Induction

Suites

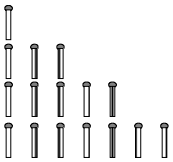
3, 4, 6, 8, 12, 14, 18, 20, 24, 30, 32, 38, 42

$n + 1$, n premier → 44, 48, 54

nombres n tels que pour tout entier k premier avec n et supérieur à k^2, le nombre $n - k^2$ est premier. → 48, 54, 60

Apprentissage par renforcement

- Exemple du jeu de Nim

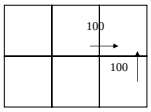


Configuration de départ

Le jacquet (Gerry Tesauro 1992, 1995)

Q-learning

- Apprendre une fonction locale plutôt qu'une stratégie globale



$\pi^*(s) = argmax_a\,(R(s,a) + \gamma V^*(\delta(s,a)))$

$Q(s,a) = R(s,a) + \gamma\ max_{a'} Q(\delta(s,a),a'))$

$\pi^*(s) = argmax_a\,(Q(s,a))$

π = stratégie

s = état

$a = \pi(s)$ = action

$R(s, a)$ = récomp. immédiate

$\delta(s, a))$ = graphe de transition

$V^*(s)$ = gain optimal

Marienbad

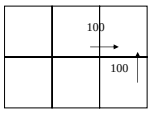
Q-learning

$\pi^*(s) = argmax_a\,(R(s,a) + \gamma V^*(\delta(s,a)))$

$Q(s,a) = R(s,a) + \gamma\, max_{a'} Q(\delta(s,a),a'))$

$\pi^*(s) = argmax_a\,(Q(s,a))$

- `Initialiser la table Q(s,a) à zéro.`
- `Observer l'état courant s.`
- `Répéter`
  - `Choisir une action a et l'exécuter`
  - `Recevoir la récompense r`
  - `Observer le nouvel état s'`
  - `Mettre à jour la table Q(s,a) par Q(s,a) ← r + γ max`$_{a'}$` Q(s',a')`
  - `s ← s'`



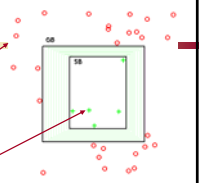
Universal A.I. (M. Hutter)

AIXI is an agent that interacts with an environment in cycles $k=1,2,...,m$. In cycle k, AIXI takes action a_k (e.g. a limb movement) based on past perceptions $o_1\ r_1 ... o_{k-1}\ r_{k-1}$ as defined below. Thereafter, the environment provides a (regular) observation o_k (e.g. a camera image) to AIXI and a real-valued reward r_k. The reward can be very scarce, e.g. just +1 (-1) for winning (losing) a chess game, and 0 at all other times. Then the next cycle $k+1$ starts. Given the above, AIXI is defined by

AIXI $\quad a_k := \arg\max_{a_k} \sum_{o_k r_k} ... \max_{a_m} \sum_{o_m r_m} [r_k + ... + r_m] \sum_{q:U(q,a_1..a_m)=o_1 r_1 .. o_m r_m} 2^{-\ell(q)}$

The expression shows that AIXI tries to maximize its total future reward $r_k+...+r_m$. If the environment is modeled by a deterministic program q, then the future perceptions $...o_k r_k ... o_m r_m = U(q,a_1..a_m)$ can be computed, where U is a universal (monotone Turing) machine executing q given $a_1..a_m$. Since q is unknown, AIXI has to maximize its expected reward, i.e. average $r_k+...+r_m$ over all possible future perceptions created by all possible environments q that are consistent with past perceptions. The simpler an environment, the higher is its a-priori contribution $2^{-l(q)}$, where simplicity is measured by the length l of program q. AIXI effectively learns by eliminating Turing machines q once they become inconsistent with the progressing history. Since noisy environments are just mixtures of deterministic environments, they are automatically included. The sums in the formula constitute the averaging process. Averaging and maximization have to be performed in chronological order, hence the interleaving of max and Σ (similarly to minimax for games).

Version space

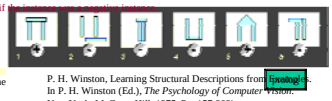


Counter-example

- Specific boundary (lgg)
- General boundary

Positive example

- *represent* the current example;
- *predict* from the representation of the current hypothesis whether or not the current instance exemplifies the concept;
  - if *correct* then *retain* the current hypothesis;
  - if *incorrect* then
    - *identify the differences* between the current example and the current concept;
    - *choose from this set of differences* and use the chosen differences to
      - *generalize* the concept hypothesis if the instance was a positive instance or
      - *specialize* the concept hypothesis if the instance was a negative instance



Only one hypothesis at a time

P. H. Winston, Learning Structural Descriptions from Examples. In P. H. Winston (Ed.), *The Psychology of Computer Vision*. New York: McGraw-Hill, 1975. Pp. 157-209)

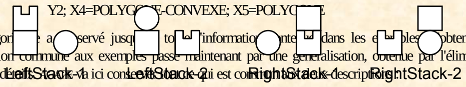
Apprentissage structurel

AGAPE part des descriptions initiales de E1 et E2 :

E1 = (CARRE A) & (CERCLE B) & (AU-DESSUS A B)
E2 = (TRIANGLE C) & (CARRE D) & (AU-DESSUS C D)

AGAPE est obligé de remonter la hiérarchie conceptuelle pour mettre E1 et E2 en correspondance :

E'1 = (POLYGONE-CONVEXE Y1 X1) & (FORME X5 X4 Y2 X2) & (AU-DESSUS X1 X2)
avec les liens : X1=A; X2=B; X1≠X2; X3=CARRE; X3=Y1; Y1=CARRE; Y2=CERCLE; X3≠Y2; Y1≠Y2; X4=ELLIPSOIDE; X5=PATATOIDE
E'2 = (POLYGONE-CONVEXE Y1 X1) & (FORME X5 X4 Y2 X2) & (AU-DESSUS X1 X2)
avec les liens : X1=C; X2=D; X1≠X2; X3=CARRE; Y1=TRIANGLE; X3≠Y1; X3=Y2; Y2=CARRE; Y1≠Y2; X4=POLYGONE-CONVEXE; X5=POLYGONE

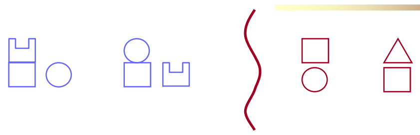


L'algorithme a conservé jusque-là toute l'information contenue dans les exemples. L'obtention d'une description commune aux exemples passe maintenant par une généralisation, obtenue par l'élimination de certains détails. AGAPE va ici conserver tout ce qui est commun aux deux descriptions :

Eg = (POLYGONE-CONVEXE Y1 X1) & (FORME X5 X4 Y2 X2) & (AU-DESSUS X1 X2)
avec les liens : X1≠X2; X3=CARRE; Y1≠Y2;

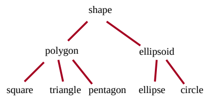
pour donner ce que l'on peut traduire par "il y a deux objets X1 et X2 l'un sur l'autre; X1 est un polygone convexe. Ces objets ont deux formes Y1 et Y2 différentes et il y a un carré X3".

Learning structures



E1 = (SQUARE A) & (CIRCLE B) & (ABOVE A B)

E2 = (TRIANGLE C) & (SQUARE D) & (ABOVE C D)



- Background knowledge

[square(A), circle(B), vPos(A,2), vPos(B,1)]

[triangle(C), square(D), vPos(C,2), vPos(D,1)]

LGG:

[polygon(object_1), form(object_2), vPos(object_1,2), vPos(object_2,1)]

prolog

Inductive Logic Programming

```
cute(X) :- dog(X), small(X), fluffy(X).
cute(X) :- cat(X), fluffy(X).
```

Generalisation: `cute(X) :- fluffy(X).`

- Background knowledge

```
pet(X) :- dog(X).
pet(X) :- cat(X).
small(X) :- cat(X).
```

Generalisation: `cute(X) :- pet(X), small(X), fluffy(X).`

prolog

Inductive Logic Programming

Examples *E* are expected to result from background knowledge *B* and hypothesis *H*.

$B \wedge H \models E$

- Inverse resolution

From example:
`cute(X) :- cat(X), fluffy(X).`
and knowledge:
`pet(X) :- cat(X).`
induce:
`cute(X) :- pet(X), fluffy(X).`

prolog

Explanation Based Generalization

Connaissance du domaine

repose_sur(X,Y) & plat(Y) ==> stable(X)
peut_être_saisi(X) & léger(X) ==> portable(X)
concave(Y) & partie_de(Y,X) & vers_le_haut(Y) ==> récipient(X)
partie_de(Y,X) & forme(Y, demi-tore) & adapté_aux_doigts(Y) & rigide(Y) ==> peut_être_saisi(X)
isolant(Y) & fait_de(X,Y) & petit(X) ==> peut_être_saisi(X)
isolant(céramique)
isolant(porcelaine)
petit(X) ==> léger(X)
diamètre(X) > 2cm & diamètre(X) < 12cm ==> adapté_aux_doigts(X)
stable(X) & portable(X) & récipient(X) ==> tasse(X)

Description de l'exemple

appartient_à(obj1, Fred) & léger(obj1) & couleur(obj1,rouge) & partie_de(h1, obj1)
& rigide(h1) & diamètre(h1) = 3cm & forme(h1,demi_tore) & repose_sur(obj1, b1)
& plat(b1) & concave(c1) & partie_de(c1, obj1) & vers_le_haut(c1)

Démonstration de l'exemple

léger(X) & partie_de(H, X) & rigide(H) & diamètre(H) > 2cm & diamètre(H) < 12cm
& forme(H, demi_tore) & repose_sur(X, B) & plat(B) & concave(C)
& partie_de(C, X) & vers_le_haut(C) ==> tasse(X)

prolog

Explanation Based Learning

Connaissance du domaine

repose_sur(X,Y) & plat(Y) ==> stable(X)
peut_être_saisi(X) & léger(X) ==> portable(X)
concave(Y) & partie_de(Y,X) & vers_le_haut(Y) ==> récipient(X)
partie_de(Y,X) & forme(Y, demi-tore) & adapté_aux_doigts(Y) & rigide(Y) ==> peut_être_saisi(X)
isolant(Y) & fait_de(X,Y) & petit(X) ==> peut_être_saisi(X)
isolant(céramique)
isolant(porcelaine)
petit(X) ==> léger(X)
diamètre(X) > 2cm & diamètre(X) < 12cm ==> adapté_aux_doigts(X)
stable(X) & portable(X) & récipient(X) ==> tasse(X)

G003(H, X) <==> partie_de(H, X) & rigide(H) & diamètre(H) > 2cm & diamètre(H) < 12cm
& forme(H, demi_tore)

G001(X) <==> léger(X) & G003(H, X) & repose_sur(X, B) & plat(B) & concave(C)
& partie_de(C, X) & vers_le_haut(C)

G004(H, X) <==> léger(X) & partie_de(H, X) & rigide(H) & diamètre(H) > 2cm
& diamètre(H) < 12cm & forme(H, demi_tore)