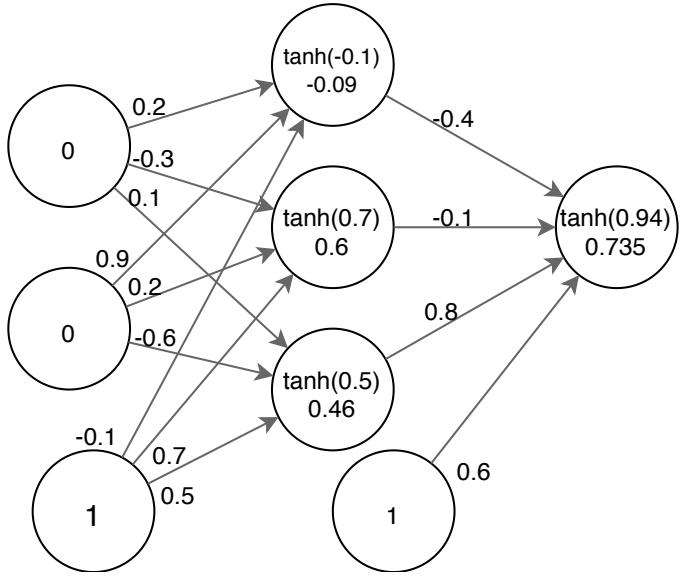
Bias neuron

- On rajoute à chaque couche un neurone qui ne reçoit pas d'entrées et dont la valeur est toujours 1
- Sa sortie est modulée par des poids comme tout autre neurone.



Intérêt

- Sans biais, pour une entrée égale à zéro, toute configuration du réseau ne peut donner que zéro comme résultat.
- On peut voir le biais comme la constante b dans l'expression $y = ax + b$



Complexité de la configuration

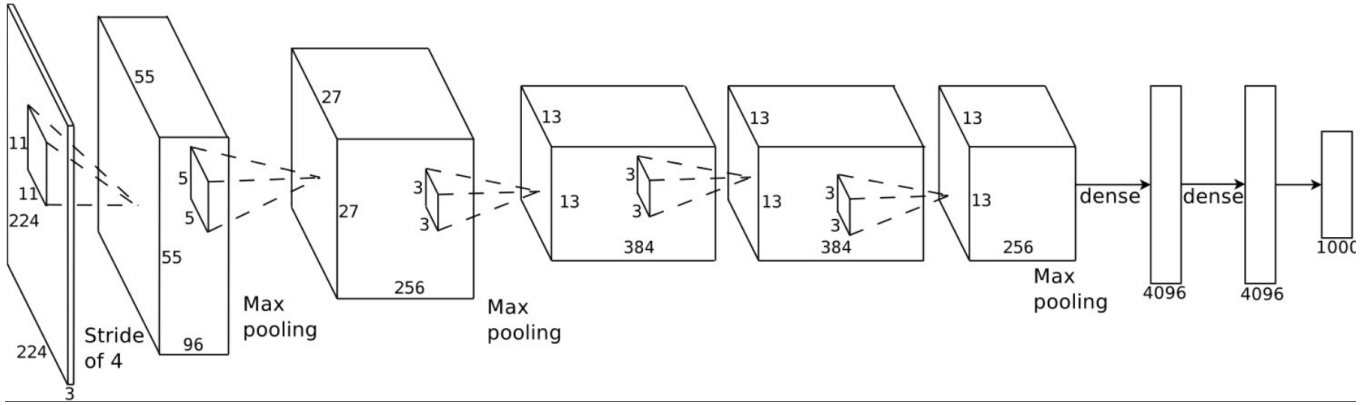
Comme pour le perceptron, la fonction représentée par le réseau dépends de la configuration des **poids** (et de la topologie).
Dans le cas d'un perceptron multi couches le nombre de poids à configurer :

$$\sum_{c=1}^{m-1} N_c * N_{c+1}$$

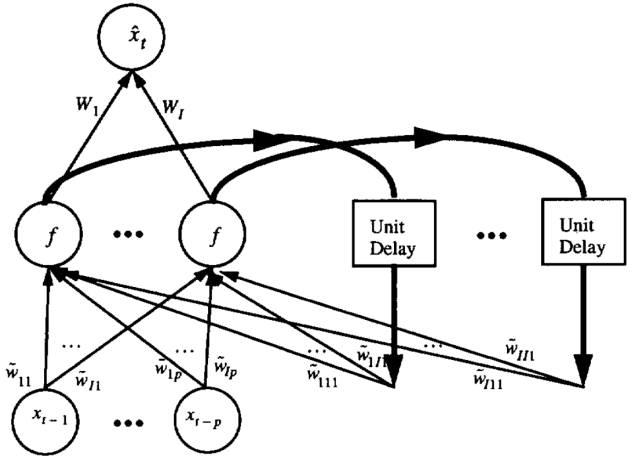
Dans le cas ou le nombre de neurones dans les couches cachées est constant :

$$N_{entre} * N_{cache} + (M_{cache} - 1) * N_{cache}^2 + N_{cache} * N_{sortie}$$

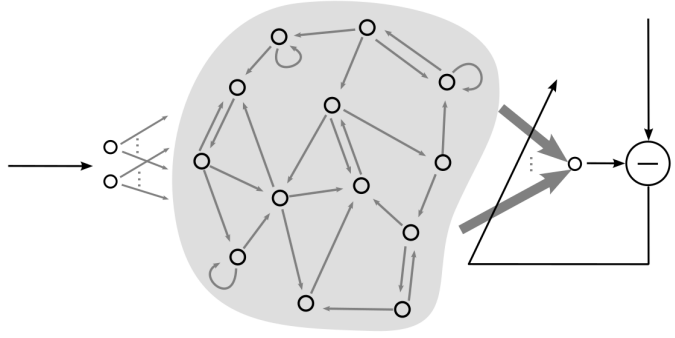
Réseaux neuronaux: topologies



Réseau neuronal convolutionnel



Réseau neuronal récurrent



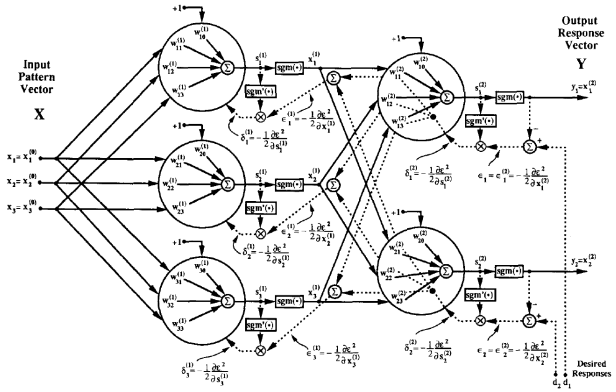
Reservoir computing
(Liquid state machines)

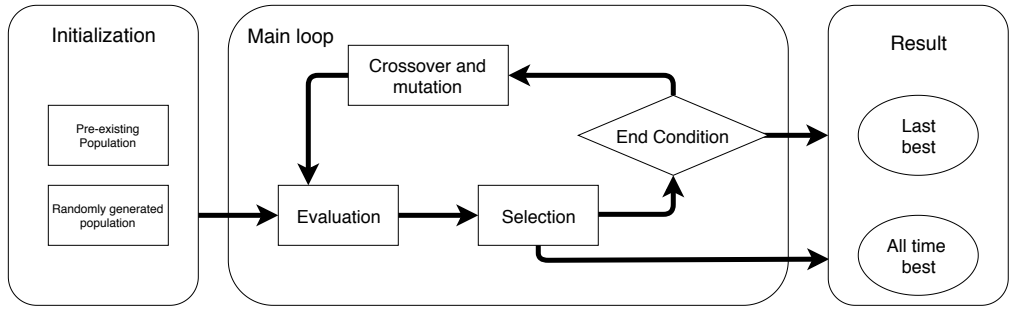
Réseaux neuronaux: apprentissage

Le comportement est décrit par les poids des connexions

Impossible à la main, besoin d'apprentissage :

- Backpropagation: corriger progressivement l'erreur sur un ensemble d'exemples.
- Algorithmes génétique: tester plusieurs configurations, sélectionner les meilleures pour en générer de nouvelles.

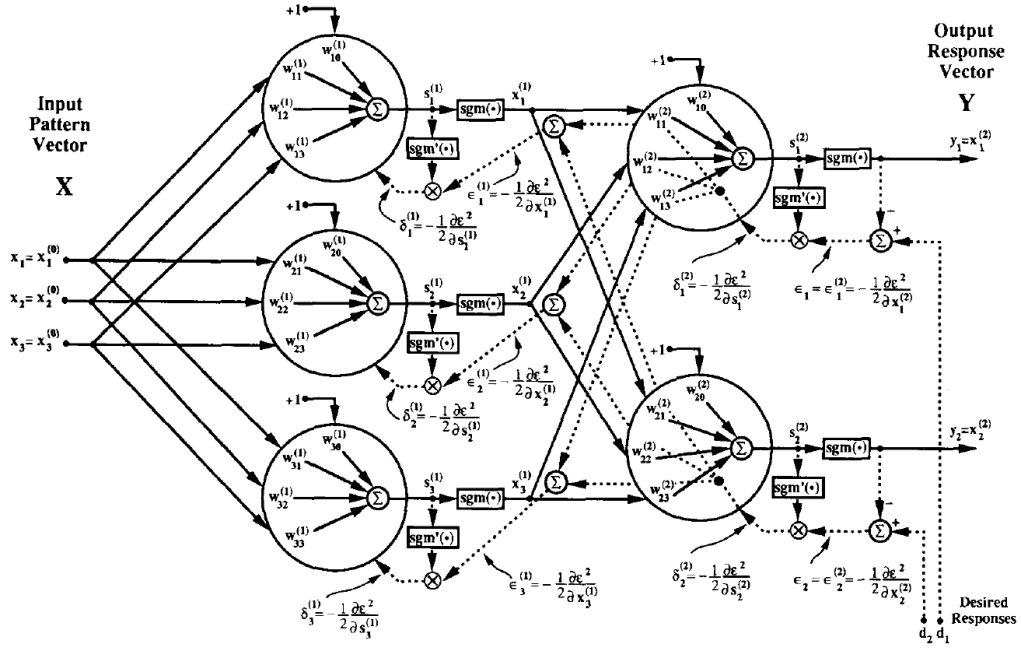




Backpropagation

Rétro-propagation de l'erreur

On corrige progressivement l'erreur sur des exemples connus, de manière à interpoler d'autres valeurs d'entrée.



Training set

Un ensemble de couples d'entrées-sorties correctes.

Validation set

Sous partie du training set qui n'est pas utilisé pour l'apprentissage. Sert à vérifier la qualité de l'apprentissage.

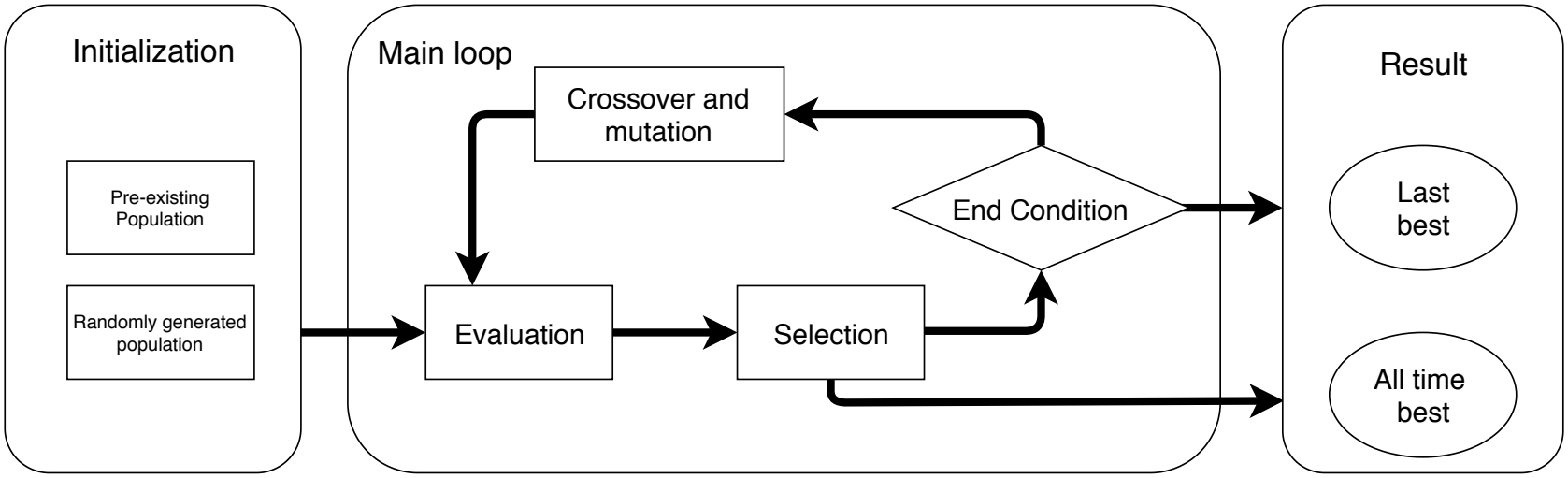
Backpropagation

1. Calculer le résultat pour l'entrée fournie (O), comparer au résultat attendu (T). Calculer l'erreur ainsi : $E_n = \frac{1}{2}(O - T)^2$
2. Calculer l'erreur à propager en multipliant par la dérivé de la fonction de transfert ($f'(\sigma_n)$): $\delta_n = E_n * f'(\sigma_n)$
3. Calculer l'erreur de la couche précédente (E_{n-1}). Pour chaque neurone, somme des erreurs à propager (δ_n) multiplié par le poids du lien(W_n): $E_{n-1} = \sum \delta_n * W_n$
4. Calculer l'erreur à propager (voir étape 2) pour cette couche : $\delta_{n-1} = E_{n-1} * f'(\sigma_{n-1})$

   $\rightarrow$ Répéter 3 et 4 pour chaque couche.
5. Mettre à jour les poids du réseau. Soustraire au poids la valeur de sortie du neurone entrant (O_n) multiplié par l'erreur propagée du neurone sortant (δ_{n+1}) et le taux d'apprentissage (μ).

   $W_{n \mapsto n+1} = W_{n \mapsto n+1} - (\mu * O_n * \delta_{n+1})$

Algorithmes génétique

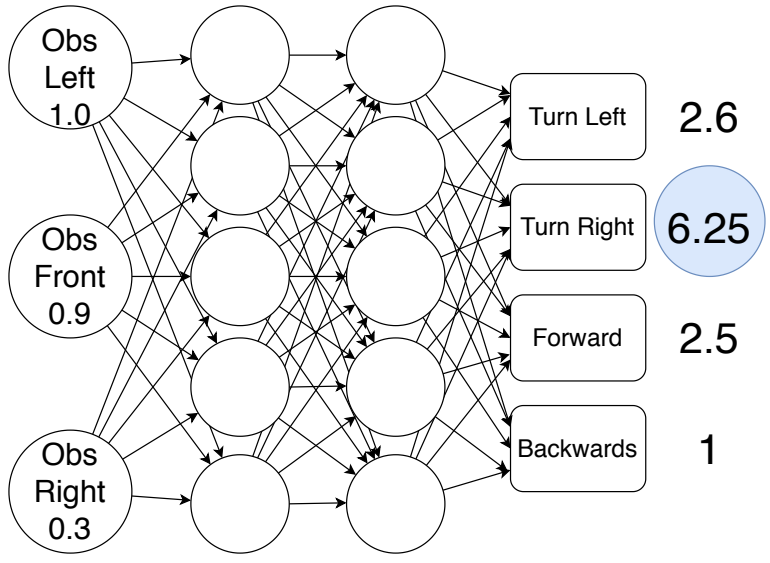


- Le génome correspond à la liste des poids du réseau.
- Peu de contraintes sur les opérations de croisement et de mutation.
- Évaluation ?

Neurocontrôleur

Classification

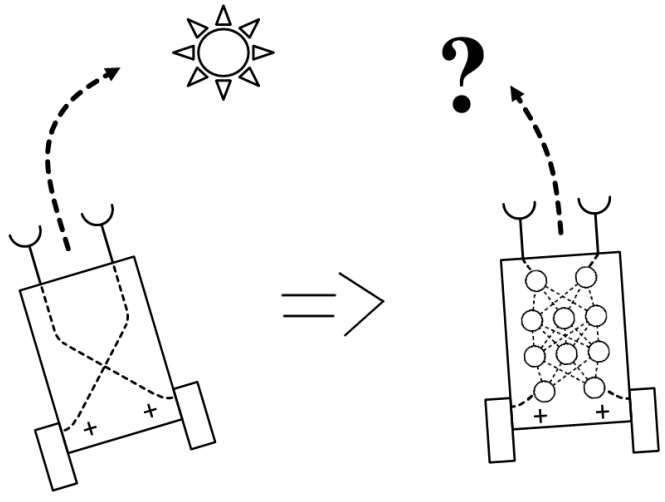
- Les percepts sont fournis en entrée du réseau qui calcules un "score" pour chaque action possible (comme la "probabilité" d'un classificateur).
- L'action avec le score le plus haut est choisie.



Neurocontrôleur

Signal, intensité

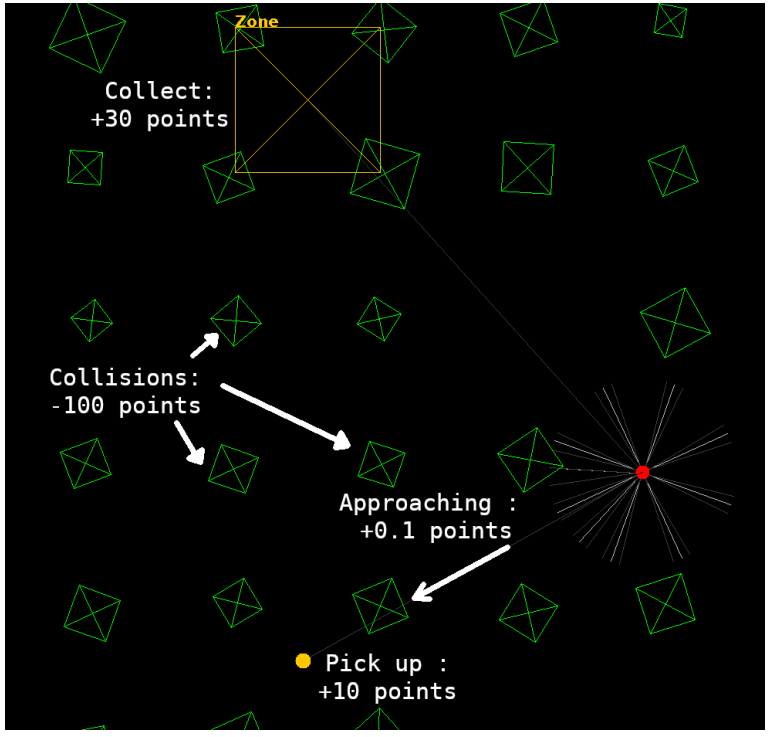
- Le signal des capteurs est modifié par le réseau dont la sortie est fournie comme signal aux actionneurs
- Replace le câblage d'un véhicule de Braitenberg



Apprentissage génétique et neurocontrôleurs

Évaluation

- On laisse notre agent agir dans un environnement pendant un certain temps.
- Par observation (automatisée ...) on récompense certaines actions.

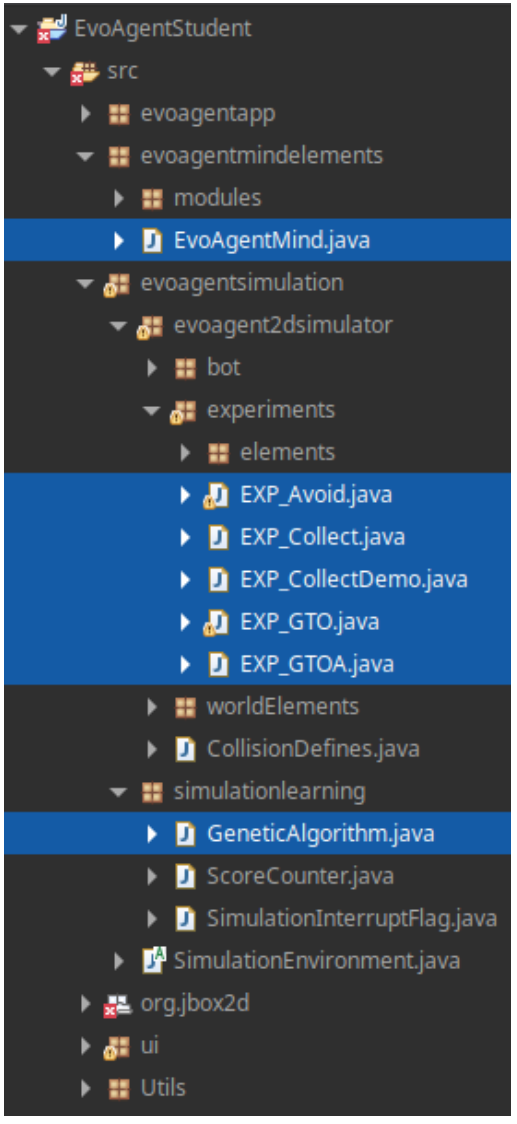


- Trouver les bonnes fonctions de récompense.
- Trouver le bon compromis sur le temps de l'évaluation.
- Essayer d'évaluer les génomes de manière équitable.

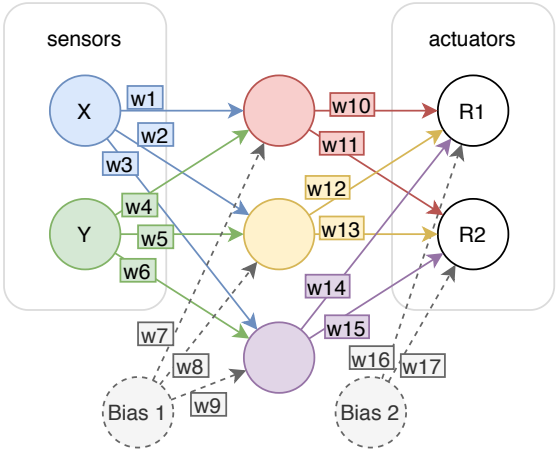
TP : EvoAgentStudents

Projet Eclipse

- EvoAgentMind : "L'esprit" de l'agent, dans lequel vous écrirez votre structure neuronale.
- GeneticAlgorithm : Vous porterez votre algorithme génétique de la séance précédente sur ce problème.
- EXP : Les classes expériences. Vous mettrez en place les environnements et fonctions de récompense afin d'entraîner votre robot à accomplir des tâches.



TP : EvoAgentStudents



EvoAgentMind:

sensorModules = { X , Y }
actuatorModules = { R1 , R2 }
hiddenLayerCount = 1
hiddenLayerSize = 3

genomeSize() => 17

weights[] =

w1	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16	w17

Classe EvoAgentMind

- sensorModules:Liste des capteurs de l'agent.
- actuatorModules:Liste des actionneurs de l'agent.
- hiddenLayerCount:le nombre de couches cachées.
- hiddenLayerSize:le nombre de neurones dans chaque couche cachée, sans compter le neurone de biais si vous souhaiter l'implémenter.
- weights[]:un tableau de doubles représentant les poids du réseau. Sa taille sera définie par la fonction genomeSize(), en fonction de tous les paramètres précédents.

TP : EvoAgentStudents

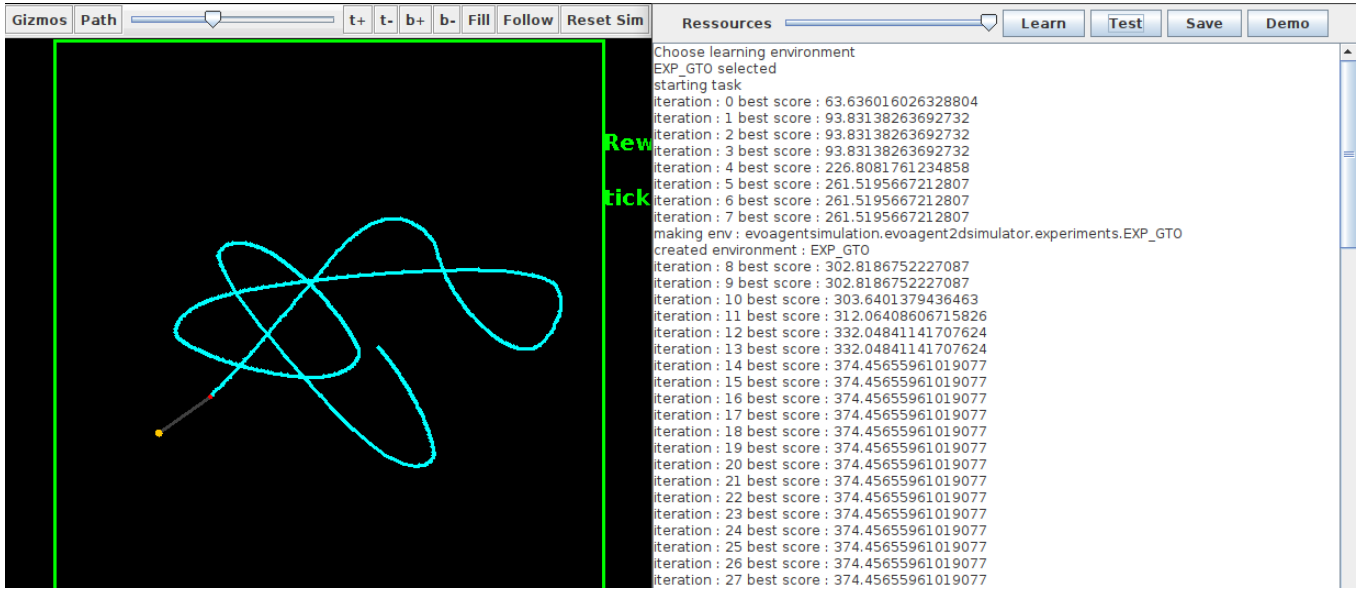
Classe interne *Individual*

1. `public void random()` : Génère un génome aléatoire.
2. `public Individual crossbreed(Individual i2)` : Fonction de croisement. Retourne un nouvel individu.
3. `public void mutation(double probabilite)` : Applique une mutation sur l'individu.
4. `public int compareTo(Individual compare)`

Classe *GeneticAlgorithm*

1. `public void initialise()` : Initialise l'algorithme génétique (par exemple avec une population aléatoire).
2. `public void breedNew()` : Génère une nouvelle population à partir de la population actuelle. La nouvelle population remplace celle de la liste `population`.

TP : EvoAgentStudents



Classes EXP

1. `public EvoAgentMind makeMind()`: Configurez votre contrôleur
2. `public GeneticAlgorithm makeGeneticAlgorihm(int gS)`: Configurez votre algorithme génétique.
3. `public void initialisation()`: Définie toutes les propriétés de l'environnement et les fonctions de récompense.