INF623

2024/1

UFV

Inteligência Artificial

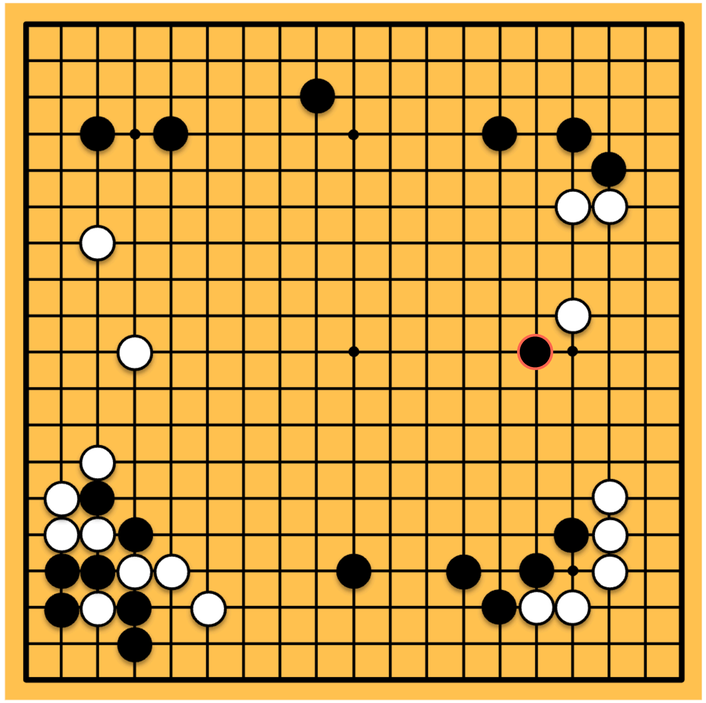
A8: Busca competitiva III

Plano de aula

- O jogo do Go
- Rollouts
- Estados promissores e incerteza
- Heurística UCB (upper confidence bound)
- Busca em árvore monte carlo (MCTS)
- AlphaGo e AlphaGo Zero

Exemplo 1: Go

Considere o tabuleiro de Go abaixo. Qual é a melhor jogada que você pode fazer?



- Fator de ramificação do Go: $b > 300, m \approx 150$
- Fator de ramificação do Xadrez: $b \approx 35, m \approx 80$
- Poda alpha-beta permite uma busca de no máximo 4 níveis de profundidade
- Não é suficiente!

"Move 37", AlphaGo

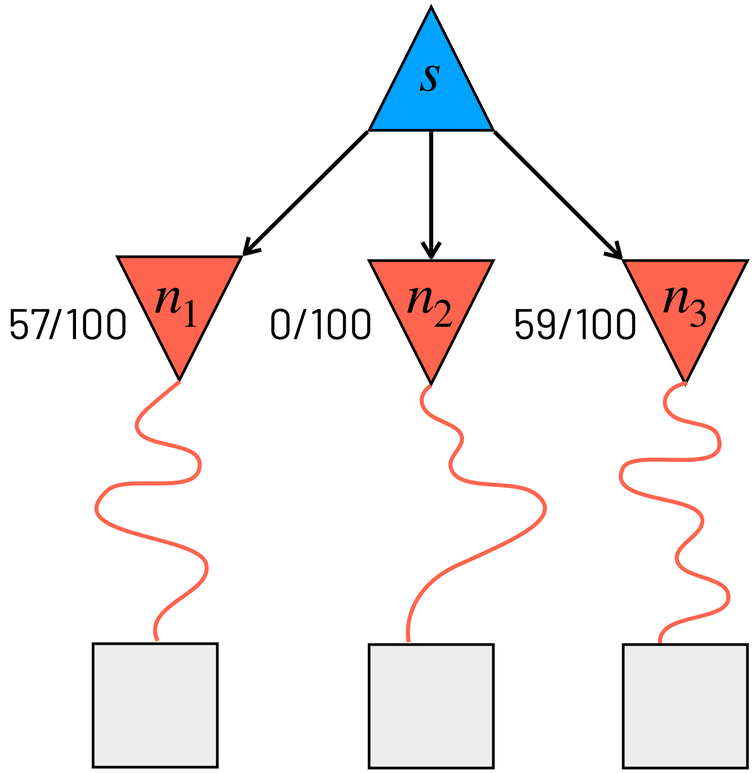
https://blog.google/technology/ai/what-we-learned-in-seoul-with-alphago/

UFV

Busca em árvore monte carlo: ideia geral

A busca em árvore monte carlo combina duas ideais principais:

- **Simulação (*rollout*)**
  - Jogar múltiplas partidas a partir de um estado s usando uma **estratégia rápida**
  - Contar o número de vitórias
- **Seleção (busca seletiva)**
  - Explorar as partes da árvore que ajudam a melhorar a decisão na raiz
  - Independente da profundidade



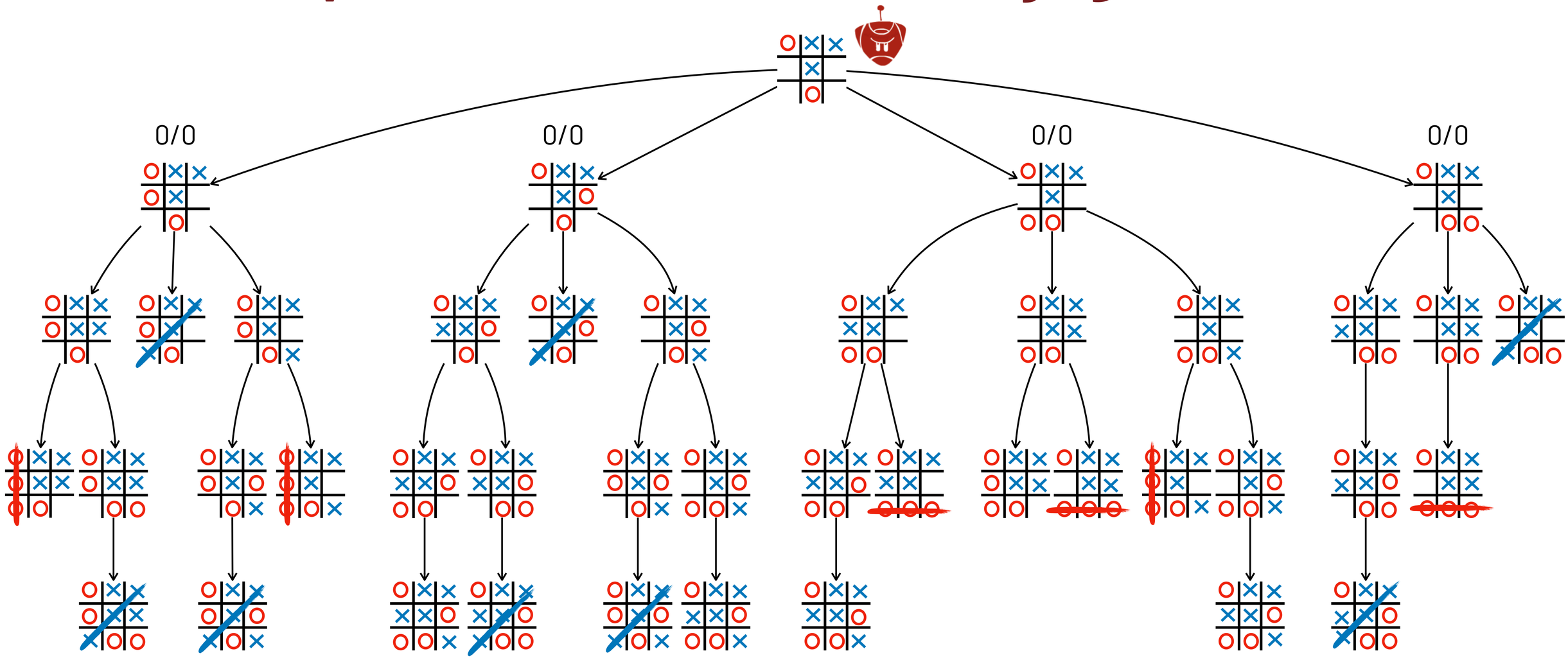
Rollouts

Jogar múltiplas partidas a partir de um estado s usando uma estratégia rápida π e contar o número de vitórias e derrotas:

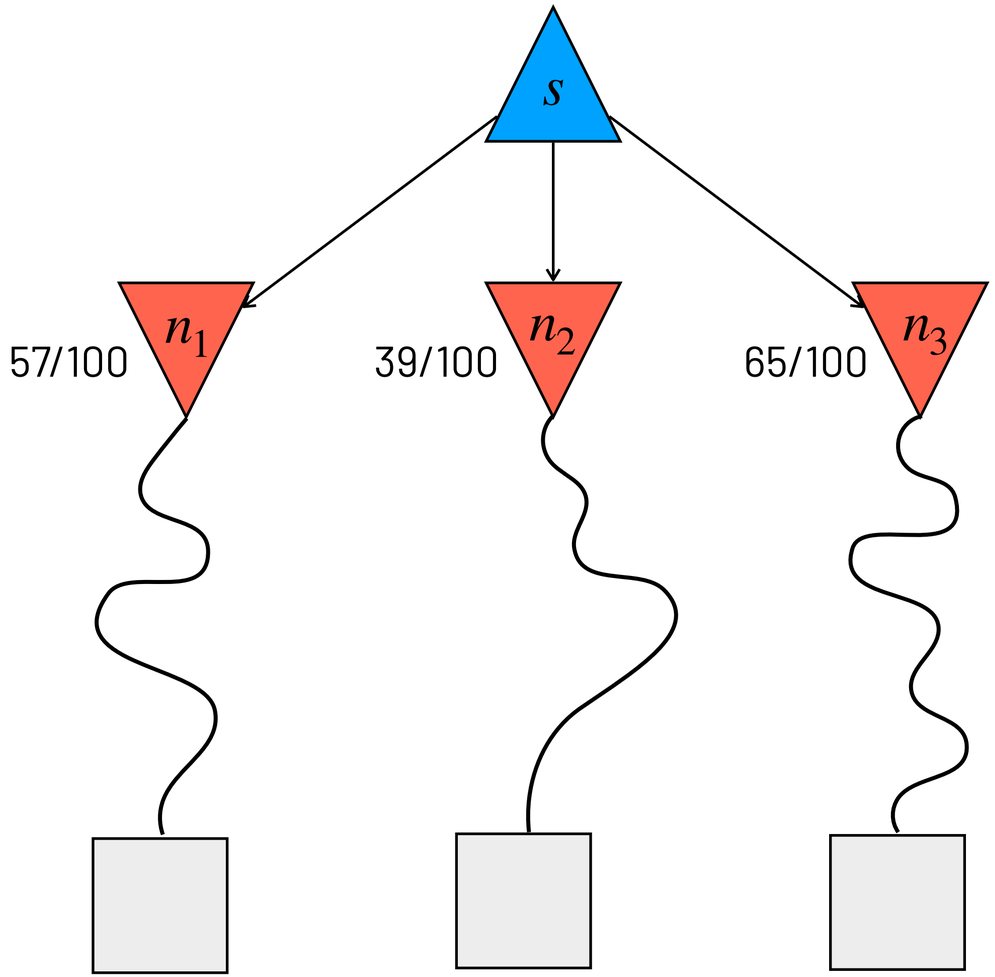
```
def rollout(s, π, E, U, p):
1. r = s
2. while E(r)!= True:
3.     r = π(r)
4. return U(r, p)
```

- Uma estratégia rápida π bastante comum consiste em escolher jogadas aleatórias
  - Melhores estratégias ajudam, mas não devem ser muito custosas (pois tornam o rollout muito demorado)
- Taxa de vitórias a partir de s está correlacionada com o valor $V(s)$

Exemplo: rollout aleatório no jogo da velha



MCTS versão preliminar 1

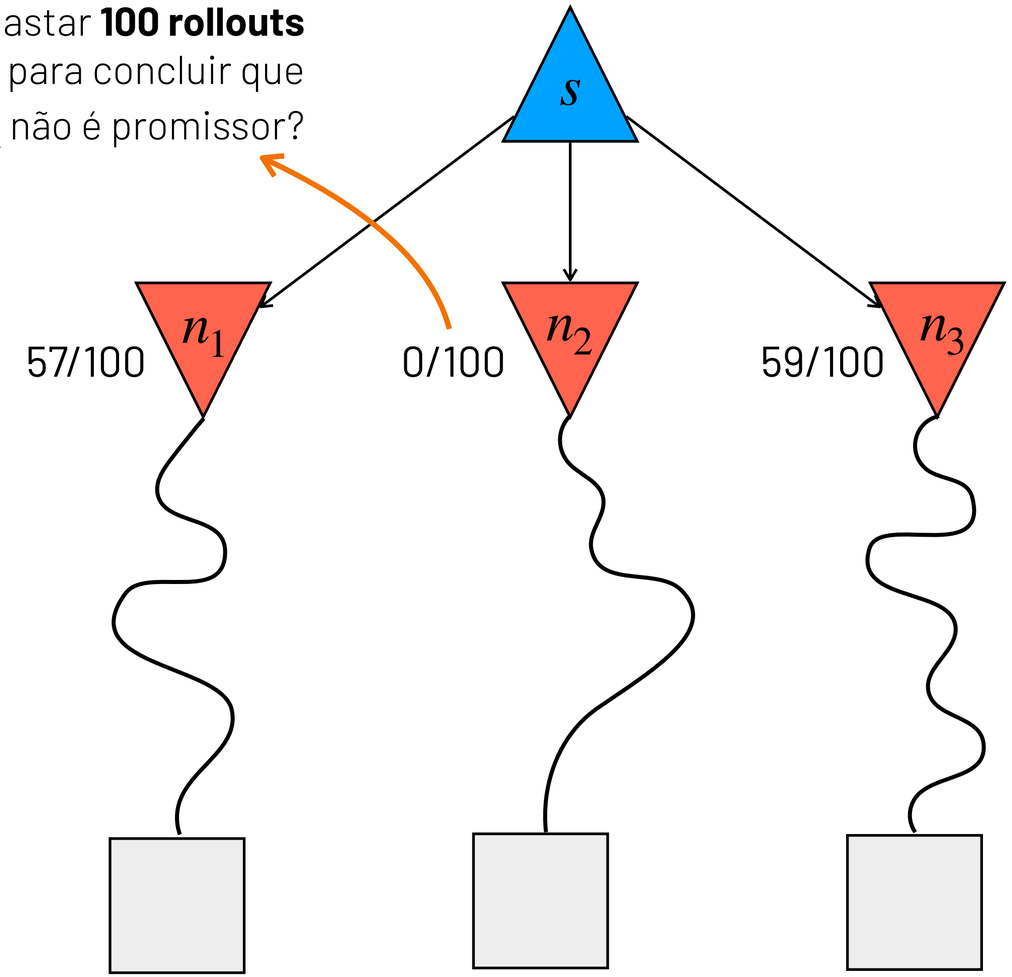


1. Execute N rollouts para cada **filho** do nó **raiz** e guarde as taxas de vitórias
2. Escolha a jogada com a melhor taxa de vitória

UFV

MCTS versão preliminar 1

Precisamos gastar **100 rollouts** com derrotas para concluir que n_2 não é promissor?

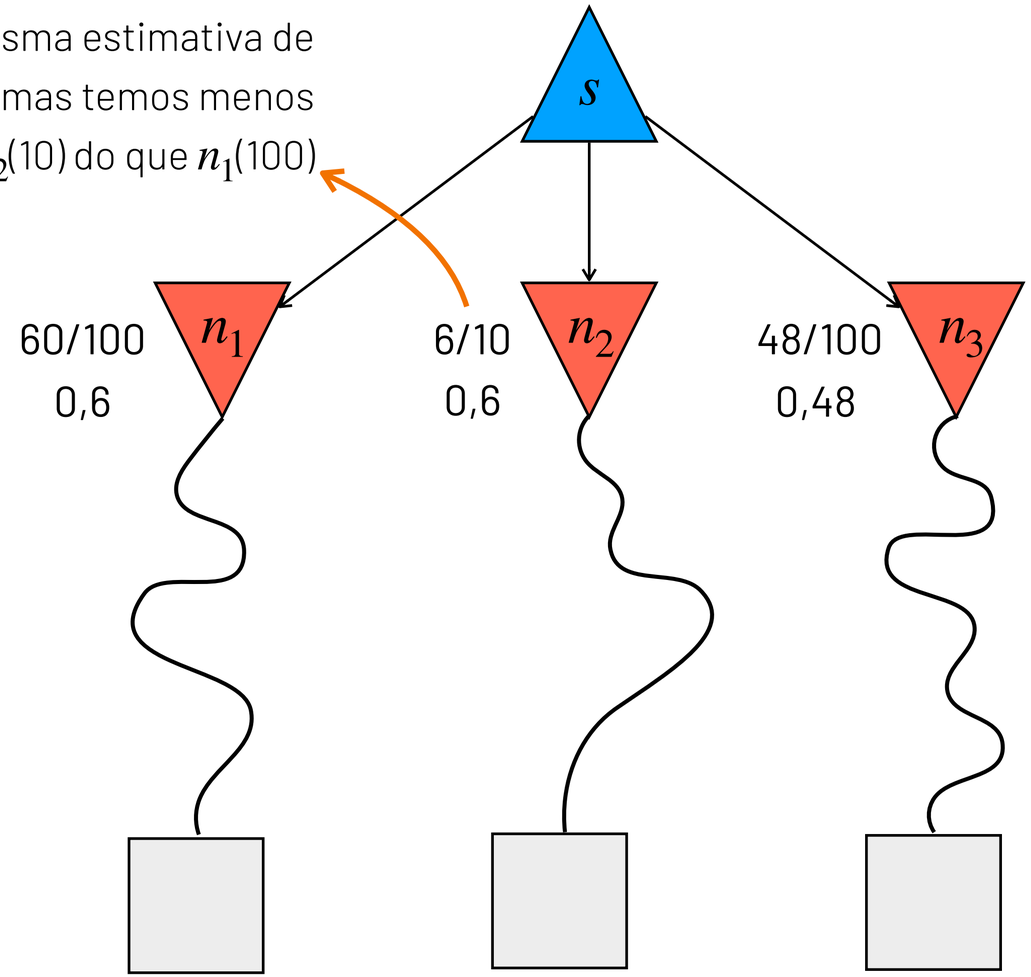


1. Execute N rollouts para cada **filho** do nó **raiz** e guarde as taxas de vitórias
2. Escolha a jogada com a melhor taxa de vitória

UFV

MCTS versão preliminar 2

n_2 tem a mesma estimativa de valor que n_1, mas temos menos certeza em $n_2(10)$ do que $n_1(100)$



1. Execute N rollouts, alocando-os aos **filhos** mais promissores do nó **raiz**, e guarde as taxas de vitórias
2. Escolha a jogada com a melhor taxa de vitória

Como balancear estados promissores considerando incerteza? 🤔

Heurística UCB

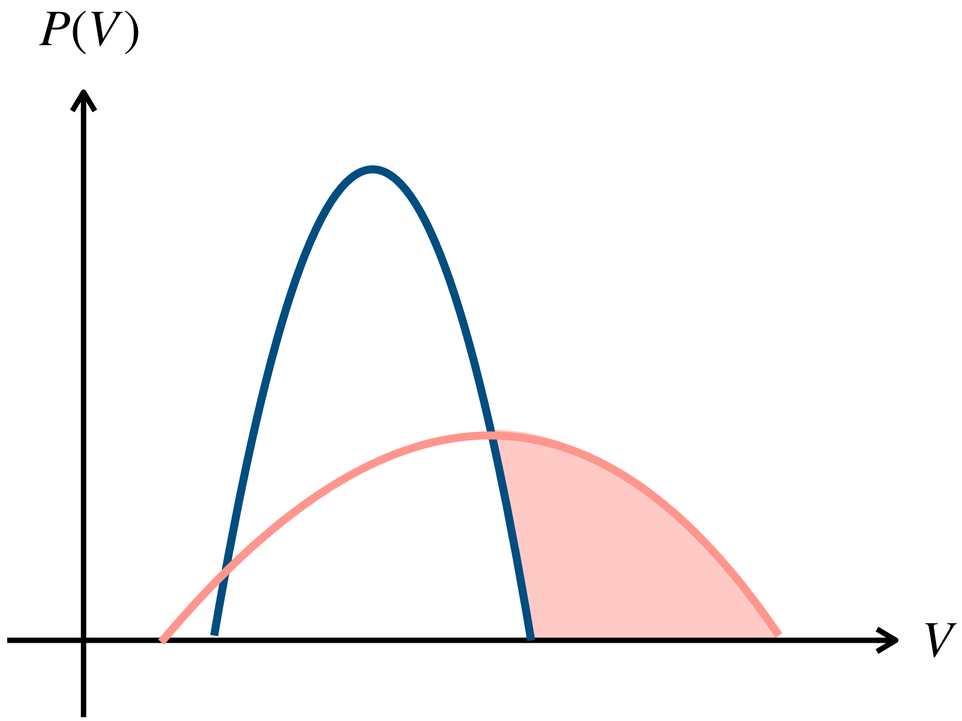
A fórmula UCB (Upper confidence bound) formaliza o trade-off entre nós promissores e incerteza:

$$UCB(s) = \frac{U(s)}{N(s)} + C \times \sqrt{(\frac{log\ N(Pai(s))}{N(s)})}$$

Promissor (intensificação)

Incerteza (exploração)

- $U(s)$ – utilidade acumulada pelo estado s nos rollouts
- $N(s)$ – número de simulações feitas no estado s
- C – constante de exploração
- $Pai(s)$ – pai de s

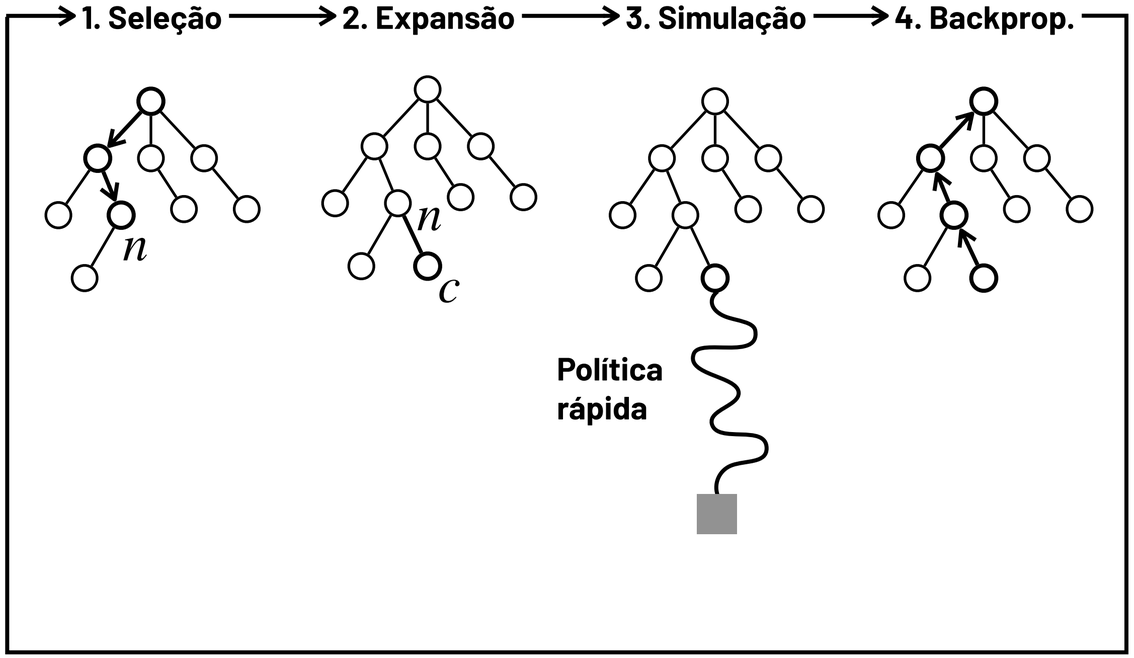


Busca em árvore monte carlo

- Repita até acabar o tempo:

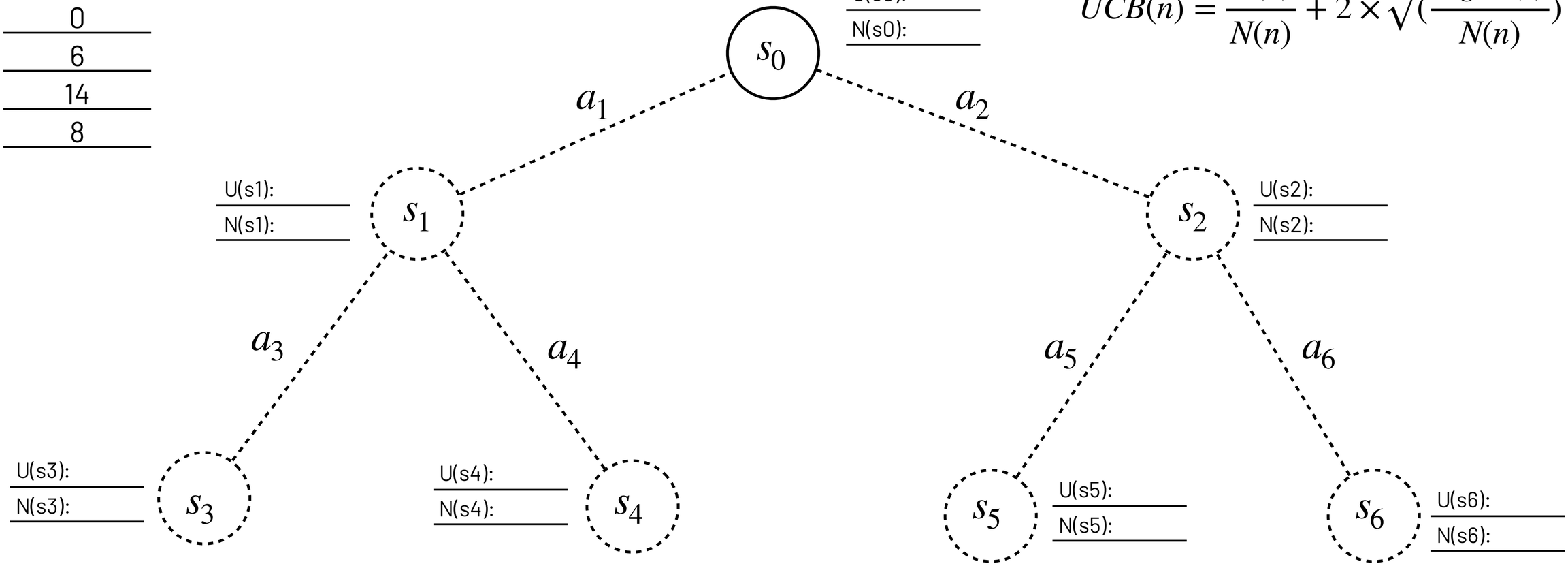
  1. **Seleção**
     (Upper confidence bounds for trees — UCT)
     Aplique recursivamente a *UCB* para escolher um um nó folha n não totalmente expandido
  2. **Expansão**
     Adicione um novo filho c a n
  3. **Simulação (rollout)**
     Execute uma simulação a partir de c
  4. **Backpropagation**
     Atualize as estatísticas de c até a raiz

- Escolha a ação com maior $N(n)$



Exemplo MCTS

Rollouts
20
10
0
6
14
8



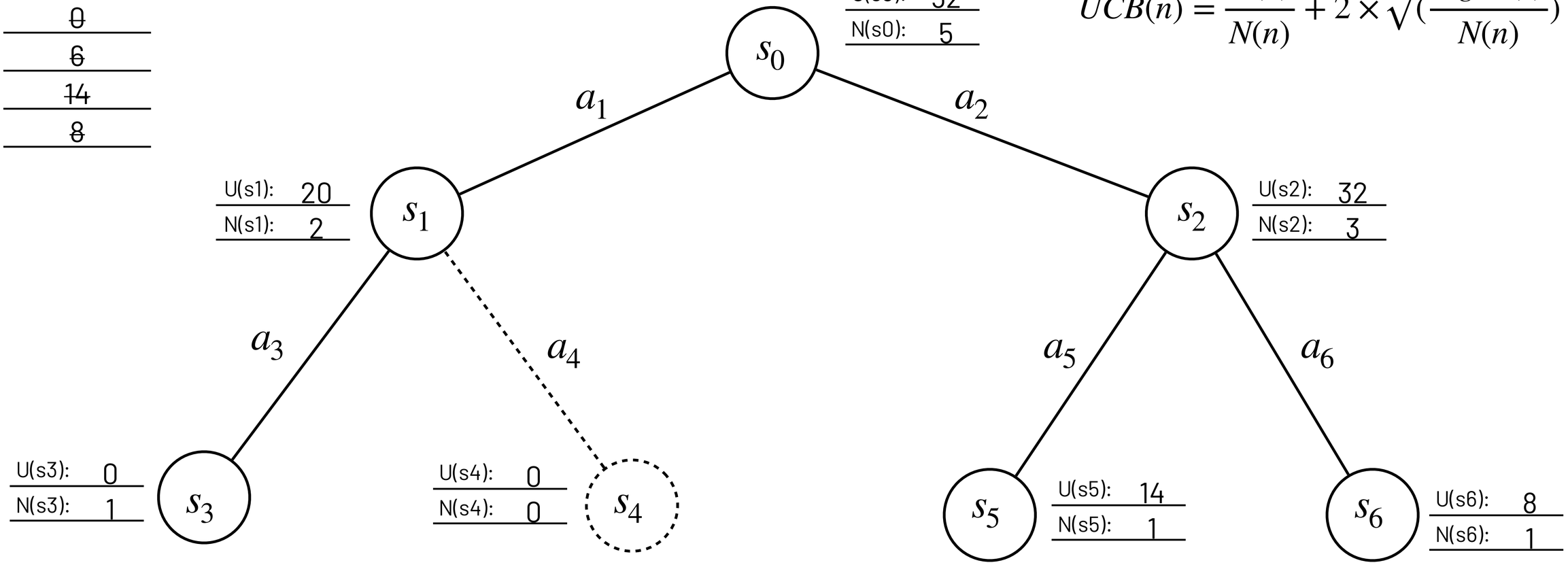
$$UCB(n) = \frac{U(n)}{N(n)} + 2 \times \sqrt{(\frac{log\ N(s)}{N(n)})}$$

https://vgarciasc.github.io/mcts-viz/

Exemplo MCTS

Rollouts
~~20~~
~~10~~
~~0~~
~~6~~
~~14~~
~~8~~

$$UCB(n) = \frac{U(n)}{N(n)} + 2 \times \sqrt{(\frac{log\ N(s)}{N(n)})}$$

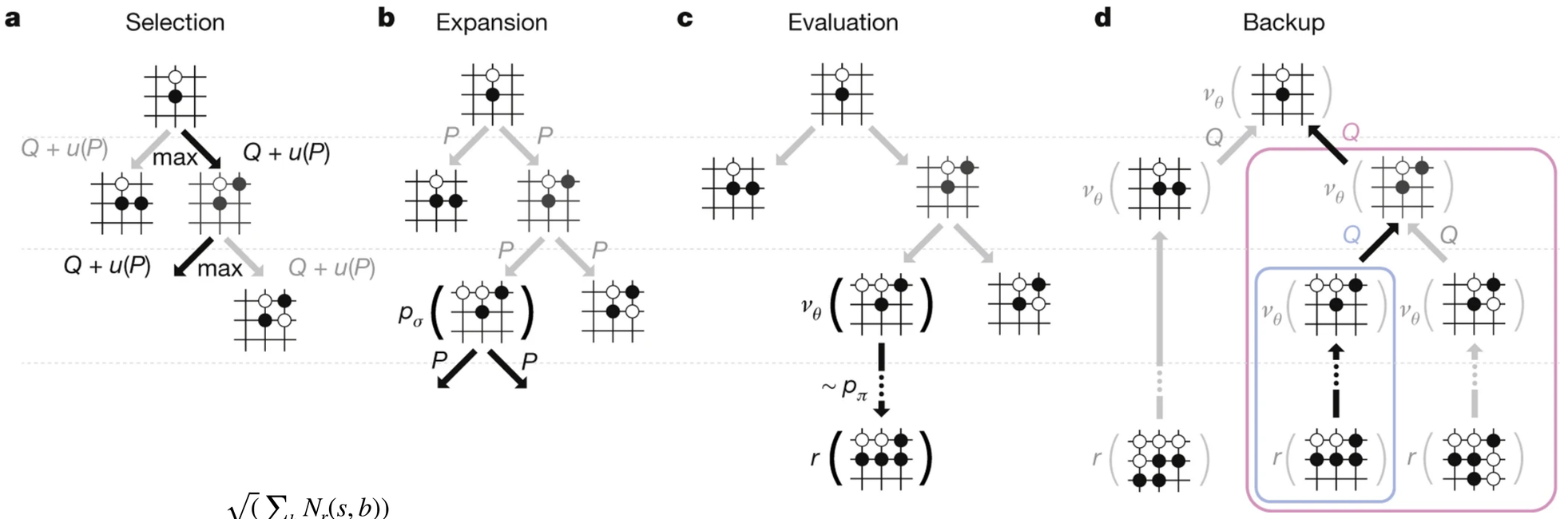


https://vgarciasc.github.io/mcts-viz/

AlphaGo

Trailer do AlphaGo Movie
https://www.youtube.com/watch?v=8tq1C8spV_g&ab_channel=AlphaGoMovie

AlphaGo



Seleção

$$PUCT(s,a) = Q(s_t,a) + c_{puct}P(s,a)\frac{\sqrt{(\sum_b N_r(s,b))}}{1+N_r(s,a)}$$

Expansão

Adiciona um filho s_l ao nó selecionado e calcula as probabilidades das ações com uma **rede neural de política lenta** $p_\sigma(s_l)$

Simulação

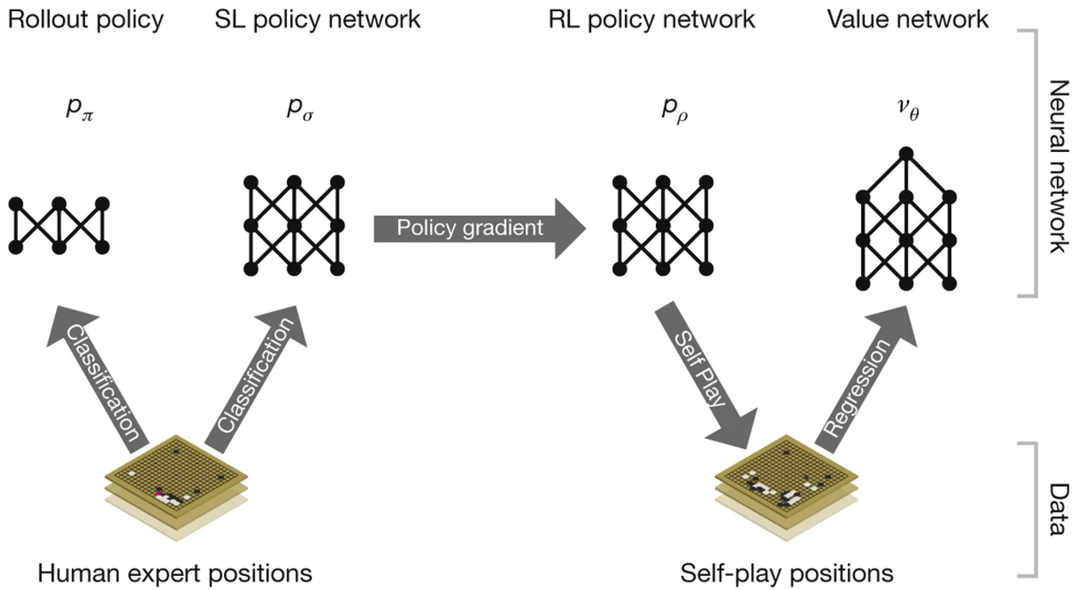
Estima o valor de s_l com uma média ponderada: $V(s_l) = (1-\lambda)v_\sigma + \lambda z_{L'}$ onde v_θ é uma **rede neural de valor** e Z_L é o resultado de um rollout com uma **rede neural de política rápida** p_π.

Backpropagation

Atualização dos valores das ações Q

https://www.nature.com/articles/nature16961

AlphaGo



1. A **rede neural de política rápida** p_π e a **rede neural de política lenta** p_σ são treinadas para prever a próxima jogada de jogadores profissionais usando um dataset de posição-jogada;

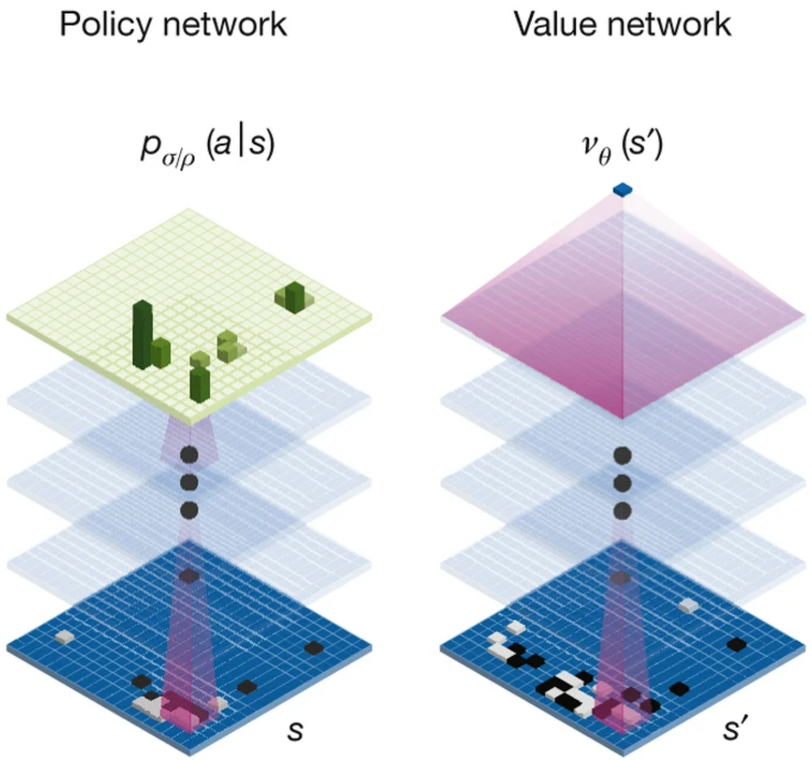
2. Uma nova rede neural de política p_ρ é inicializada com $p_{\sigma'}$ e é melhorada via aprendizado por reforço, jogando com versões anteriores de si mesma escolhidas aleatoriamente;

3. Um novo conjunto de dados é gerado jogando vários jogos de p_ρ contra si mesma (*self-play*);

4. A **rede neural de valor** v_θ é treinada por regressão para prever o resultado da partida (isto é, se o jogador corrente irá ganhar) usando o dataset de *self-play*.

https://www.nature.com/articles/nature16961

AlphaGo

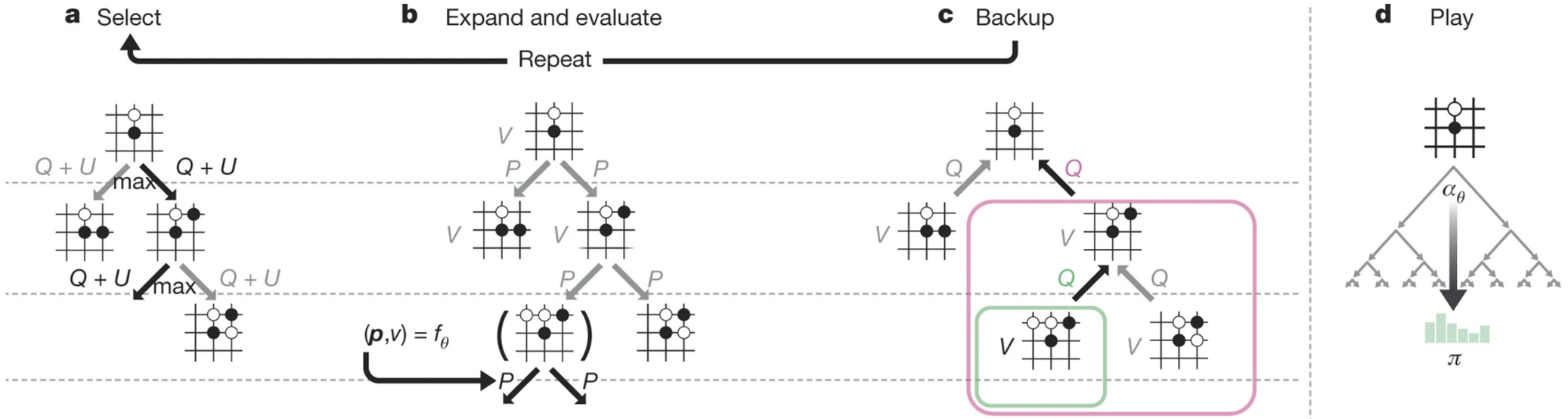


- As **redes neurais de política** p_π **e** p_σ recebem um estado (tabuleiro) s como entrada e o processam com uma sequência de **camadas convolucionais**, e retornam uma distribuição de probabilidades dos movimentos, representado como um mapa de probabilidades no tabuleiro.

- A **rede neural de valor** v_θ usa camadas convolucionais similares, mas retornam um valor escalar $v_\theta(s)$ como previsão do resultado esperado do jogo na posição s.

https://www.nature.com/articles/nature16961

AlphaGo Zero



Seleção

$$PUCT(s,a) = Q(s_t,a) + c_{puct}P(s,a)\frac{\sqrt{(\sum_b N_r(s,b))}}{1+N_r(s,a)}$$

Backpropagation

Atualização dos valores das ações Q

Após N iterações, escolha uma jogada aleatoriamente seguindo a distribuição de visitas π_t

Expansão

Adiciona um filho s_l ao nó selecionado , calcula as probabilidades das ações com uma **rede neural** $f_\theta(s_l)$

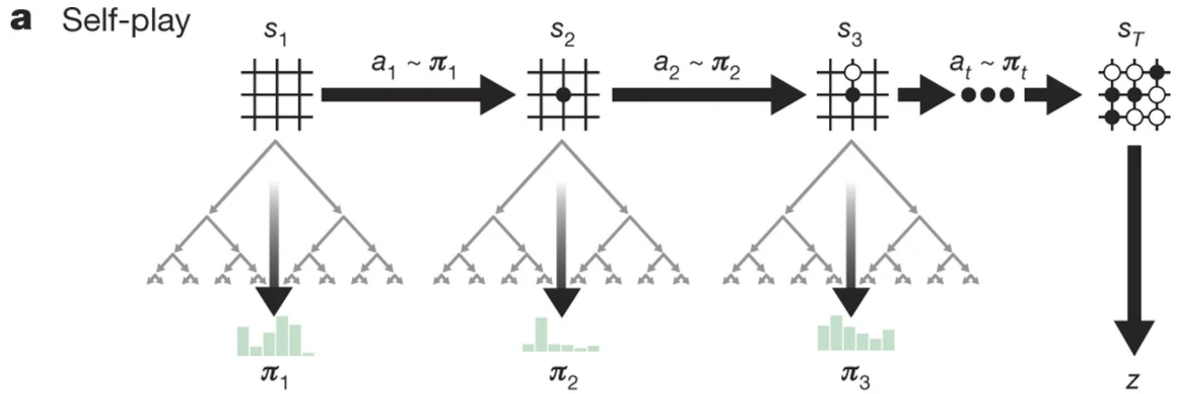
Simulação

Estima o valor de s_l com a mesma **rede neural** $f_\theta(s_l)$**.** Não executa rollout!

https://www.nature.com/articles/nature24270

AlphaGo Zero

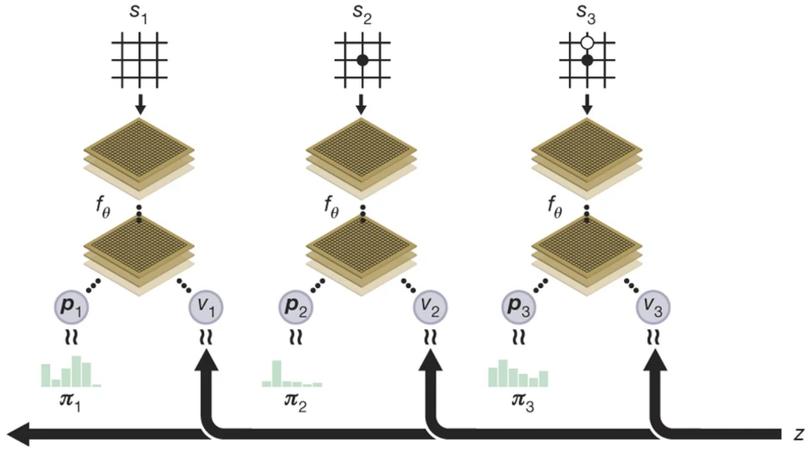
O AlphaGo Zero joga uma partida $s_1, \ldots, s_T$ contra si mesmo e, em cada posição $s_{t'}$ MCTS α_θ é executado usando a **última rede neural** f_θ.



- Os movimentos são selecionados de acordo com a distribuição de visitas feito pelo MCTS, $a_t \sim \pi_t$.
- A utilidade z do estado terminal s_T é definida de acordo com as regras do jogo.

https://www.nature.com/articles/nature24270
https://suragnair.github.io/posts/alphazero.html

AlphaGo Zero



- A rede neural f_θ tem como entrada um estado (tabuleiro) s_t, que é passado por uma sequência de camadas convolucionais, e retorna:
  - (a) Distribuição de probabilidades p_t (vetor) das ações;
  - (b) Probabilidade v_t (escalar) do jogador corrente vencer na posição s_t.
- Os parâmetros θ de f_θ são atualizados para
  - Maximizar a similaridade do vetor p_t com as probabilidades de busca π_t
  - Minimizar o erro entre o vencedor previsto v_t *e o* vencedor real do jogo z
- Os novos parâmetros são utilizados na próxima iteração de self-play

https://www.nature.com/articles/nature24270
https://suragnair.github.io/posts/alphazero.html

Próxima aula

A9: Satisfação de restrição I

Formulação de problemas de busca e algoritmos de busca sem informação: busca em largura, profundidade e custo uniforme