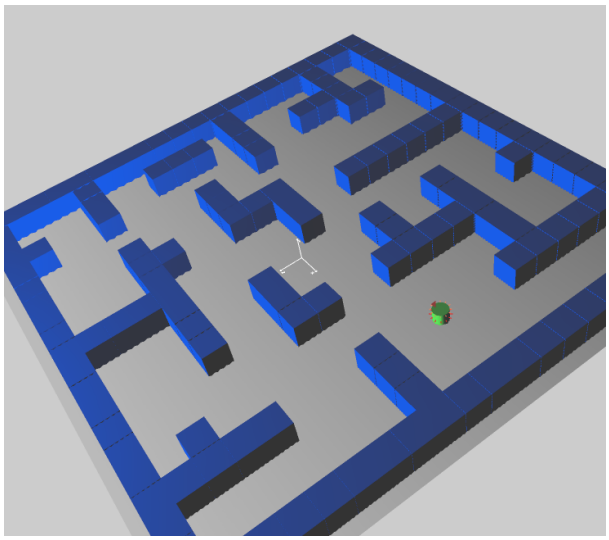
Optimisation par évolution artificielle: application à la robotique autonome

L3 IARO

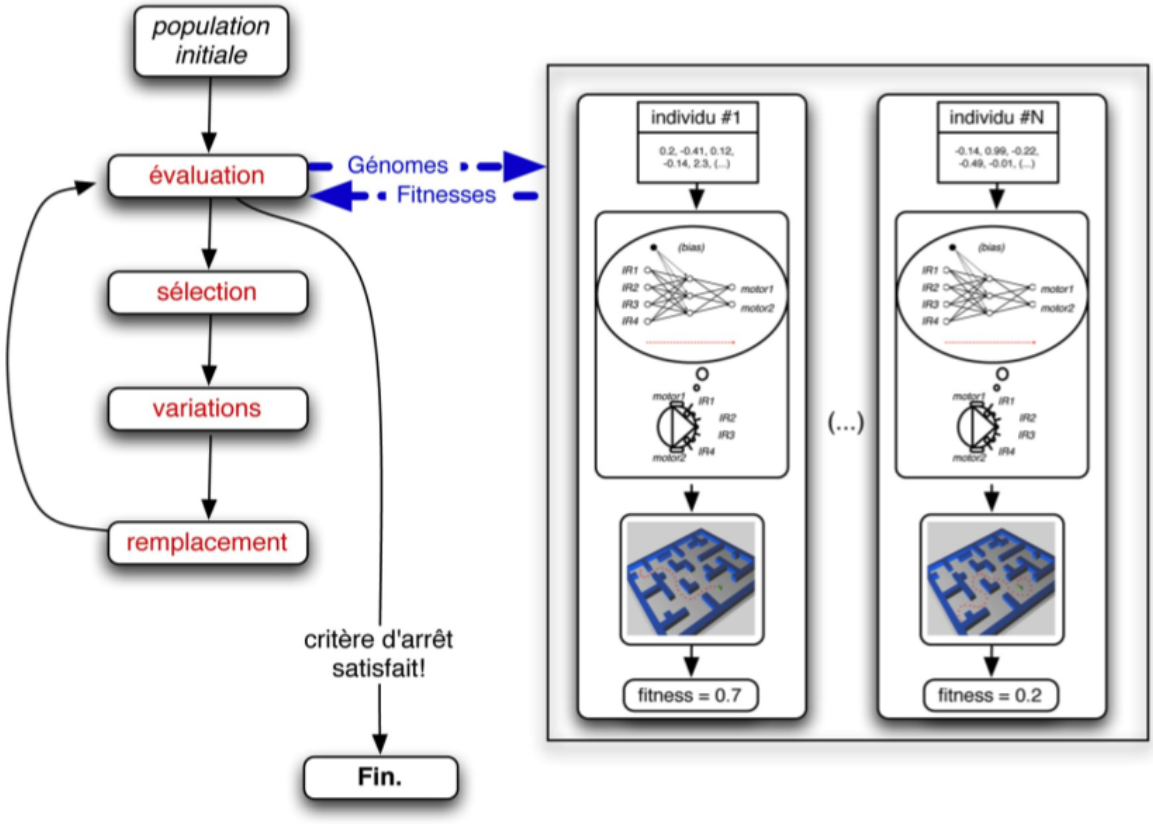
Nicolas Bredeche
Université Pierre et Marie Curie
ISIR, UMR 7222
Paris, France
nicolas.bredeche@upmc.fr

…

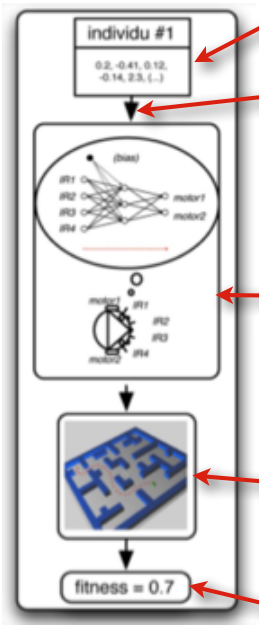


mise à jour: 2017-03-29

Evolution et robotique autonome



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



genome de la solution candidate

mapping genotype-phenotype

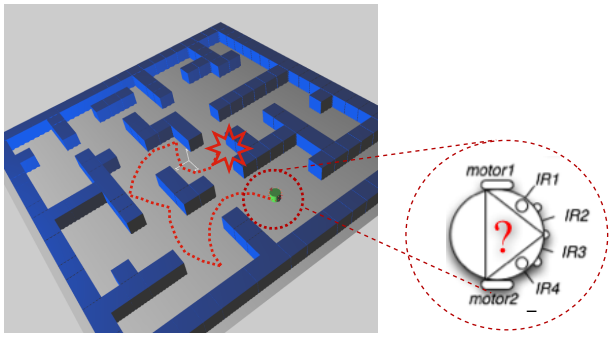
phenotype de la solution candidate

ce qui est évalué

évaluation *in situ* du comportement

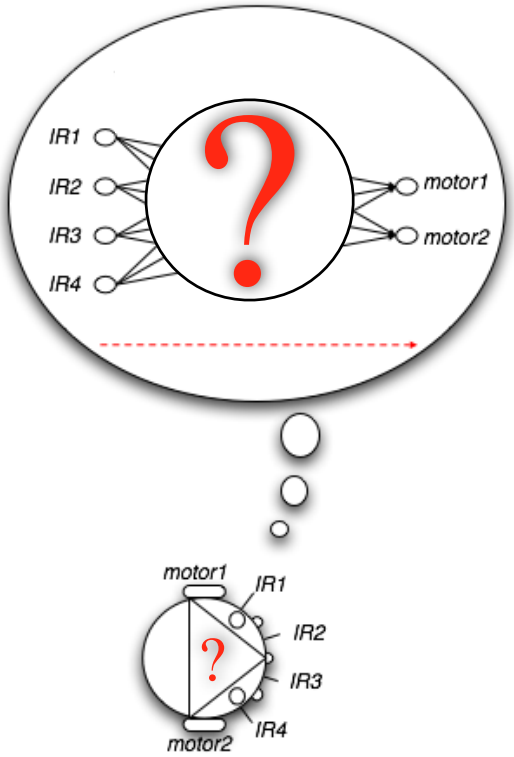
performance de la solution candidate ("fitness")

objectif: maximiser l'exploration



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

- Décomposition:
  - maximiser la vitesse de chaque moteur
  - maximiser la vitesse de translation
  - maximiser la distance au mur
- Fonction de performance *comportementale et embarquée*



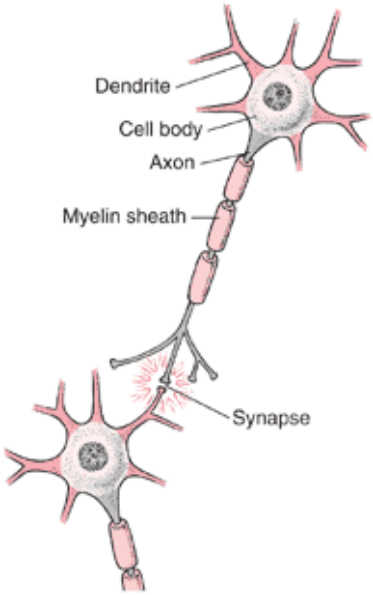
- ce qui est donné
  - entrées et sorties
  - formalisme de représentation
- ce qui peut être optimisé
  - paramètres (ex.: paramètres des neurones)
  - topologie (ex.: connexion entre neurones)
  - morphologie (ex.: assemblage de blocs)
  - ...

- Une tache
  - décrite sous forme fonctionnelle ou comportementale
  - ici: on souhaite maximiser le déplacement du robot
- Une mesure de performance
  - ... permettant l'évaluation d'une solution candidate
  - ex.: maximiser la vitesse de translation, minimiser la rotation
- Un espace de recherche
  - paramètres d'une combinaison linéaire… ou autre
  - ex.: réseaux de neurones artificiels
- Une méthode de conception
  - connaissances de l'expert, outils d'optimisation...
  - ex.: optimisation par évolution artificielle

Réseaux de neurones artificiels

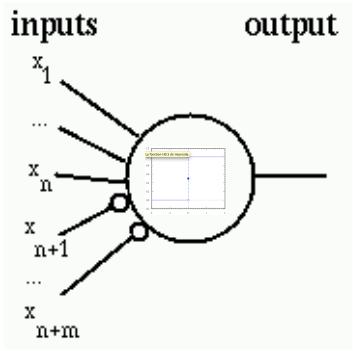
…comme formalisme de représentation pour la prise de décision

Qu'est ce qu'un neurone?



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

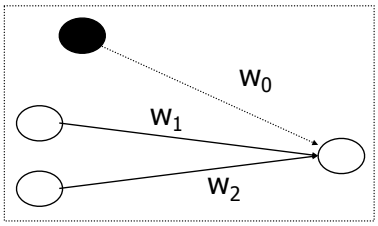
Qu'est ce qu'un neurone *artificiel?*



- 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

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

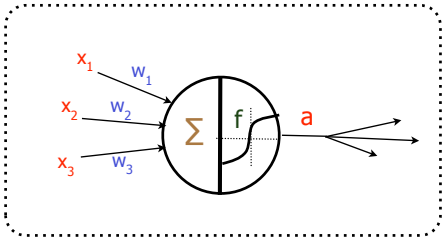
Perceptron



- 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:

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

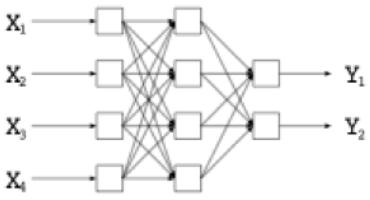


- Notation:
  - x : activité interne du neurone (ie. avant fonction d'activation)
  - a : activité du neurone en sortie (ie. après fonc. d'activation)

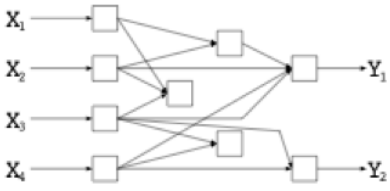
Variantes

- 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]

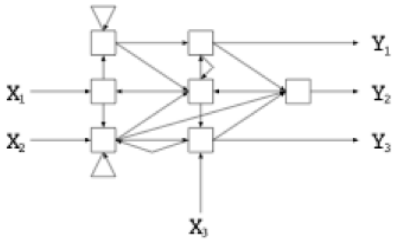
Topologie



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

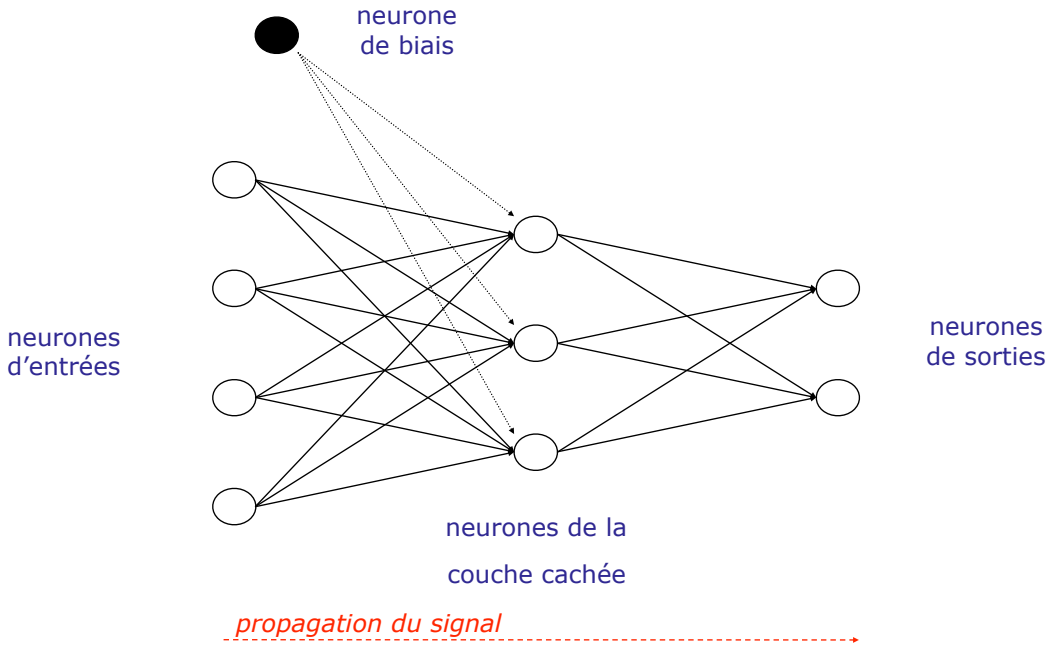
- Résultats théoriques
  - approximateur universel
    - une couches "cachée" suffit, si suffisamment de neurones.
    - régression [Cybenko 1988-89], classification [Hornik 1989]
- Propriétés empiriques
  - Bonne résistance au bruit
  - Bonnes propriétés de généralisation
  - Meilleur si domaine plutôt 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]



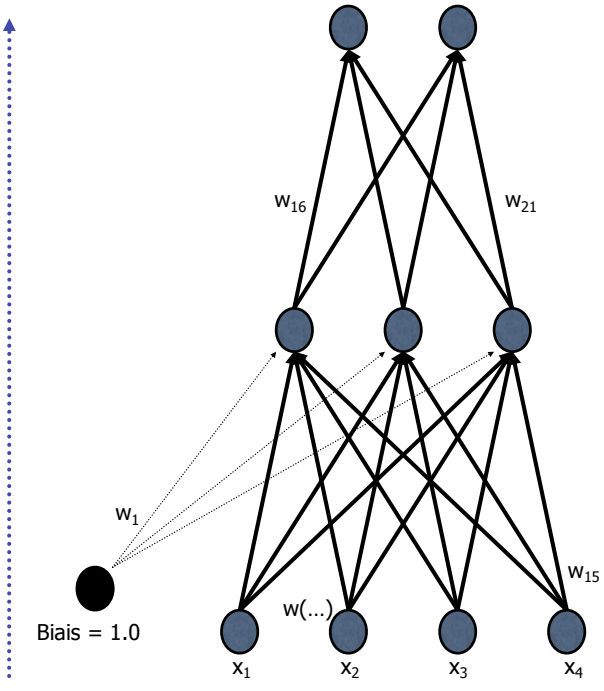
Propriété requise : fonction d'activation non-linéaire et dérivable

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



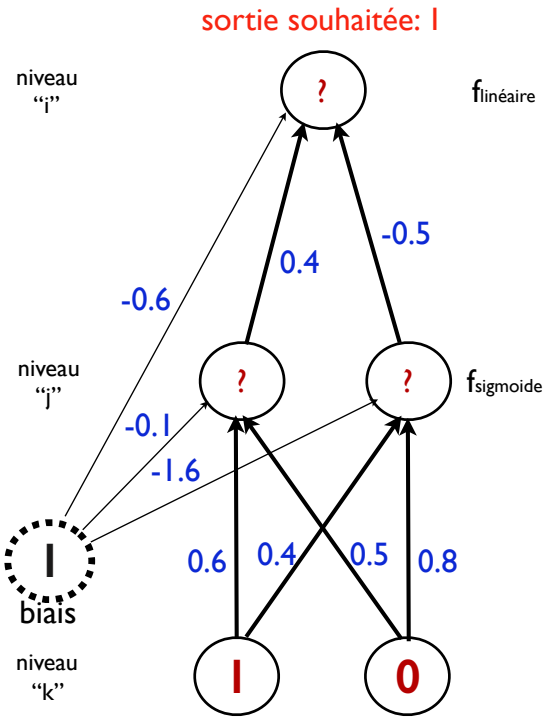
$$a_i = \sum_{j=0}^{n} w_{ji} a_j$$

On utilise une fonction d'activation linéaire

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

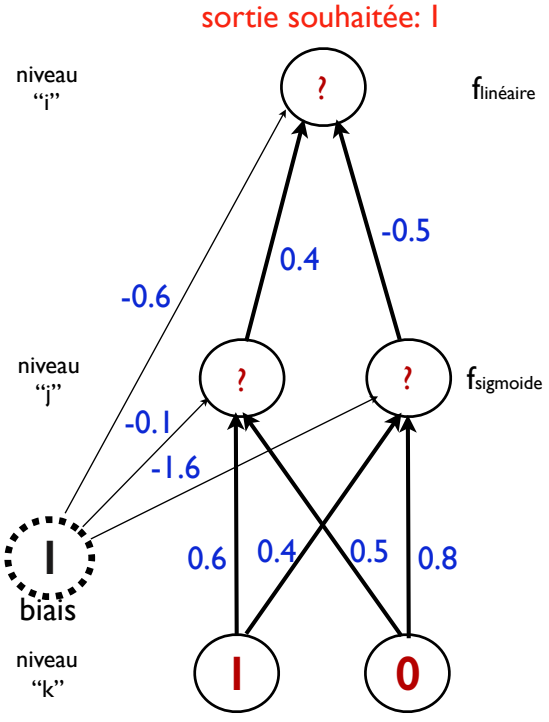
$$a_k = (inputs)$$

Mise en pratique : propagation et rétro-propagation



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

Etape 1 : propagation du signal



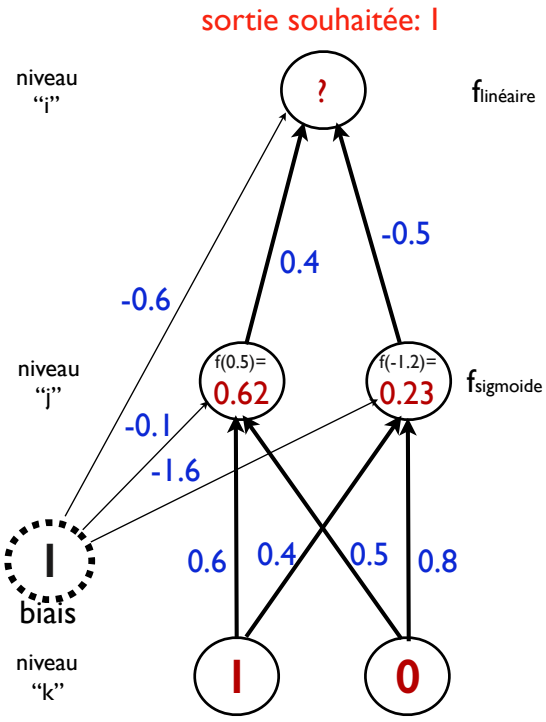
$$a_k = (inputs)$$

$a_{k=1} = 1$

$a_{k=2} = 0$

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

Etape 1 : propagation du signal



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

$a_{j=1}$ = 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

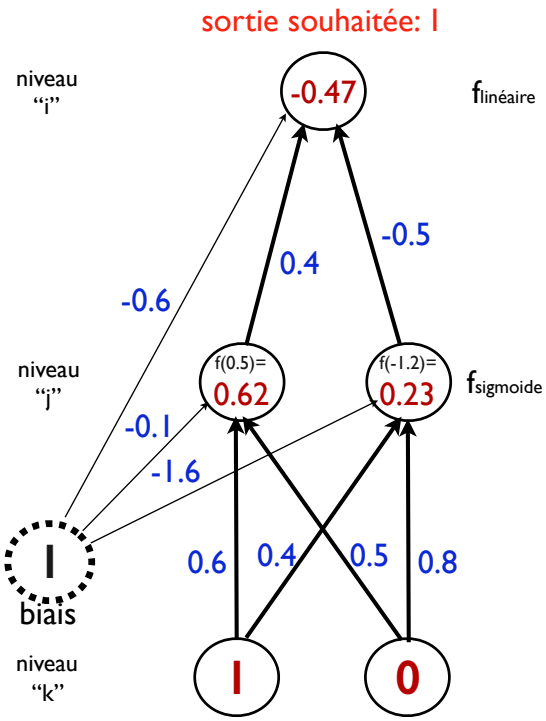
$$a_k = (inputs)$$

$a_{k=1}$ = 1

$a_{k=2}$ = 0

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

Etape 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

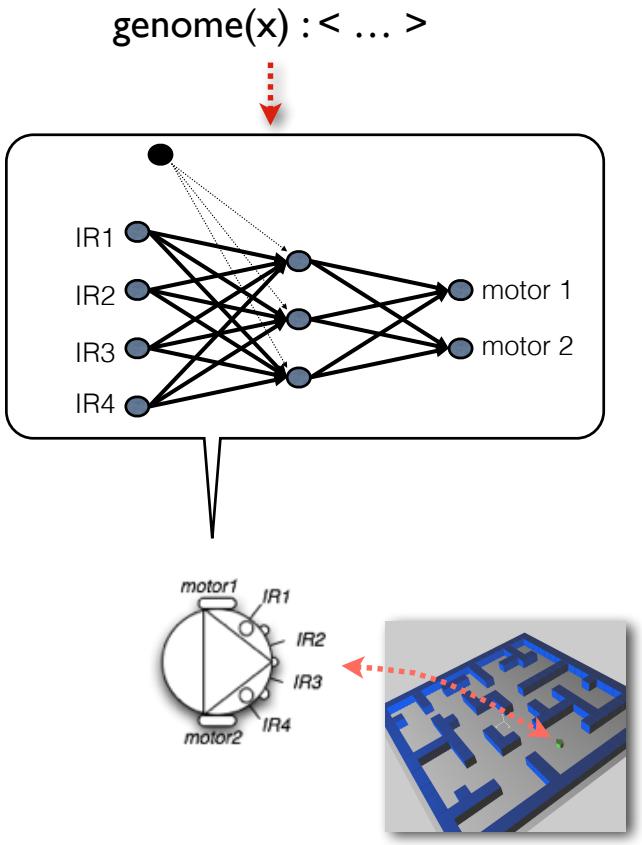
$a_{j=2}$ = f_{sig}(0.4*1 + 0.8*0 + -1.6) = f_{sig}(-1.2) = 0.23

$$a_k = (inputs)$$

$a_{k=1}$ = 1

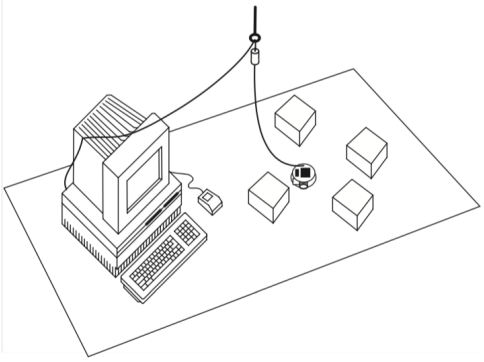
$a_{k=2}$ = 0

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



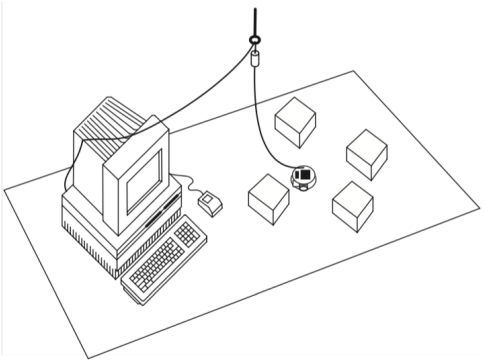
Etude de cas

Optimisation d'un comportement d'exploration simple

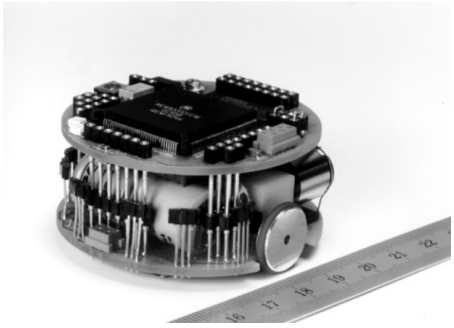


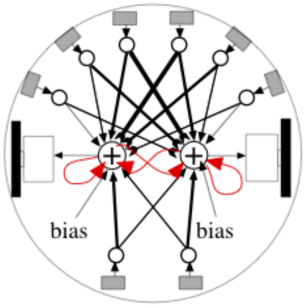
$$\Phi = V\left(1-\sqrt{\Delta v}\right)(1-i)$$

$$\begin{aligned} 0 &\leq V &&\leq 1 \\ 0 &\leq \Delta v &&\leq 1 \\ 0 &\leq i &&\leq 1 \end{aligned}$$



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

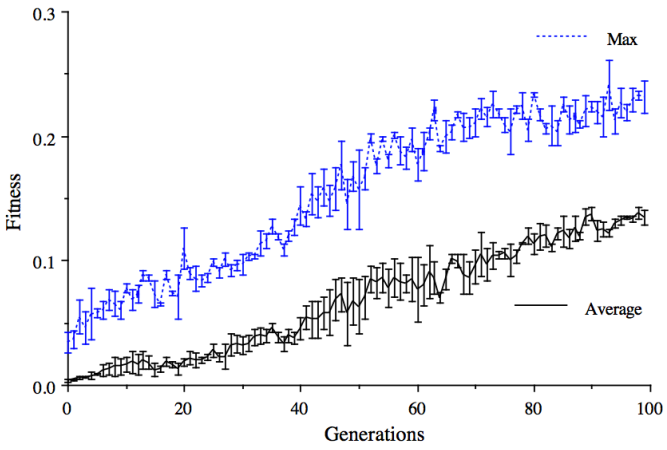




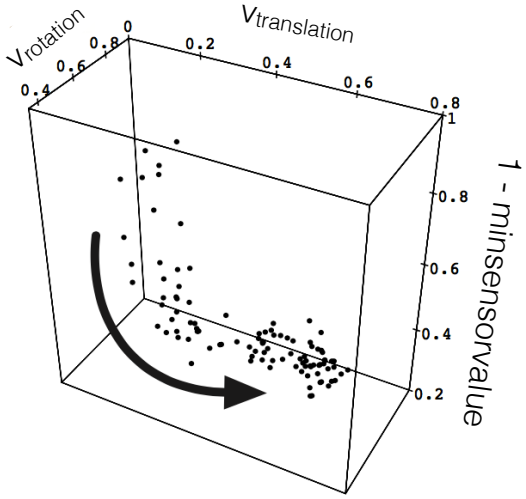
Population size	80
Generation number	100
Crossover probability	0.1
Mutation probability	0.2
Mutation range	±0.5
Initial weight range	±0.5
Final weight range	Not bounded
Life length	80 actions
Action duration	300 ms

- 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)

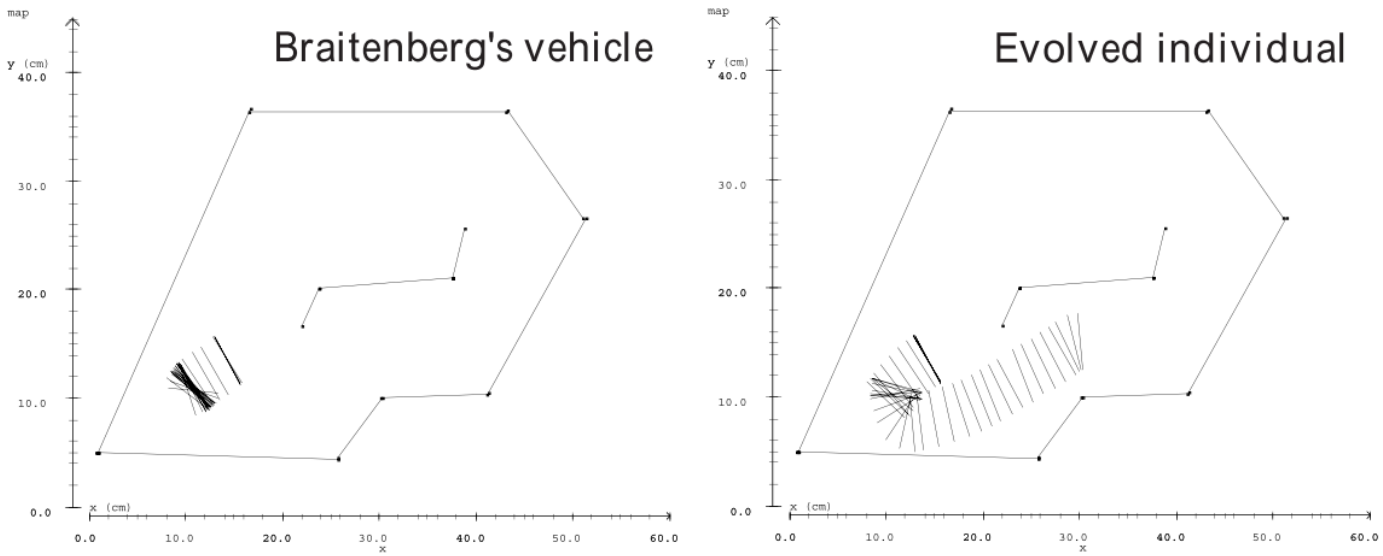


- Premières générations:
  - tournent sur place; foncent dans les murs
- après 20 générations:
  - évite les obstacles; tournent sur place
- après 50 générations:
  - rapide; évitent les obstacles; indépendant du point de départ

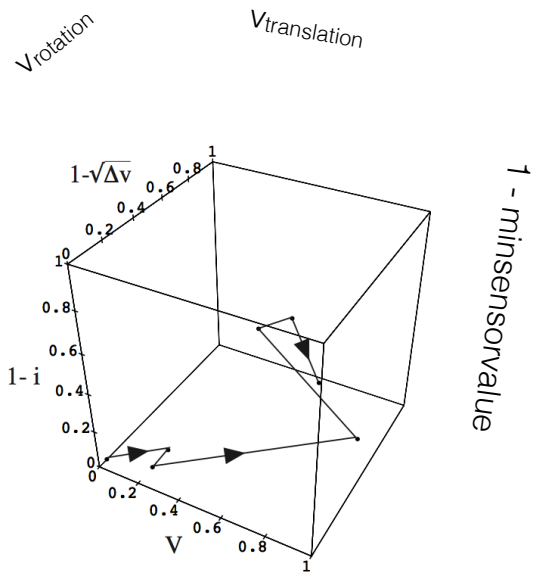


extrait de: [Floreano, Mondada, 1994], légende adaptée

- 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)

Conclusions

- Ce qu'il faut retenir
  - Réseaux de neurones artificiels pour le contrôle
  - Evolution artificielle appliquée à la robotique
- Application dans le cadre du projet
  - Un nouveau formalisme pour la prise de décision d'un agent
  - Exemple d'évolution de comportement

Fin du cours