Monte Carlo Tree Search

Alpha-beta : succès et limites

Grand succès permis par alpha-beta

- Jeu d'échec: victoire de Deep Blue sur Kasparov en 1997
- Moteur pour d'autres jeux: Dames, Othello,…

Limites

- Inadapté pour jeu de Go
  - Espace de recherche immense: ~10^{761} (rappel: échecs ~ 10^{120})
  - Pas de bonnes heuristiques (problèmes de « retournements de situation »)
- Connaissances pour un jeu non transférables aux autres
  - Une bonne heuristique aux dames ne sert à rien aux échecs
  - -> il faut repartir de zéro à chaque fois, beaucoup de travail humain…

General Game Playing

- Finalité de l'IA ≠ savoir jouer parfaitement à UN jeu !
- Mieux : savoir bien joueur à TOUS les jeux
  - -> General Game Playing

- General Game Playing
  - L'IA connaît juste les règles du jeu
    - *Pas d'heuristiques faites main par des grands maîtres humains…*
  - Après un éventuel temps d'apprentissage, doit pouvoir joueur au jeu en temps limité
  - Meilleure approche actuelle: **Monte-Carlo Tree Search (MCTS)**

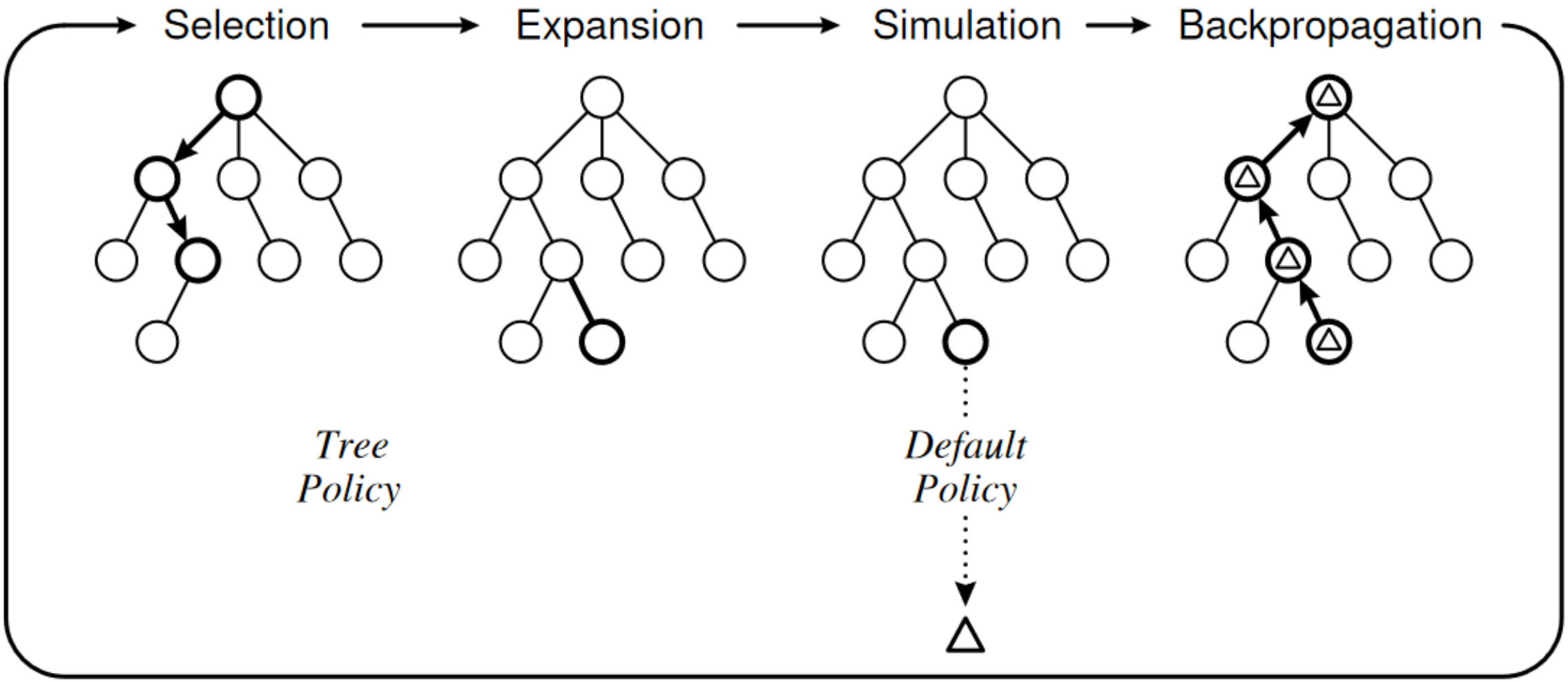
Monte Carlo Tree Search : idée générale

- Exploration d'un arbre de recherche *asymétrique*
  - = on ne développe pas tout comme min-max
  - On a plutôt une approche type « best-first » (*peut faire penser à A**)

- Pour un nœud à développer
  - On simule des **parties aléatoires** jusqu'à la fin, et on compte les victoires
  - On update les parents avec les résultats des parties aléatoires jouées

- Idée derrière les parties aléatoires
  - Il suffit de connaître les règles du jeu
  - Exploitation d'approches statistiques
  - Il est moins dangereux d'être faible que d'être **biaisé** !

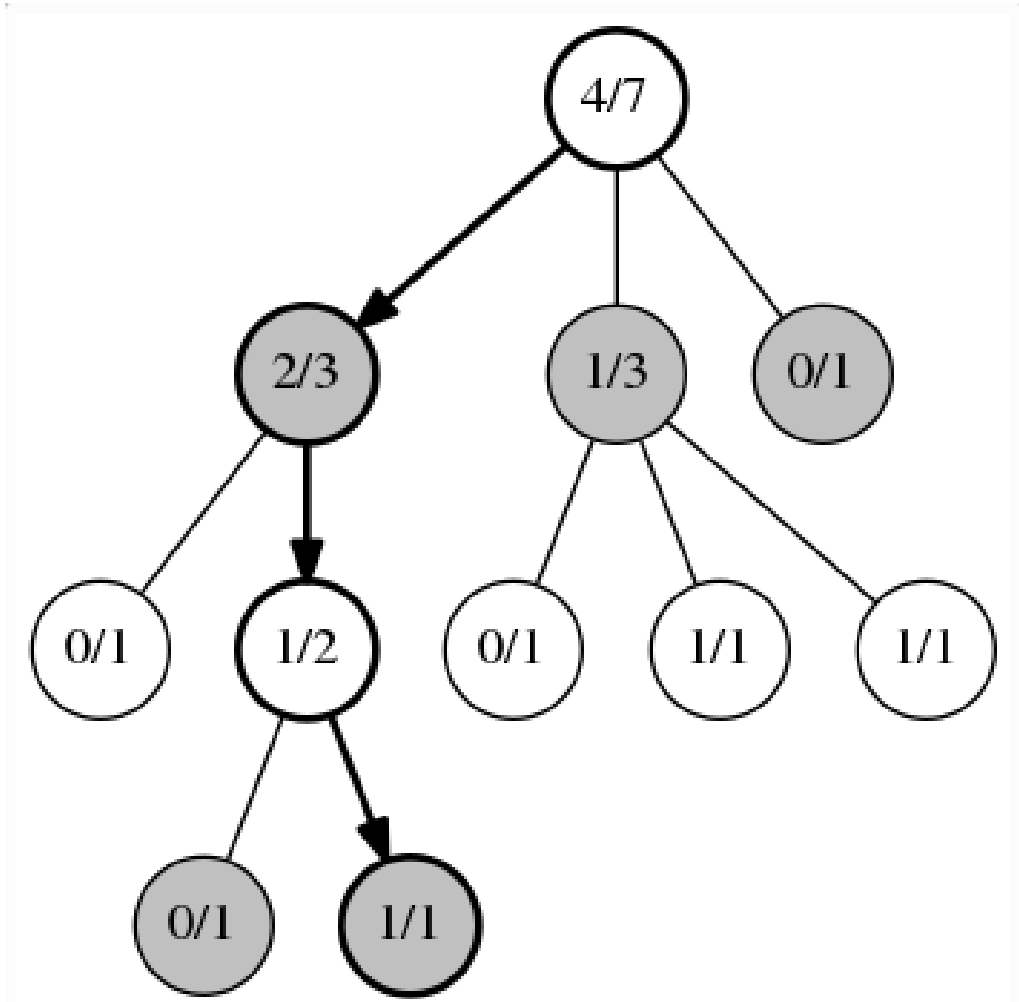
Etapes MCTS


Selection
Expansion
Simulation
Backpropagation
Tree Policy
Default Policy

Etapes MCTS

Sélection

- Partir de la racine
- Utiliser les statistiques des nœuds pour choisir les fils les plus prometteurs
- Stop quand on n'a plus les statistiques de tous les fils

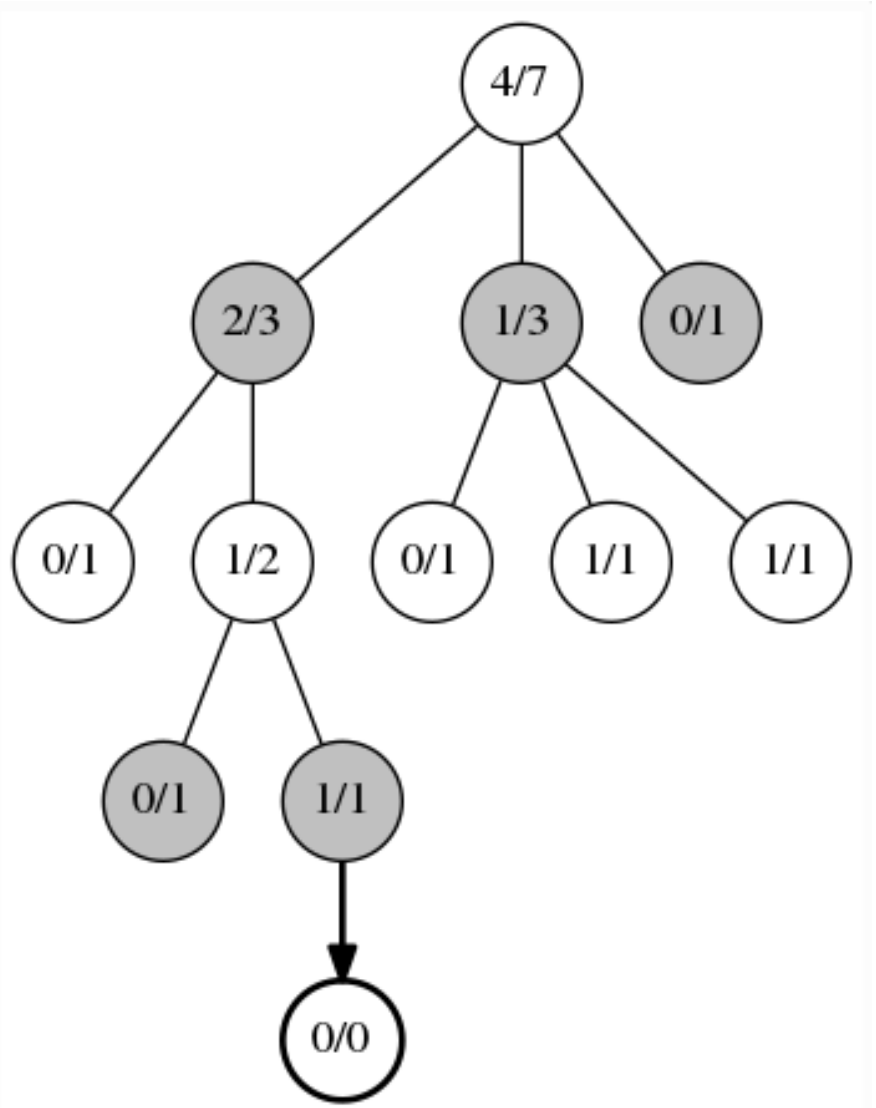


Selection

Etapes MCTS

Expansion

- Pour le nœud sur lequel on s'est arrêté à l'étape Sélection :
  - Créer au hasard un nouveau fils

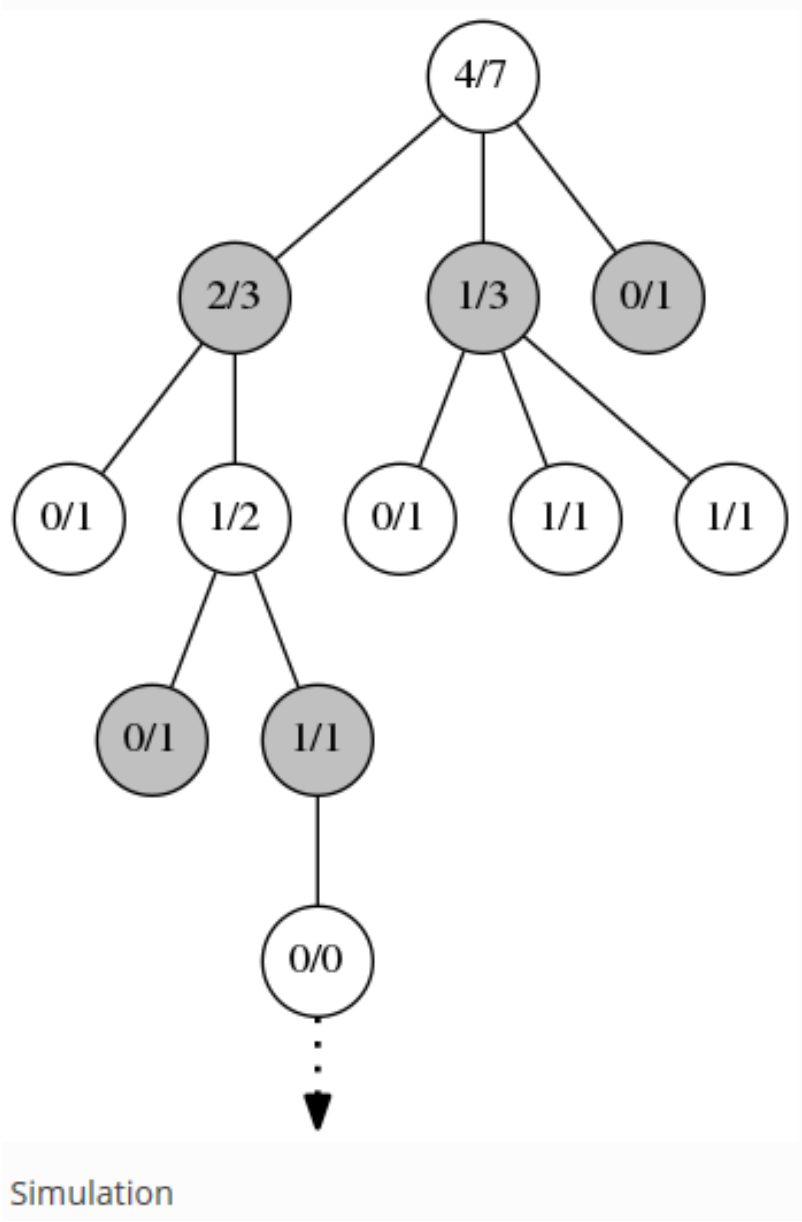


Expansion

Etapes MCTS

Simulation

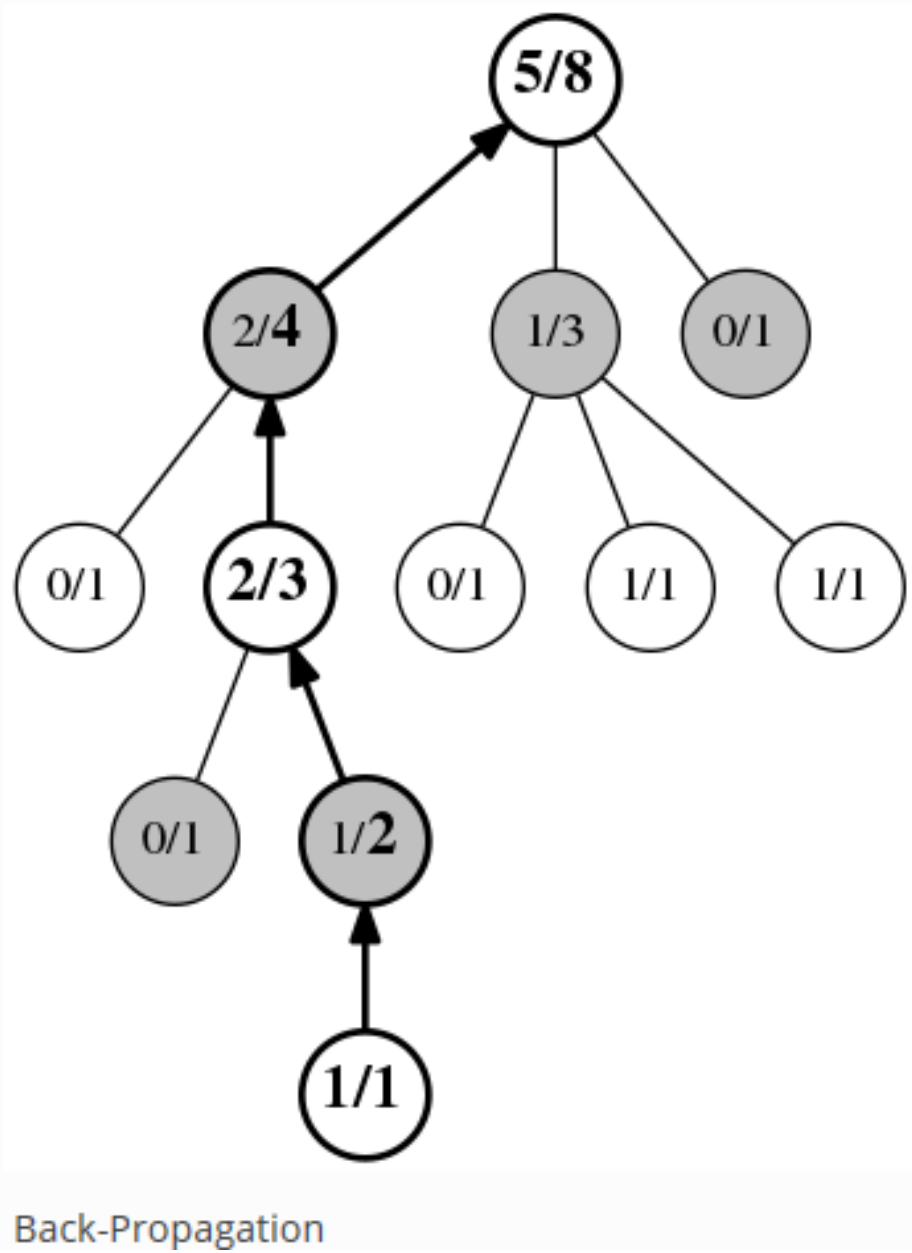
- Simuler une ou plusieurs parties aléatoires en partant du nœud courant
- Aussi appelé phase de **rollout**
- Politique « Default Policy »
  - On respecte les règles du jeu
  - On va jusqu'en fin de partie
  - Jeu peut être random
  - On peut aussi utiliser des heuristiques ici !



Etapes MCTS

Rétropropagation (*backpropagation)*

- Remonter les résultats de la simulation à la racine



Back-Propagation

Pseudo-code MCTS

```
tant que (il reste du temps) {
        visités = new List<Node>()
        node = racine ; visités.add(node)
        tant que (node n'est pas une feuille)
            node = choixFils(node.children) // tree policy pour choisir un fils
            visités.add(node)
        newFils = expand(node) // créer un nouveau fils au hasard
        visités.add(newFils)
        valeur = rollOut(newFils)  // simulation à partir de newFils
        for (node : visités)
            node.updateStats(valeur) // rétropropagation
}
return fils de la racine avec la meilleure valeur
```

Propriétés de MCTS

- Si temps infini, converge vers min-max
  - mais converge très lentement…
- Algorithme **anytime**
  - = on peut l'arrêter quand on veut et avoir un résultat
- Il suffit de connaître les règles du jeu (résultat des parties simulées)

Détails sur la phase de Selection

- Idée que l'on veut trouver les nœuds les plus « urgents » à développer
  - Ex: nœud prometteur mais où l'on manque de simulations

- Problème dit de « bandit »
  - Référence aux *bandits manchots* des casinos
  - Très étudié par les matheux !

Problèmes de bandits

Chance de gagner: *(inconnue du joueur)*	*1/1000*	*1/20*	*1/300*	*1/15*

Problème: a partir des 4 bandits à l'air identiques, comment identifier celui qui a les meilleures chances de gain, en un nombre minimal d'essais ?

Dilemme exploration / exploitation

- On trouve une machine avec une chance de gagner raisonnable
  - Ex: la machine 1/20

- On a deux choix :

  - **Exploitation**: jouer au maximum sur cette machine pour engranger des gains

  - **Exploration**: jouer aussi sur d'autres machines, au cas où il y en a une meilleure
    - *Ici cela permettra de trouver la machine 1/15*
    - *Mais on va perdre des essais (et des gains) dans cette exploration*

Comment gagner à Las Vegas

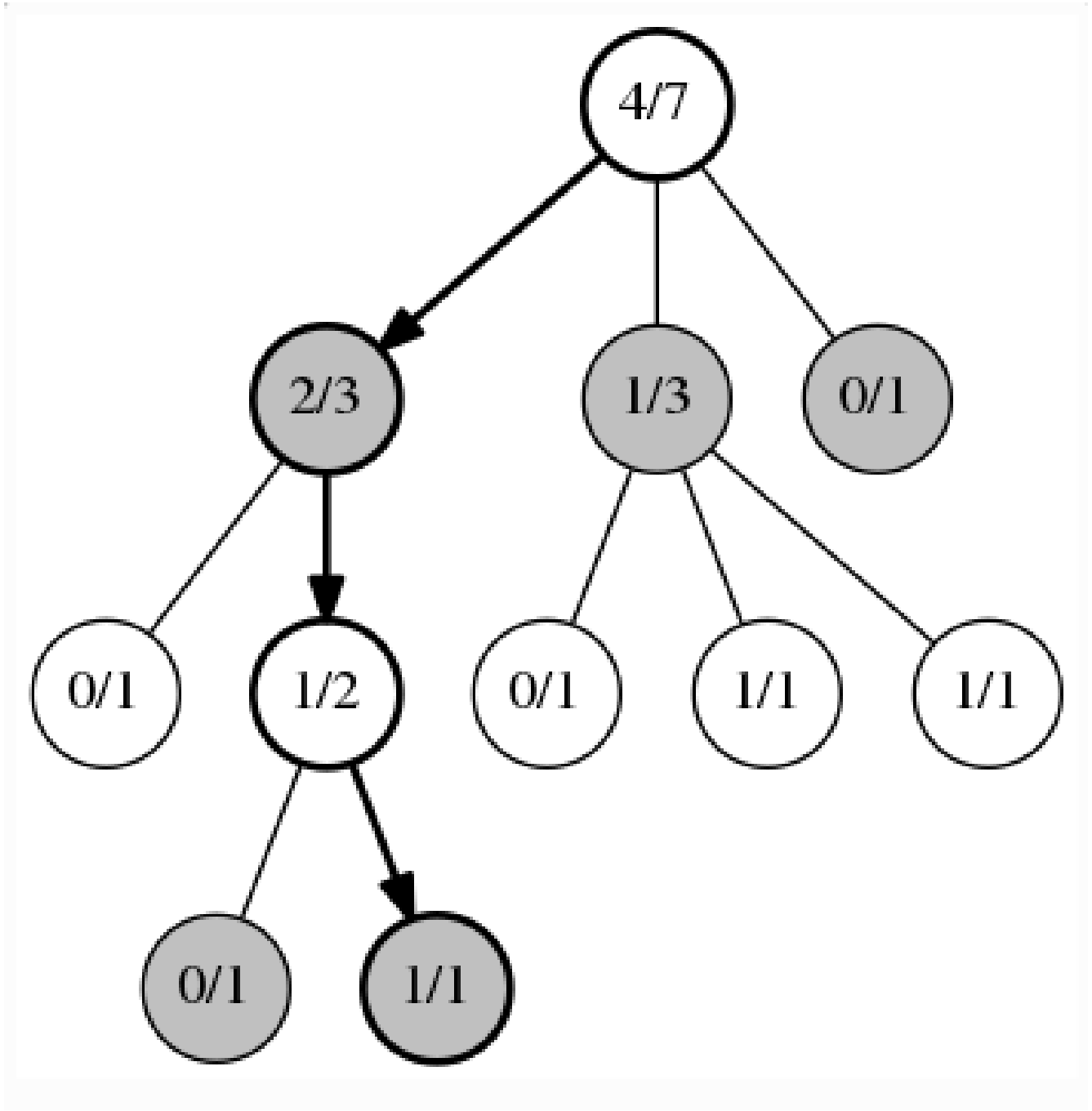
- Stratégie UCB1 : Upper Confidence Bound
- Jouer la machine *i* qui maximise :

- Augmentation la plus faible (O(ln n)) du *regret*
  - Exploit: x_i s'améliore, intervalle de confiance précis pour *i* mais grandit pour les autres -> à un moment une autre machine sera meilleure -> explore

Application à MCTS

- UCT = formule d'UCB adaptée au cas MCTS
- On veut choisir un fils i qui maximise :

- Si i jamais visité, $n_i = 0$ -> $UCT_i = +\infty$
  - Garantit que tout fils sera visité au moins une fois
- Augmenter la constante *C* fera faire plus d'exploration (en pratique : $1/\sqrt{2}$)


4/7
2/3
1/3
0/1
0/1
1/2
0/1
1/1
1/1
0/1
1/1

Improving the roll-out policy π

π_0	Put stones uniformly in empty positions	
π_{random}	Put stones uniformly in the neighborhood of a previous stone	
π_{MoGo}	Put stones matching patterns	prior knowledge
π_{RLGO}	Put stones optimizing a value function	Silver et al. 07

Beware! Gelly Silver 07

$$\pi \text{ better } \pi' \quad \not\Rightarrow \quad MCTS(\pi) \text{ better } MCTS(\pi')$$

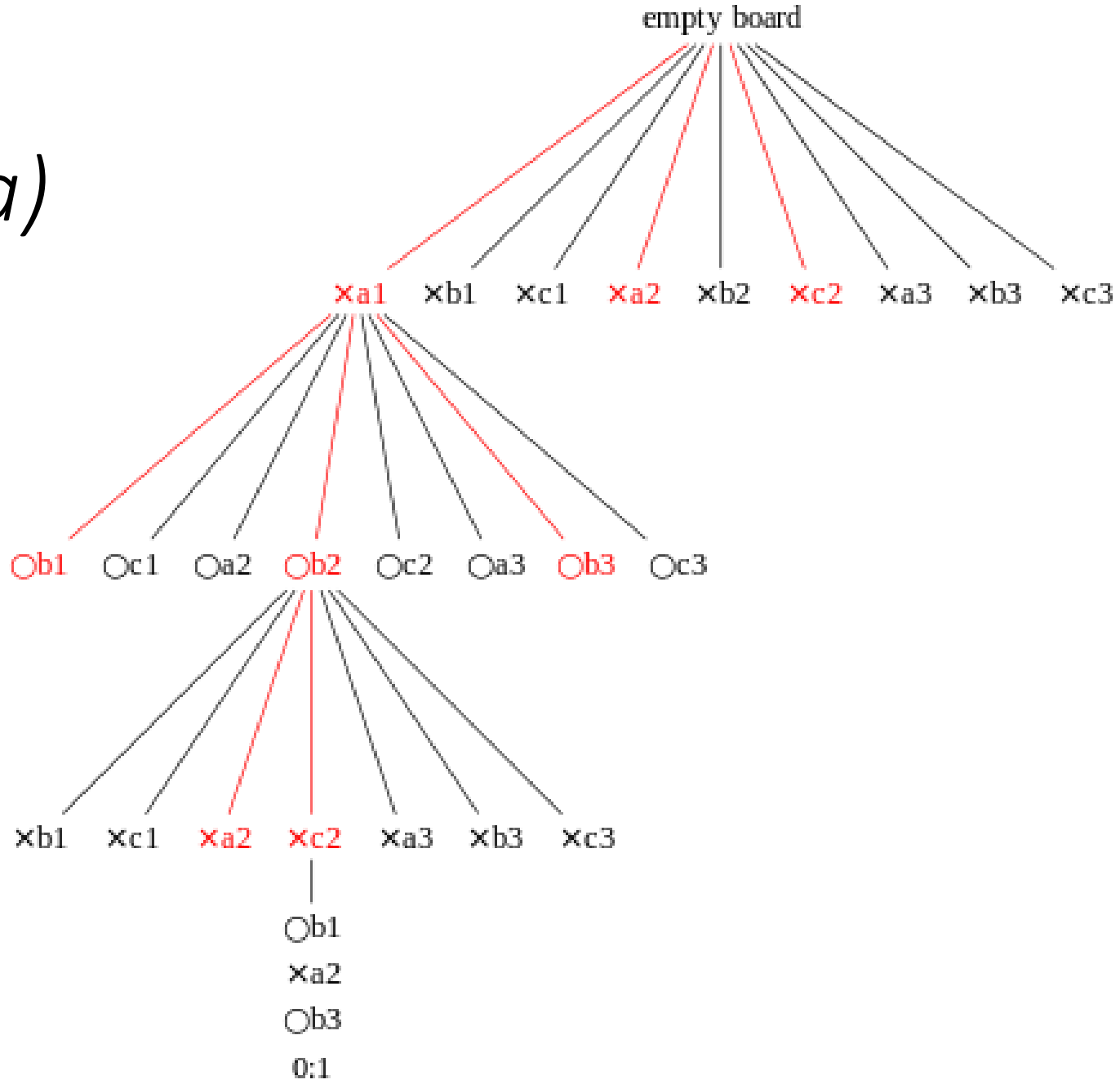
Slide from M. Sebag, CP 2012

RAVE – Rapid Action Value Estimate

- Au début, exploration de MCTS très aléatoire
- Pourtant il explore beaucoup de nœuds: comment l'exploiter ?

- Dans certains jeux (go), ordre des mouvements n'est pas toujours important – i.e., plusieurs façons d'arriver à la meilleure position
  - Pour une simulation, stocker ses statistiques dans tous les nœuds de l'arbre contenant des mouvements explorés par la simulation
  - Modification légère de la formule d'UCT pour prendre en compte ces statistiques supplémentaires

Ex. RAVE sur Tic-Tac-Toe (*wikipedia)*



Tous les nœuds modifiés par la simulation du bas indiqués en rouge

MCTS + Go: historique

1990	Abramson demonstrates that Monte Carlo simulations can be used to evaluate value of state [1].
1993	Brügmann [31] applies Monte Carlo methods to the field of computer Go.
1998	Ginsberg's GIB program competes with expert Bridge players.
1998	MAVEN defeats the world scrabble champion [199].
2002	Auer et al. [13] propose UCB1 for multi-armed bandit, laying the theoretical foundation for UCT.
2006	Coulom [70] describes Monte Carlo evaluations for tree-based search, coining the term Monte Carlo tree search.
2006	Kocsis and Szepesvari [119] associate UCB with tree-based search to give the UCT algorithm.
2006	Gelly et al. [96] apply UCT to computer Go with remarkable success, with their program MoGo.
2006	Chaslot et al. describe MCTS as a broader framework for game AI [52] and general domains [54].
2007	CadiaPlayer becomes world champion General Game Player [83].
2008	MoGo achieves *dan* (master) level at 9×9 Go [128].
2009	Fuego beats top human professional at 9×9 Go [81].
2009	MoHex becomes world champion Hex player [7].

Octobre 2015 : AlphaGo bat Fan Sui, champion européen, 2ème dan (sur 9)

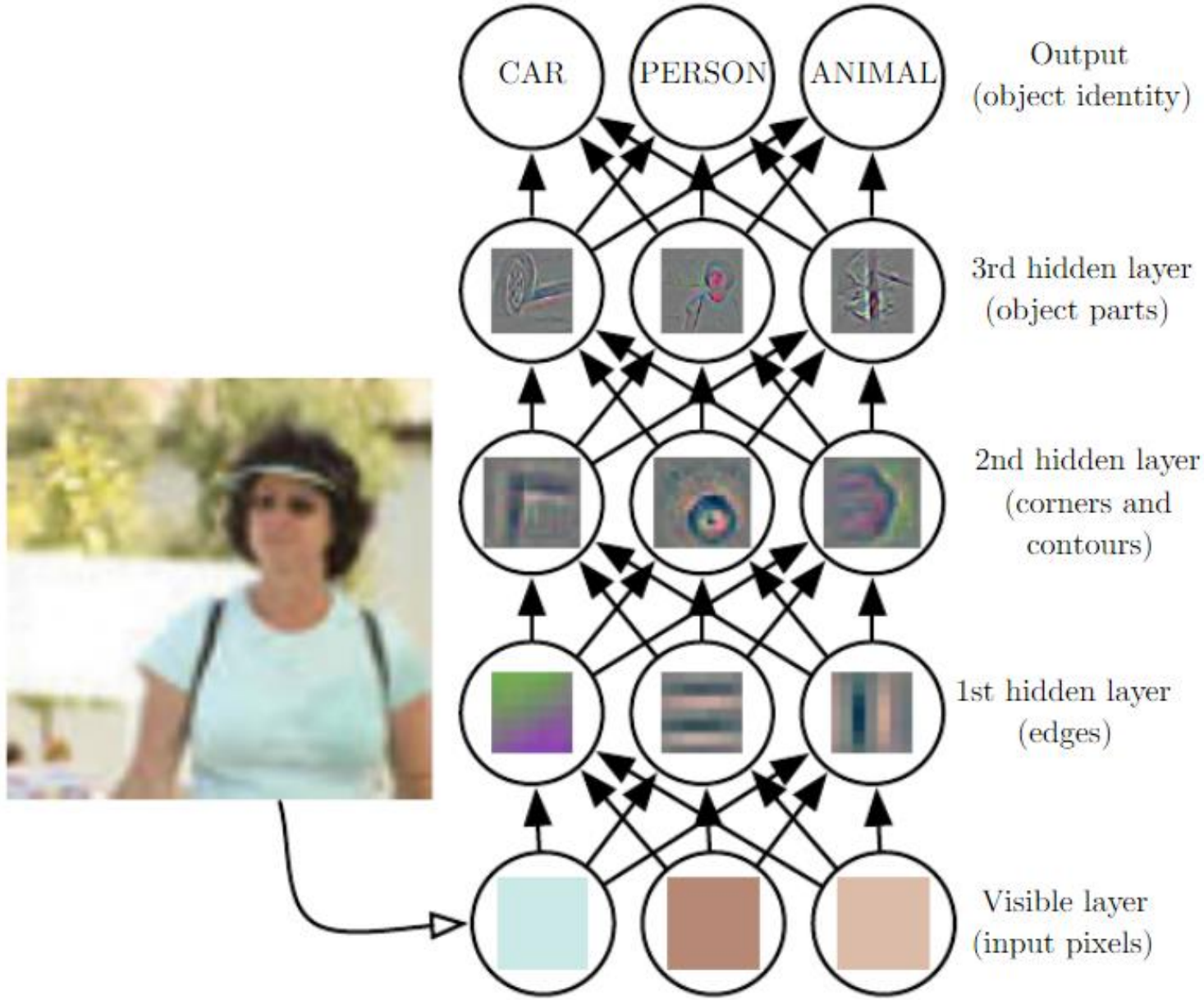
Mars 2016 : AlphaGo bat Lee Sedol, champion mondial, 9ème dan

AlphaGo

- AlphaGo = MCTS + Deep Learning

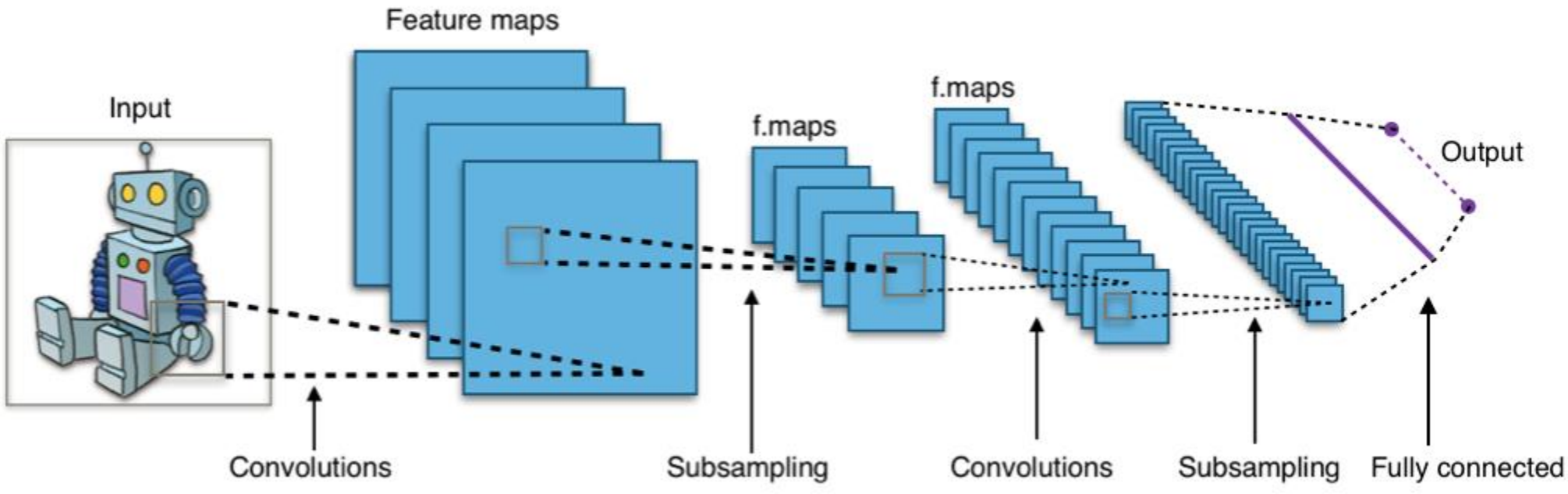
- Deep Learning ?
  - Technique de machine learning
    - Apprentissage sur de nombreux exemples (*supervised learning*)
    - Basé sur des « neurones artificiels »
  - Deep : nombreuses couches, beaucoup de neurones
    - Pas possible avant 2010 car gros besoin en calcul

http://www.deeplearningbook.org/contents/intro.html



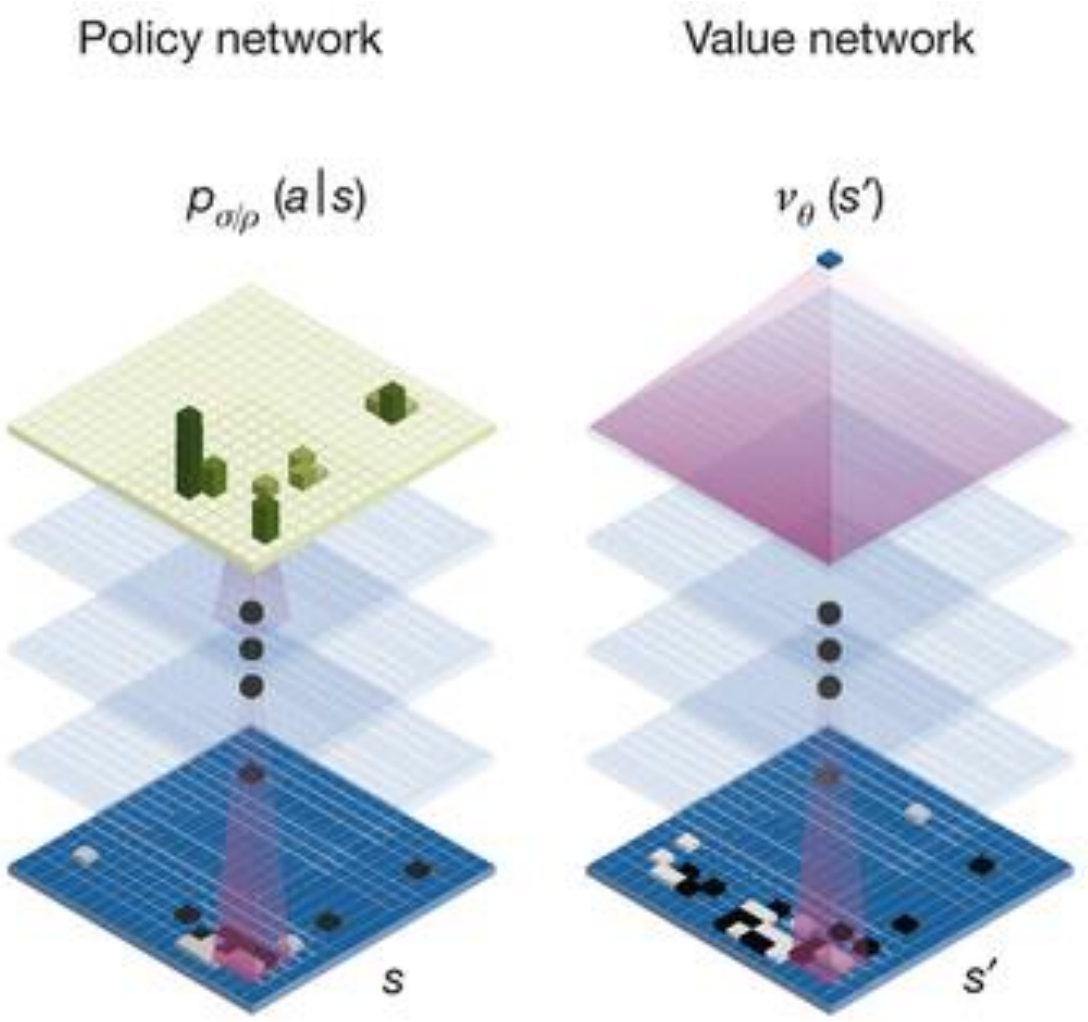
Convolutional network

- Types de réseaux de neurones adaptés pour les images
- Filtres successifs
- Utilisés dans AlphaGo avec goban = image d'input



Les réseaux profonds d'AlphaGo

- Trois réseaux appris par Deep Learning :
  - Policy network (un rapide + un lent)
    - Proba de victoire pour chaque mouvement possible
  - Value network
    - Valeur d'un goban = estimation de la proba de victoire pour les noirs



Entrainement des réseaux

Policy network

- A partir d'historiques de parties d'experts humains
  - KGS Go Server -> 30 millions de coups
  - Prédiction des coups humains avec une précision de 57%
  - Bon à prédire les humains…mais ça ne suffit pas pour gagner !
- Apprentissage par renforcement
  - Différentes versions du policy network se battent entre elles (version t-1 versus version t)
  - Optimise pour le but de gagner le jeu

Value network

- Apprend la proba de gagner à partir des données d'apprentissage du policy network contre lui même

MCTS dans AlphaGo

- AlphaGo combine les réseaux profonds et le principe MCTS:
    - Policy network lent (mais précis) utilisé pour le guidage de MCTS
    - Valeur d'un état = output value network + simulation

La simulation n'est pas random: utilise version rapide du policy network pour simuler des jeux

Sources

- https://en.wikipedia.org/wiki/Monte_Carlo_tree_search
- https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/
- http://www.cameronius.com/cv/mcts-survey-master.pdf
- https://en.wikipedia.org/wiki/AlphaGo
- www.nature.com/nature/journal/v529/n7587/full/nature16961.html
- https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/
- http://scalab.uc3m.es/~seminarios/seminar11/slides/lucas2.pdf
- https://www.cs.swarthmore.edu/~bryce/cs63/s16/slides/2-15_MCTS.pdf
- http://www.deeplearningbook.org/
- https://www.lri.fr/~sebag/Slides/InvitedTutorial_CP12.pdf