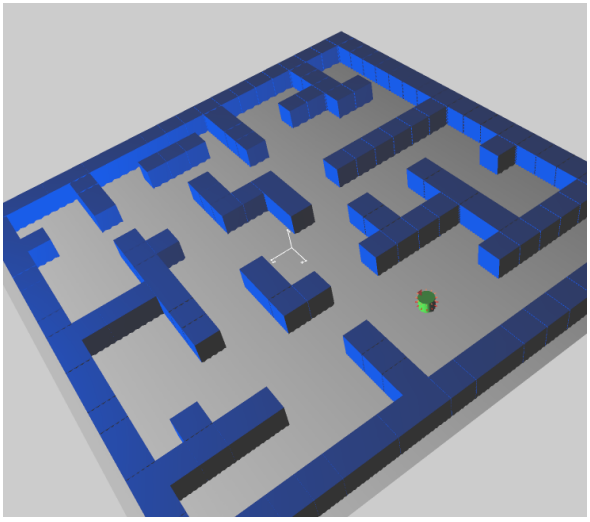
Apprentissage et optimisation pour la robotique autonome

L3 IARO

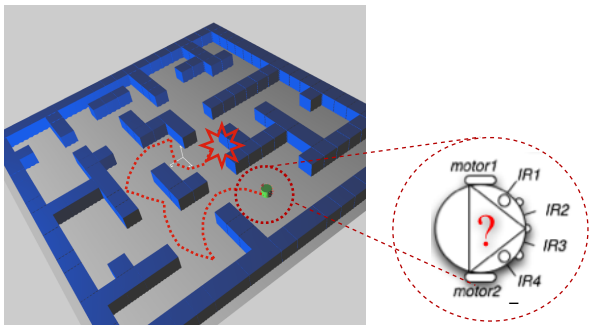
Nicolas Bredeche
Université Pierre et Marie Curie
ISIR, UMR 7222
Paris, France
nicolas.bredeche@upmc.fr

…



mise à jour: 2018-04-04

Etude de cas



Objectif: maximiser l'exploration

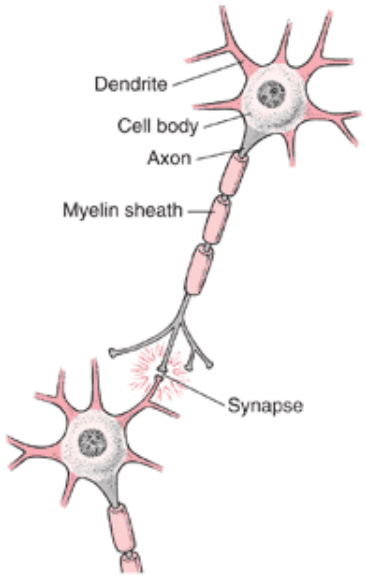
- Eléments à définir:
  - la fonction objectif (p.ex. maximiser la distance parcourue)
  - le formalisme de contrôle (p.ex. combinaison linéaire des entrées sensorielles)
  - l'espace de recherche (p.ex. les paramètres de la combinaison linéaire)
  - la méthode (p.ex. la recherche au hasard)

[Nolfi, Floreano, 2000]

Espace de recherche et contrôleur

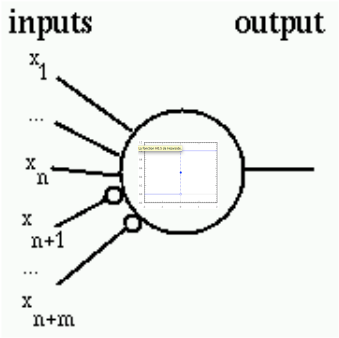
Réseaux de neurones artificiels
perceptron multi-couches

Qu'est ce qu'un neurone?



En pratique: un neurone artificiel n'est qu'***inspiré*** du neurone réel.

Qu'est ce qu'un neurone *artificiel?*

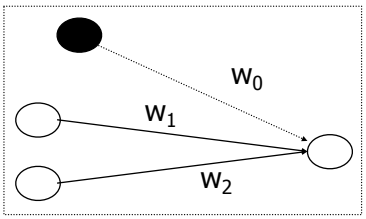


- La forme la plus simple: le neurone formel [McCulloch, Pitts, 1943]
  - Poids synaptiques
    - excitation, inhibition
  - Fonction d'activation $\phi(\sum_{i=0}^{n} w_i * x_i)$
    - fonction de Heaviside

$$H(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0 \end{cases}$$

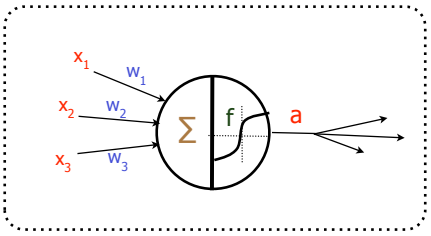
Perceptron

- Perceptron [Rosenblatt, 1957]
  - Neurones formels
  - Ajout d'un neurone de biais
  - Fonction d'activation non-linéaire *et* dérivable



- Calcul de l'activité d'un neurone:

$$a = f_{activation}(\sum_{i=0}^{n}(w_i * x_i))$$

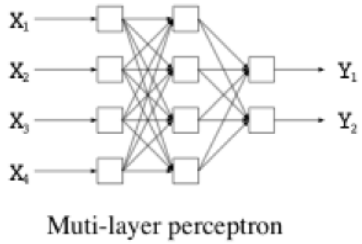


- Notation:
  - x : activité interne du neurone (ie. avant fonction d'activation)
  - a : activité du neurone en sortie (ie. après fonc. d'activation)

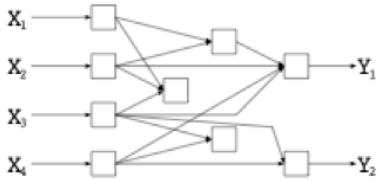
Variantes

- Fonctions d'agrégation / de combinaison
  - Combinaison linéaire des entrées (cf. transp. précédent)
  - Radial Basis Function (RBF)
    - norme euclidienne de la différence entre les vecteurs d'entrées
- Fonctions d'activation
  - Heaviside (ie. non-linéaire, non dérivable)
  - Linéaire (ie. équivalent à une matrice de transformation)
  - Sigmoïde, Tangente hyperbolique (ie. non-linéaire, dérivable)
  - Gaussienne (ie. non-linéaire, dérivable)
  - Rectified Linear Unit (ie. 0 si x<0 ; x sinon) [fréquent dans les réseaux profonds]

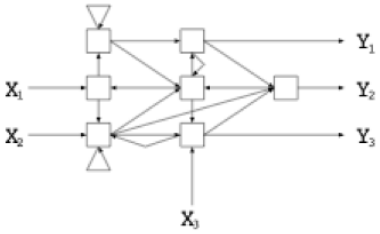
Topologie



Muti-layer perceptron



"Feed-forward" network



Recurrent Network

- Réseaux à propagation directe ["feed-forward NN"]
  - À chaque instant : $F:\Re_n \Rightarrow \Re_m$
- Réseaux récurrents ["recurrent NN"]
  - À chaque instant : l'activation de chaque neurone à *t* peut dépendre:
    - des entrées
    - de l'état du réseau à *t-1*

Espace de recherche

Question: qu'est ce qui peut être modifié dans un réseau?

- Paramètres
  - Poids des arcs
  - Paramètres de la fonction d'activation
- Topologie
  - Ajout/suppression des arcs
  - Ajout/suppression des noeuds

Théorie et pratiques

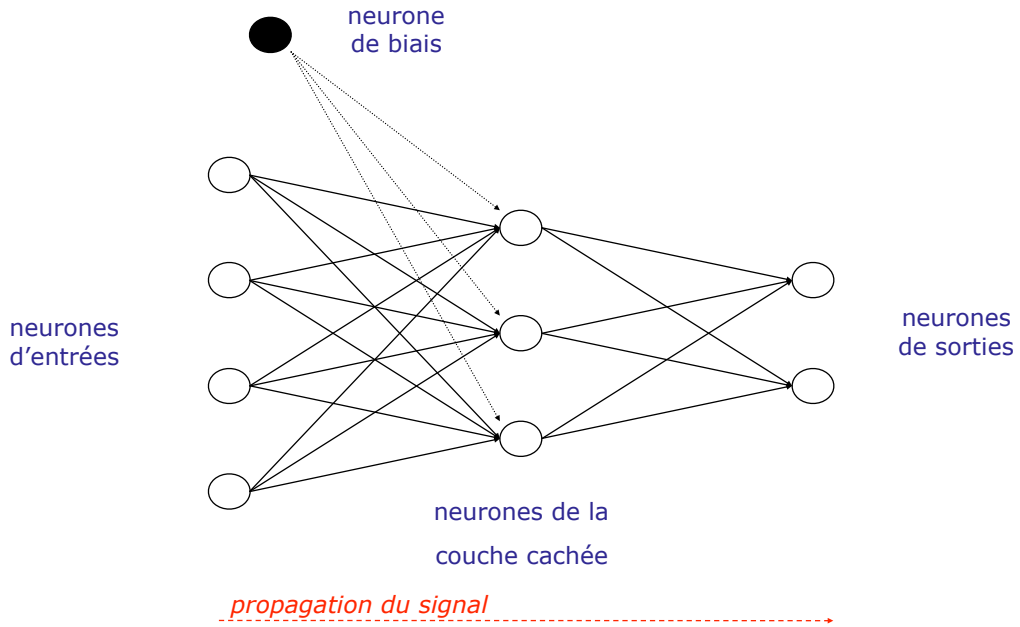
- Résultats théoriques
  - approximateur universel
    - une couches "cachée" suffit, si suffisamment de neurones.
    - régression [Cybenko 1988-89], classification [Hornik 1989]
- Propriétés empiriques
  - Bonne résistance au bruit
  - Bonnes propriétés de généralisation
  - Meilleur si domaine plutôt convexe
    - « Si A et B positifs, tout le segment AB est positif »
    - Sensible lors de dépendances sur les entrées
      - P.ex. le nombre d'entrées à 1 est paire ou impaire ?
      - Un perceptron mono-couche n'y arrive pas; un PMC difficilement

Repères historiques

- Inspiration biologique: « neuro-mimétisme »
  - 1943 : neurone formel de McCulloch&Pitts
  - 1949 : Règle de Hebb
- Première époque (1959-1969) [Rosenblatt 59, Widrow&Hoff 60, Minsky&Papert 69]
  - 1959 : perceptron et règle d'apprentissage
  - 1969 : Limites du perceptron (aux problèmes linéairement séparables)
- Seconde époque (1986-…) [Rumelhart&McClelland 86][LeCun 86][Werbos 74][Parker 82]
  - 1986 : Perceptron Multi-Couches (pour les problèmes linéairement non-séparables)
  - Algorithme de rétro-propagation du gradient pour les PMC
  - Algorithme d'apprentissage des poids *et* de la structure
- Aujourd'hui (~2000-...)
  - Dynamique interne et prédiction de séries temporelles (ESN, LSM)
  - Optimisation paramétrique et non-paramétrique des réseaux de neurones
  - Réseaux de neurones profonds (Deep NN) et/ou convolutionnel

Perceptron Multi-Couches (MLP)

[LeCun, 1984] [Rumelhart et McLelland, 1984][Werbos, 1974][Parker, 1982]



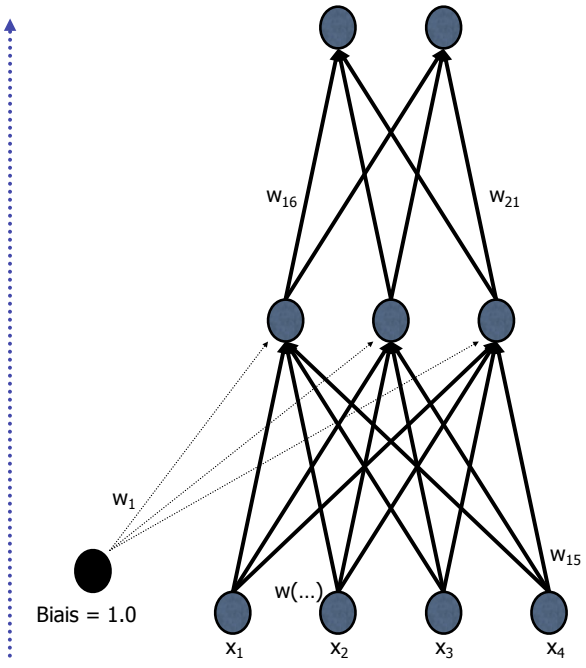
Propriété requise : fonction d'activation non-linéaire et dérivable

Perceptron Multi-Couches (MLP)

[LeCun, 1984] [Rumelhart et McLelland, 1984][Werbos, 1974][Parker, 1982]

- Propriétés requises :
  - couche cachée: fonction d'activation non-linéaire et dérivable
- En pratique:
  - couche cachée:
    - fonction sigmoïde $f(x) = 1/(1+e^{-kx})$ equiv. à : $f(x) = (1+tanh(x))/2$ dérivé: $f'(x) = f(x)(1-f(x))$
    - tangente hyperbolique
  - couche de sortie : fonction linéaire
- Remarques
  - paramètre clé: le nombre de neurones cachés
    - Remarque: évidemment, la topologie, le nombre de couche, ont aussi une influence, mais sont aussi plus difficile à régler.

Propagation du signal dans un MLP

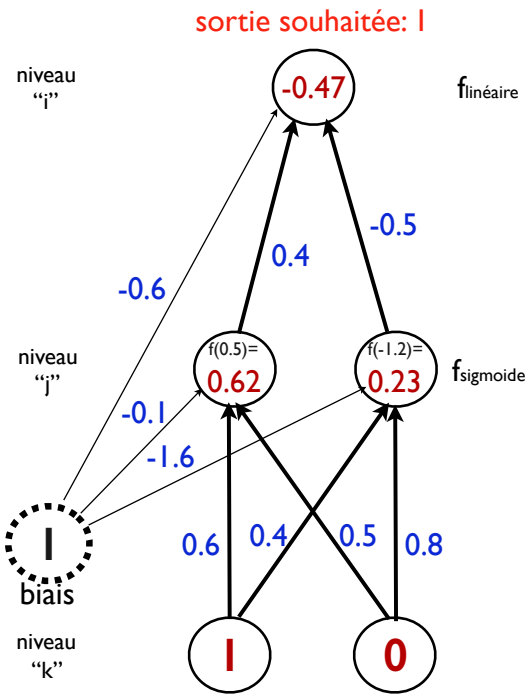


$$a_i = \sum_{j=0}^{n} w_{ji} a_j$$

On utilise une fonction d'activation linéaire

$$a_j = f_{activation}\left(\sum_{k=0}^{n} w_{kj} x_k\right)$$

$$a_k = (inputs)$$



$$a_i = \sum_{j=0}^{n} w_{ji} a_j$$

$a_{i=1}$ = 0.4*0.62 + -0.5*0.23 + -0.6 = -0.47

$$a_j = f_{activation}\left(\sum_{k=0}^{n} w_{kj} x_k\right)$$

$a_{j=1}$ = f_{sig}(0.6*1 + 0.5*0 + -0.1) = f_{sig}(0.5) = 0.62

$a_{j=2}$ = f_{sig}(0.4*1 + 0.8*0 + -1.6) = f_{sig}(-1.2) = 0.23

$$a_k = (inputs)$$

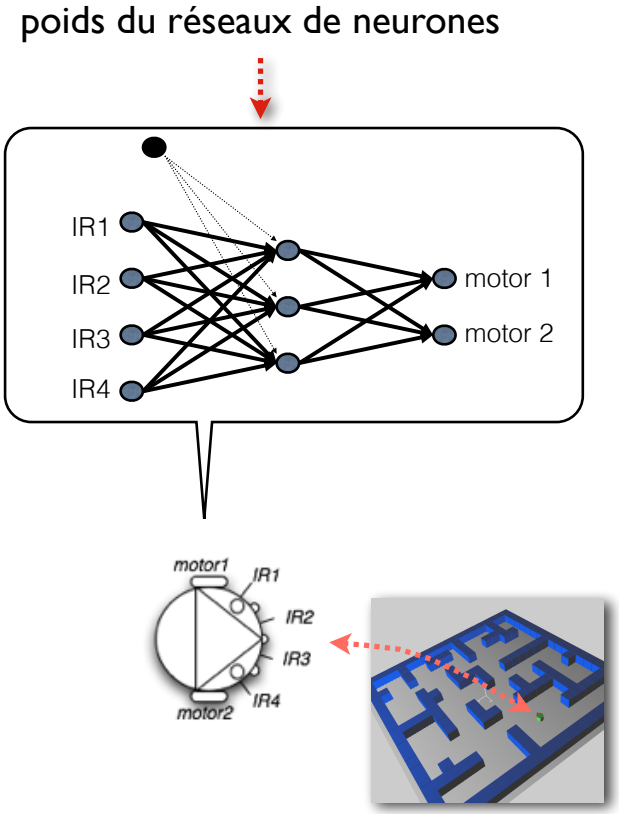
$a_{k=1}$ = 1

$a_{k=2}$ = 0

activation: sigmoide $f(x) = 1/(1+e^{-kx})$ pour la couche cachée, et linéaire pour les sorties

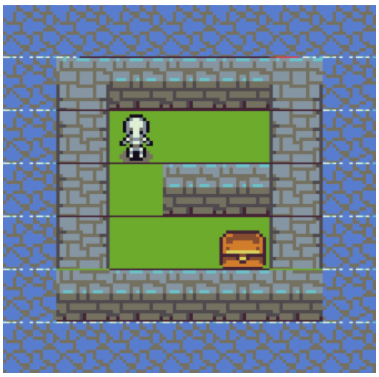
Méthodes d'apprentissage

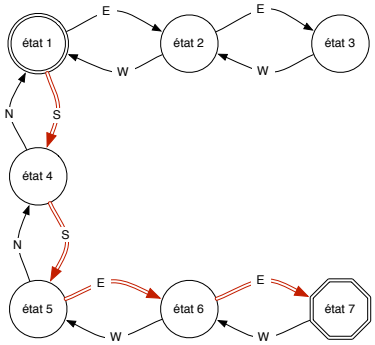
Apprentissage par renforcement par évolution artificielle
Apprentissage supervisé et algorithme de rétro-propagation



- Comment apprendre?
  - Apprentissage par renforcement
    - Lors de l'apprentissage, on doit effectuer une séquence d'actions avant d'obtenir une estimation de la qualité de notre stratégie
    - Méthode: model-based RL, recherche de politique
  - Apprentissage supervisé
    - Lors de l'apprentissage, on connait la réponse attendue pour chaque pas de temps (i.e. on dispose d'un "oracle").
    - Méthode: algorithme de rétro-propagation du gradient, …

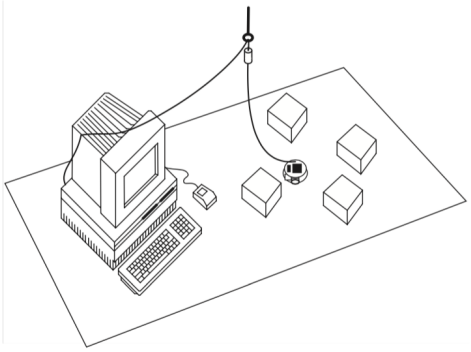
Apprentissage par renforcement





- **Objectif:**
  - trouver la politique optimale qui maximise la récompense globale
  - on veut $\forall s_t , \pi^*(s_t) \longrightarrow a_t^*$ avec un espérance de gain max.
- Notations:
  - a_t^* action optimale à *t* ; ($a_t^* \in A$)
  - s_t état courant à *t* ; ($s_t \in S$)

Optimisation pour la navigation

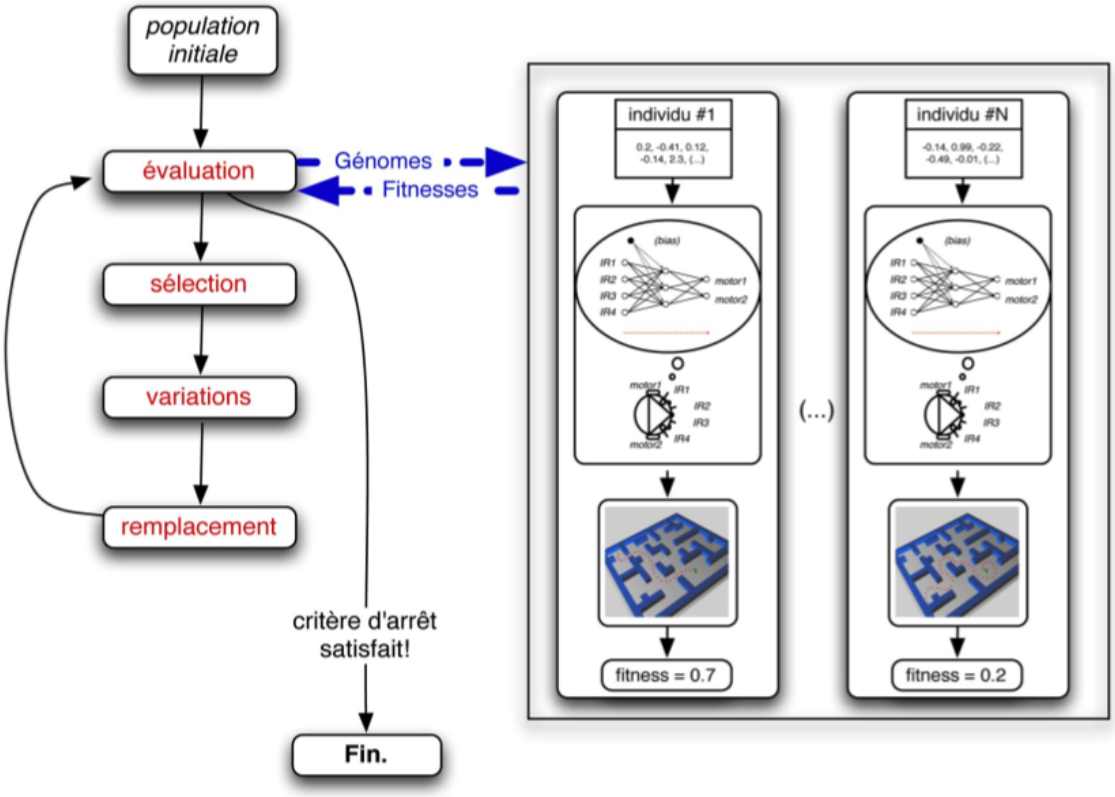


$$fitness = \sum_{t=0}^{evalTime} (v_t * (1 - v_r) * (minSensorValue))$$

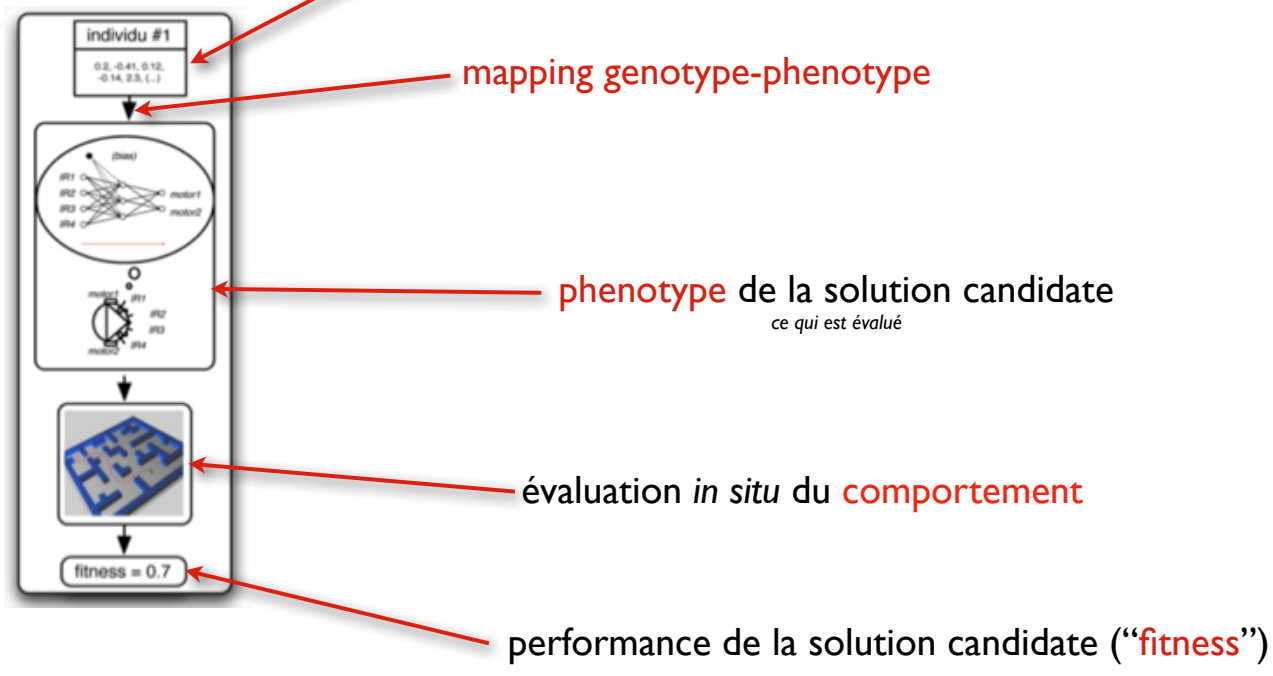
Vt: vitesse de translation
Vr: vitesse de rotation
MinSensorValue: valeur du senseur le plus "stimulé"
si aucun obstacle: renvoi 1
sinon: renvoi une valeur dans [0,1[

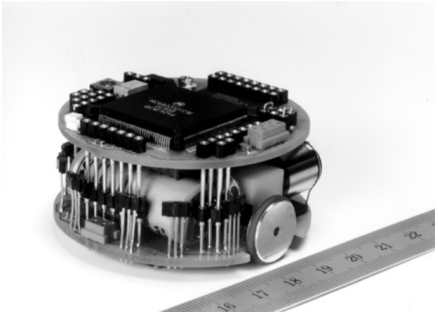
Remarques:
chaque terme est normalisé entre 0 et 1
la vitesse de rotation est donnée en valeur absolue

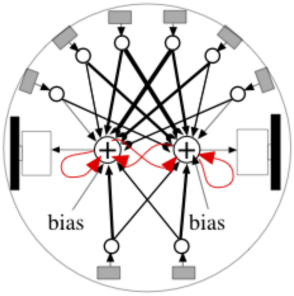
[Floreano, Mondada 1994][Nolfi, Floreano, 2000]



cf. [Nolfi, Floreano, 2000][Doncieux et al. 2015] pour une introduction complète



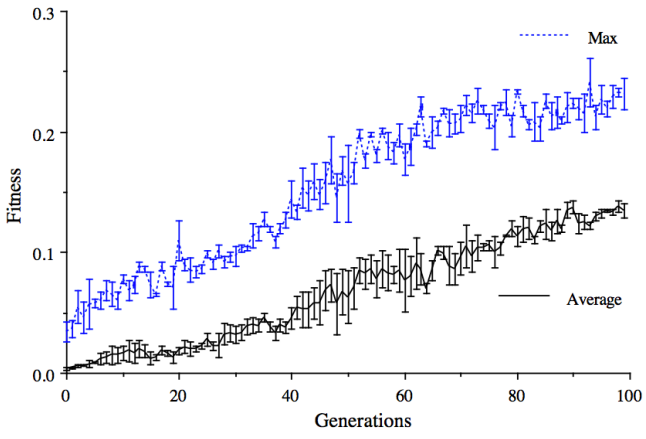




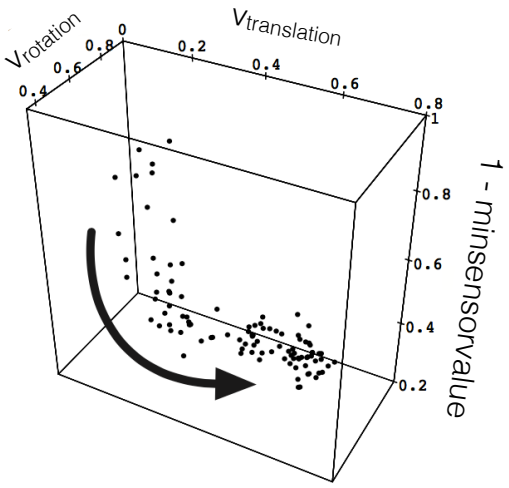
Population size	80
Generation number	100
Crossover probability	0.1
Mutation probability	0.2
Mutation range	±0.5
Initial weight range	±0.5
Final weight range	Not bounded
Life length	80 actions
Action duration	300 ms

- conditions expérimentales
  - Réseau de neurones fct sigmoïde, génome: poids et seuil d'activ., cnx récurrentes
  - Algorithme génétique sélection par roulette, mutation, cross-over
  - Robot réel 40 min par génération; 100 générations; durée totale: 66 heures!

Average and best individual fitnesses over generations (3 trials/dot)

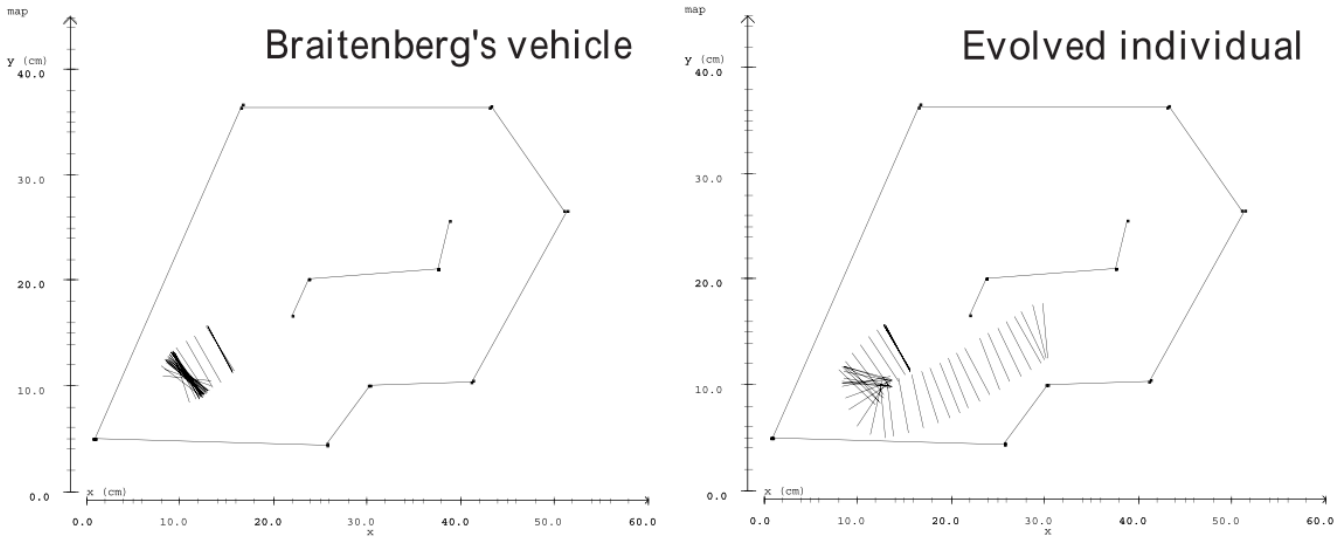


- Premières générations:
  - tournent sur place; foncent dans les murs
- après 20 générations:
  - évite les obstacles; tournent sur place
- après 50 générations:
  - rapide; évitent les obstacles; indépendant du point de départ

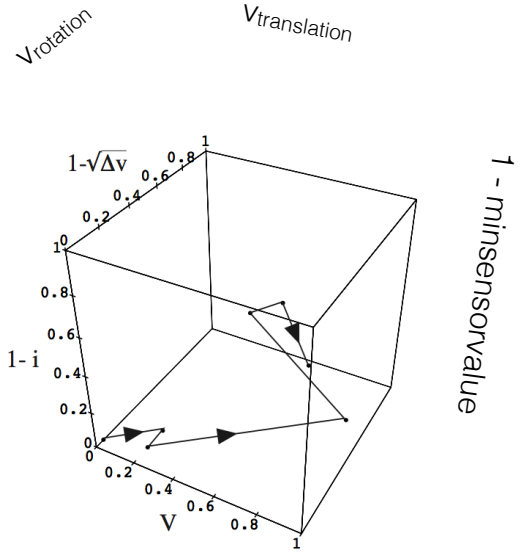


extrait de: [Floreano, Mondada, 1994], légende adaptée

- Premières générations:
  - tournent sur place; foncent dans les murs
- après 20 générations:
  - évite les obstacles; tournent sur place
- après 50 générations:
  - rapide; évitent les obstacles; indépendant du point de départ



Comparaison qualitative



Trajectoire d'un robot dans l'espace de la fitness
changement au cours du temps de la valeur de fitness instantanée
à partir d'un point de départ défavorable (coincé dans un coin)

 [Floreano, Mondada 1994][Nolfi, Floreano, 2000]

- Comment apprendre?
  - Apprentissage par renforcement
    - Lors de l'apprentissage, on doit effectuer une séquence d'actions avant d'obtenir une estimation de la qualité de notre stratégie
    - Méthode: model-based RL, recherche de politique
  - Apprentissage supervisé
    - Lors de l'apprentissage, on connait la réponse attendue pour chaque pas de temps (i.e. on dispose d'un "oracle").
    - Méthode: algorithme de rétro-propagation du gradient, …

Remarque: cette partie n'est pas pertinente pour le projet

Apprentissage supervisé

- Le problème:

  $\{(x_i, y_i), x_i \in \mathcal{X}, y_i \in \mathcal{Y}\}, \mathcal{Y} = \{-1, 1\}$ or $\mathbf{R}$ (classification / regression)

- Une fonction de perte (l'oracle):

  $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbf{R}^+$

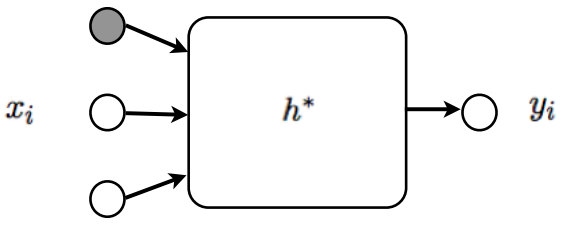
- Objectif:
  - Trouver l'hypothèse qui minimise l'erreur de prédiction

    $h^* : \mathcal{X} \mapsto \mathcal{Y}$

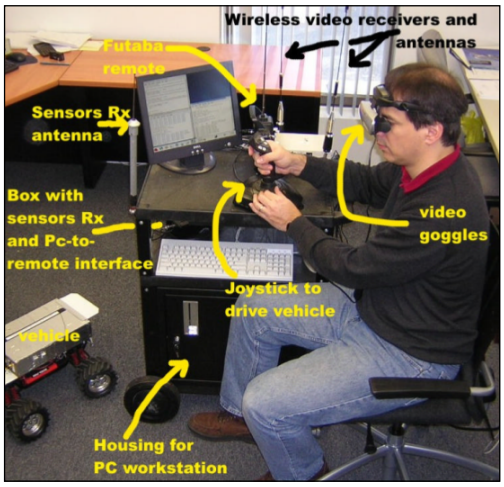
    tel que: $L(h^*) = argmin\{L(h),\ h \in H\}$

Hypothèse cherchée

$\{(x_i, y_i), x_i \in \mathcal{X}, y_i \in \mathcal{Y}\}, \mathcal{Y} = \{-1, 1\}$ or $\mathbf{R}$



h* peut être représenté par de multiples formalismes... un réseau de neurones, des règles, un arbre de décision, SVM, etc.



"Dave", autonomous off-road vehicle

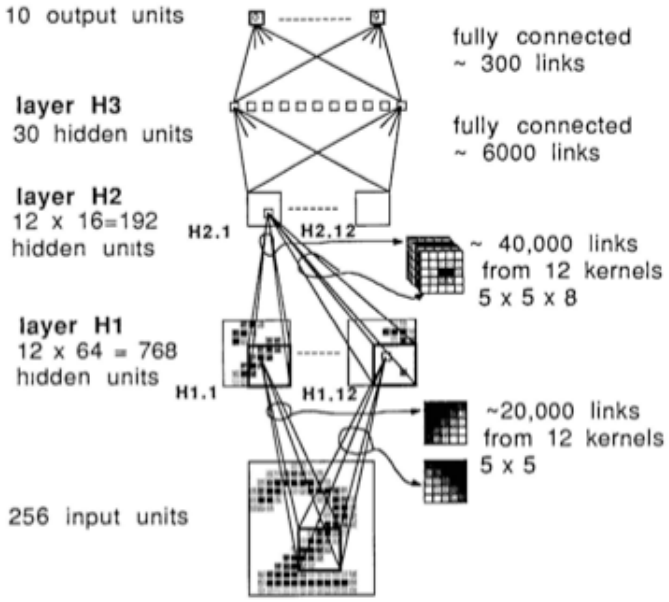
entrée: image stéréoscopique acquise par les caméras
sortie: contrôle des moteurs gauche et droite

Problème d'apprentissage supervisé
minimiser l'erreur entre la sortie prédite et la sortie donnée par l'oracle (ici: le pilote)

"Dave", autonomous off-road vehicle

entrée: image stéréoscopique acquise par les caméras
sortie: contrôle des moteurs gauche et droite

Problème d'apprentissage supervisé
minimiser l'erreur entre la sortie prédite et la sortie donnée par l'oracle (ici: le pilote)

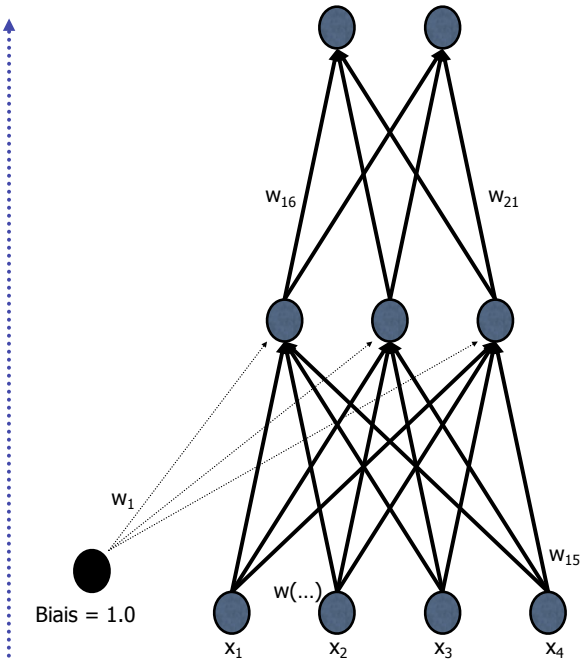


9760 poids (au lieu de 200000 poids si on connecte tout)

7300 exemples d'apprentissage, 2000 exemples de tests, 99% de précision.

Propagation du signal dans un MLP

rappel des slides précédents



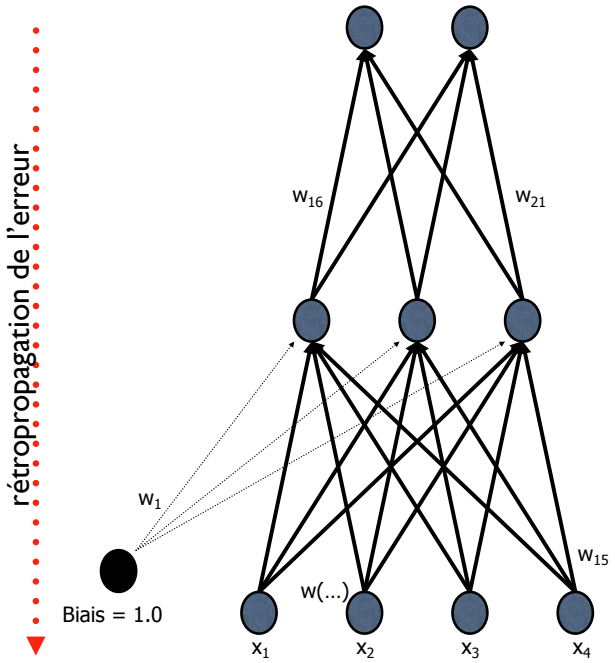
$$a_i = \sum_{j=0}^{n} w_{ji} a_j$$

On utilise une fonction d'activation linéaire

$$a_j = f_{activation}\left(\sum_{k=0}^{n} w_{kj} x_k\right)$$

$$a_k = (inputs)$$

Rétro-propagation de l'erreur



$$\delta_i = f'_{activation}(x_i) * (a_i^* - a_i)$$

$$w_{ji}^{t+1} = w_{ji}^t + \mu * \delta_i * a_j$$

$$\delta_j = f'_{activation}(x_j) * (\sum_{i=0}^{nb_{sorties}} (w_{ji} * \delta_i))$$

$$w_{kj}^{t+1} = w_{kj}^t + \mu * \delta_j * a_k$$

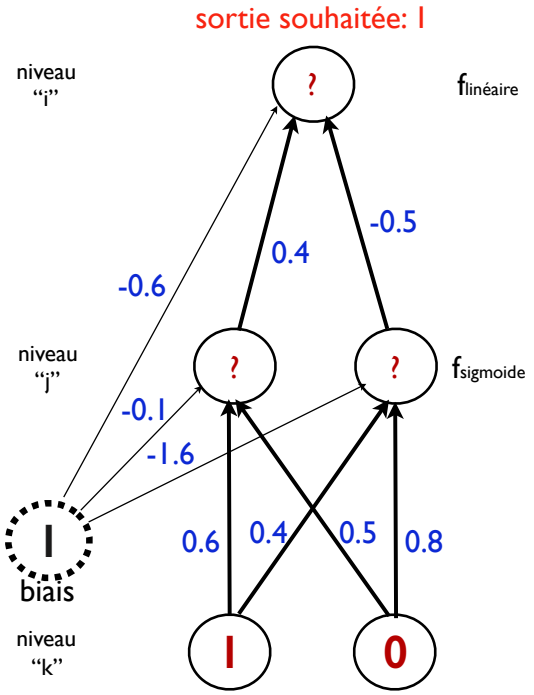
$a_i^* : i^{eme} sortie\_desiree$

$\mu : pas\_d'apprentissage$

$\delta : "sensibilite"$

«back-propagation algorithm»

Mise en pratique : propagation et rétro-propagation



Imaginons:

- sortie attendue: "1"

Objectif:

Régler les poids du réseau.

Etude pratique:

1. propagation du signal
2. rétro-propagation de l'erreur

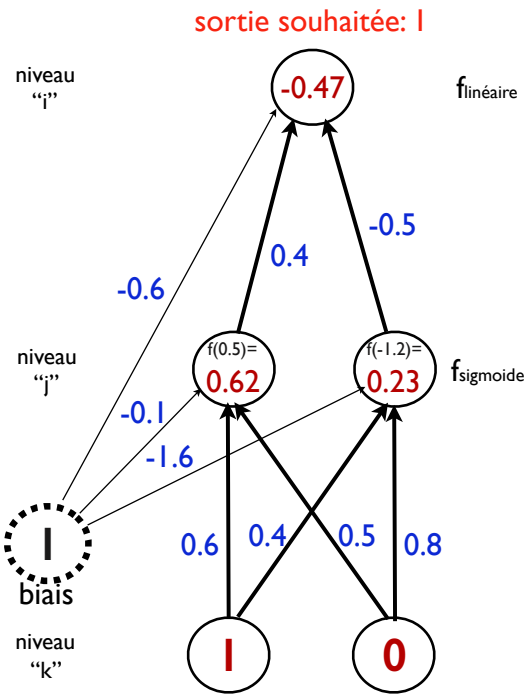
Remarque: *en pratique, il faudrait faire cela un grand nombre de fois pour chaque exemple de la base d'apprentissage.*

activation: sigmoide $f(x) = 1/(1 + e^{-kx})$ pour la couche cachée, et linéaire pour les sorties

dérivé sigmoïde: $f'(x) = f(x)(1 - f(x))$

Ici, on fixe mu=0.1 (pas d'apprentissage)

Etape 1 : propagation du signal



$$a_i = \sum_{j=0}^{n} w_{ji} a_j$$

$a_{i=1}$ = 0.4*0.62 + -0.5*0.23 + -0.6 = -0.47

$$a_j = f_{activation}\left(\sum_{k=0}^{n} w_{kj} x_k\right)$$

$a_{j=1}$ = f_{sig}(0.6*1 + 0.5*0 + -0.1) = f_{sig}(0.5) = 0.62

$a_{j=2}$ = f_{sig}(0.4*1 + 0.8*0 + -1.6) = f_{sig}(-1.2) = 0.23

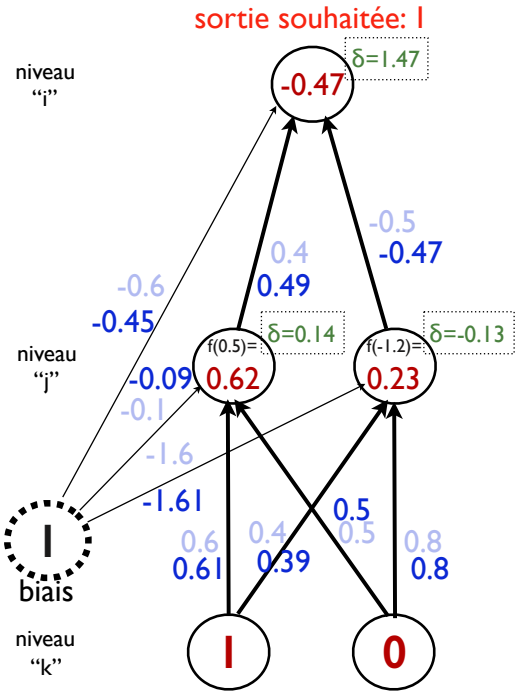
$$a_k = (inputs)$$

$a_{k=1}$ = 1

$a_{k=2}$ = 0

activation: sigmoide $f(x) = 1/(1+e^{-kx})$ pour la couche cachée, et linéaire pour les sorties

Etape 2 : rétro-propagation de l'erreur et correction



1.a $\delta_i = f'_{activation}(x_i) * (a_i^* - a_i)$

$\delta_{i=1}$ = $f_{linéaire}$'(-0.47) * (1 - -0.47) = 1 * 1.47 = 1.47

1.b $\delta_j = f'_{activation}(x_j) * \left(\sum_{i=0}^{nb_{sorties}} (w_{ji} * \delta_i)\right)$

$\delta_{j=1}$ = f_a'(0.5) * (0.4*1.47) = (f_a(0.5) * (1 - f_a(0.5))*0.59

= (0.62 * 0.38) * 0.59 = 0.14

$\delta_{j=2}$ = f_a'(-1.2) * (-0.5* 1.47) = (0.23 * 0.77) * 0.74 = -0.13

2.a $w_{ji}^{t+1} = w_{ji}^t + \mu * \delta_i * a_j$

$w_{j=1,i=1}$ = 0.4 + 0.1*1.47*0.62 = 0.49

$w_{j=2,i=1}$ = -0.5 + 0.1*1.47*0.23 = -0.47

$w_{bias,i=1}$ = -0.6 + 0.1*1.47*1 = -0.45

2.b $w_{kj}^{t+1} = w_{kj}^t + \mu * \delta_j * a_k$

$w_{k=1,j=1}$ = 0.6 + 0.1*0.14*1 = 0.61

$w_{k=2,j=1}$ = 0.5 + 0.1*0.14*0 = 0.5

$w_{bias,j=1}$ = -0.1 + 0.1*0.14*1 = -0.09

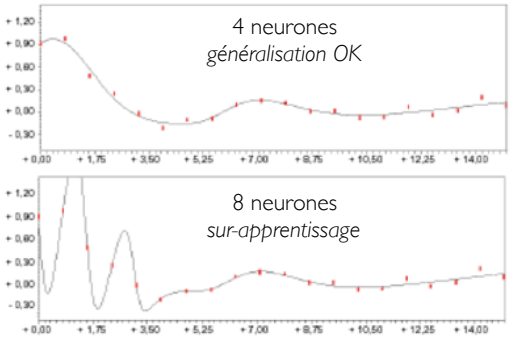
$w_{k=1,j=2}$ = 0.4 + 0.1*-0.13*1 = 0.39

$w_{k=2,j=2}$ = 0.8 + 0.1*-0.13*0 = 0.8

$w_{bias,j=2}$ = -1.6 + 0.1*-0.13*1 = -1.61

sigmoide: $f'(x) = f(x)(1-f(x))$ -- rappel: a=f(x) -- f'lin(x)=1 -- Ici, on fixe mu=0.1 (pas d'apprentissage)

Critères de succès



- Compromis mémorisation vs. généralisation
  - mémorisation: approximation +/- bonne des exemples
  - généralisation: risque de sur-apprentissage (= apprendre par coeur)
- Méthodologie:
  - Base d'apprentissage : exemples utilisés pour l'apprentissage (répétitions)
  - Base de test : exemples utilisés pour la validation (ex.: 10-fold cross-validation)

Crédit image: A. Cornuejols

Conclusions

- Ce qu'il faut retenir
  - Architecture de contrôle: réseaux de neurones artificiels
  - Espace de recherche: les poids du réseaux
  - Type de problèmes: appr. supervisé, appr. par renforcement
  - Méthodes: retro-propagation, évolution artificielle, etc.
    - différentes méthodes pour différents problèmes!
- Application dans le cadre du projet
  - Un nouveau formalisme pour la prise de décision d'un agent
  - Exemple d'évolution de comportements

Fin du cours