



# APRENDIZADO POR REFORÇO

## Aula 7: Model-Based Reinforcement Learning

Lucas Pereira Cotrim

Marcos Menon José

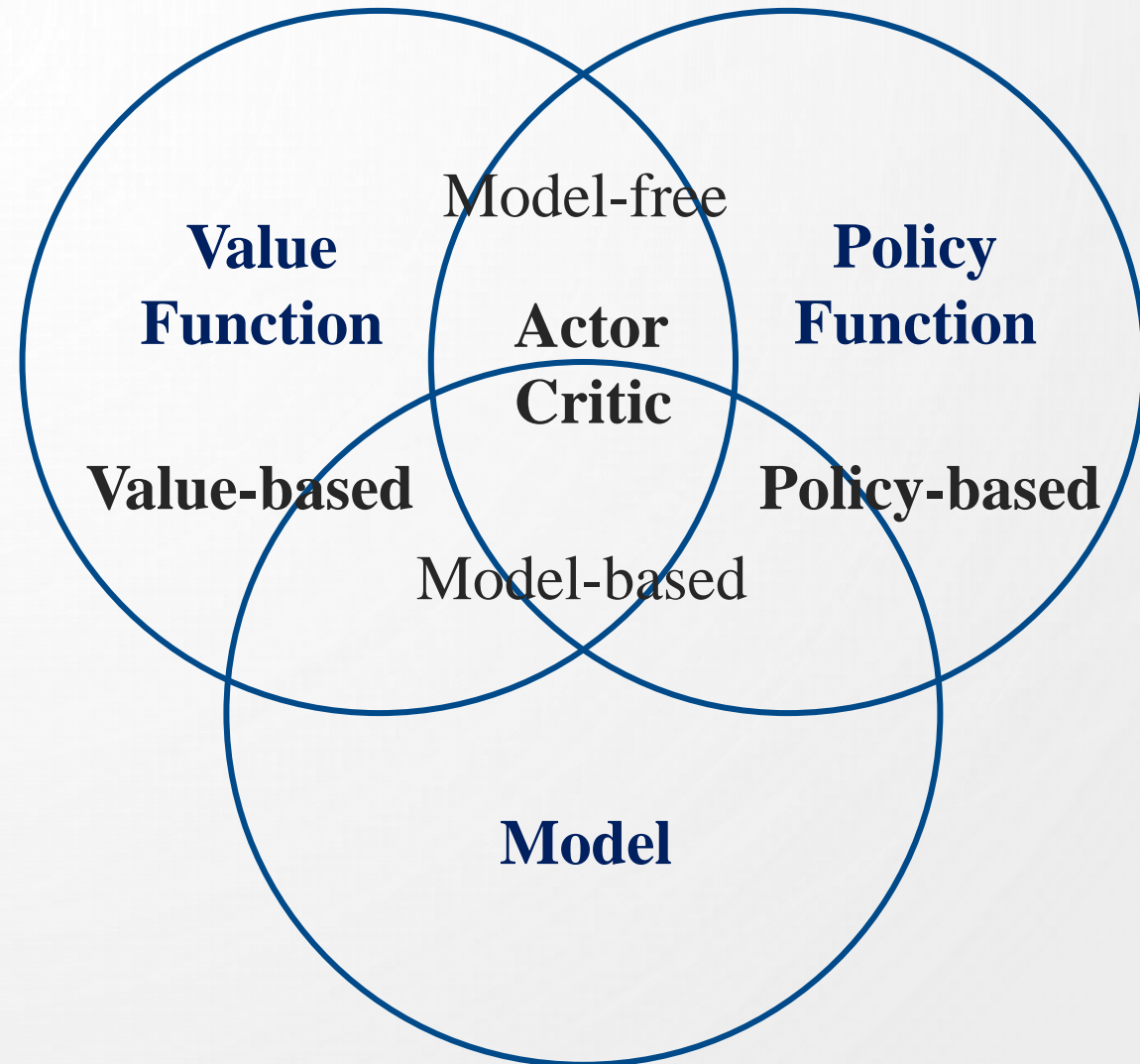
[lucas.cotrim@maua.br](mailto:lucas.cotrim@maua.br)

[marcos.jose@maua.br](mailto:marcos.jose@maua.br)

# RELEMBRANDO ÚLTIMAS AULAS

Algoritmos de aprendizado por Reforço podem ser classificados de acordo com os componentes treinados e presentes no agente:

- Value Function:  $V(s)$ ,  $Q(s, a)$
- Policy Function:  $\pi(a|s)$
- Model:  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$



# RELEMBRANDO ÚLTIMAS AULAS

No começo do curso estudamos algoritmos de **Planejamento** de Programação Dinâmica, nos quais **conhecemos o Modelo do MDP** e resolvemos os problemas de Previsão e Controle **analiticamente**:

- Policy Evaluation
- Policy Iteration
- Value Iteration

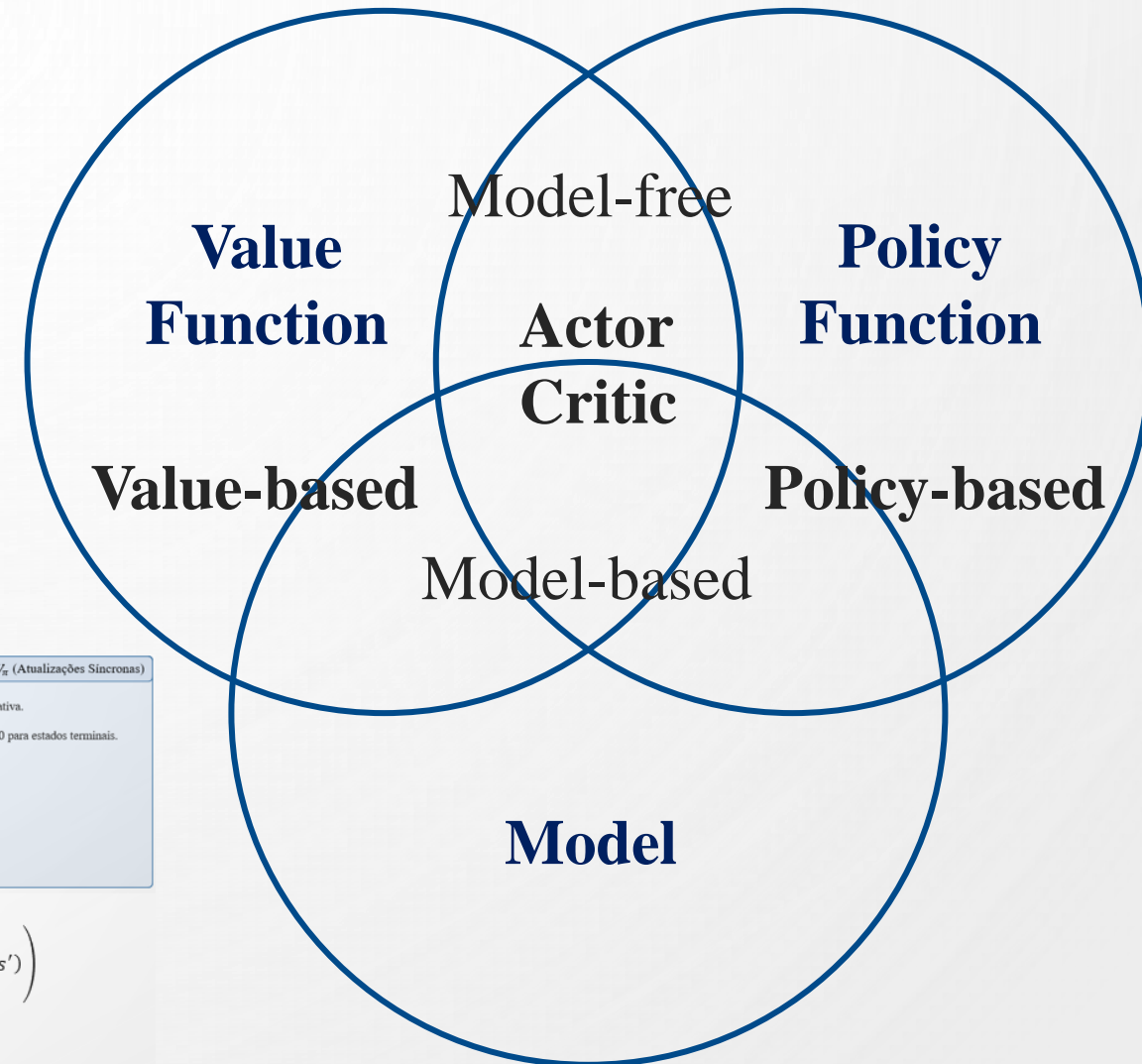
$V_3$	0	-2.44	-2.94	-3
	-2.44	-2.87	-3	-2.94
$\vdots$	-2.94	-3	-2.87	-2.44
	-3	-2.94	-2.44	0
$V_{10}$	0	-6.14	-8.35	-8.97
	-6.14	-7.74	-8.43	-8.35
$\vdots$	-8.35	-8.43	-7.74	-6.14
	-8.97	-8.35	-6.14	0
$V_\infty$	0	-14	-20	-22
	-14	-18	-20	-20
	-20	-20	-18	-14
	-22	-20	-14	0

**Algoritmo:** Avaliação Iterativa de Política  $\pi$  para obtenção de  $V_\pi$  (Atualizações Síncronas)

**Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  e política  $\pi$  a ser avaliada.  
**Parâmetro do Algoritmo:**  $\theta > 0$  para determinar precisão da estimativa.

Inicializar  $V_0(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.  
 $k = 0$   
 Repetir:  
    $\Delta = 0$   
   Repetir para cada  $s \in \mathcal{S}$ :  
      $V_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$   
      $\Delta \leftarrow \max\{\Delta, |V_{k+1}(s) - V_k(s)|\}$   
    $k \leftarrow k + 1$   
 Até que  $\Delta < \theta$   
**Retorna:**  $V_{k+1}$

$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

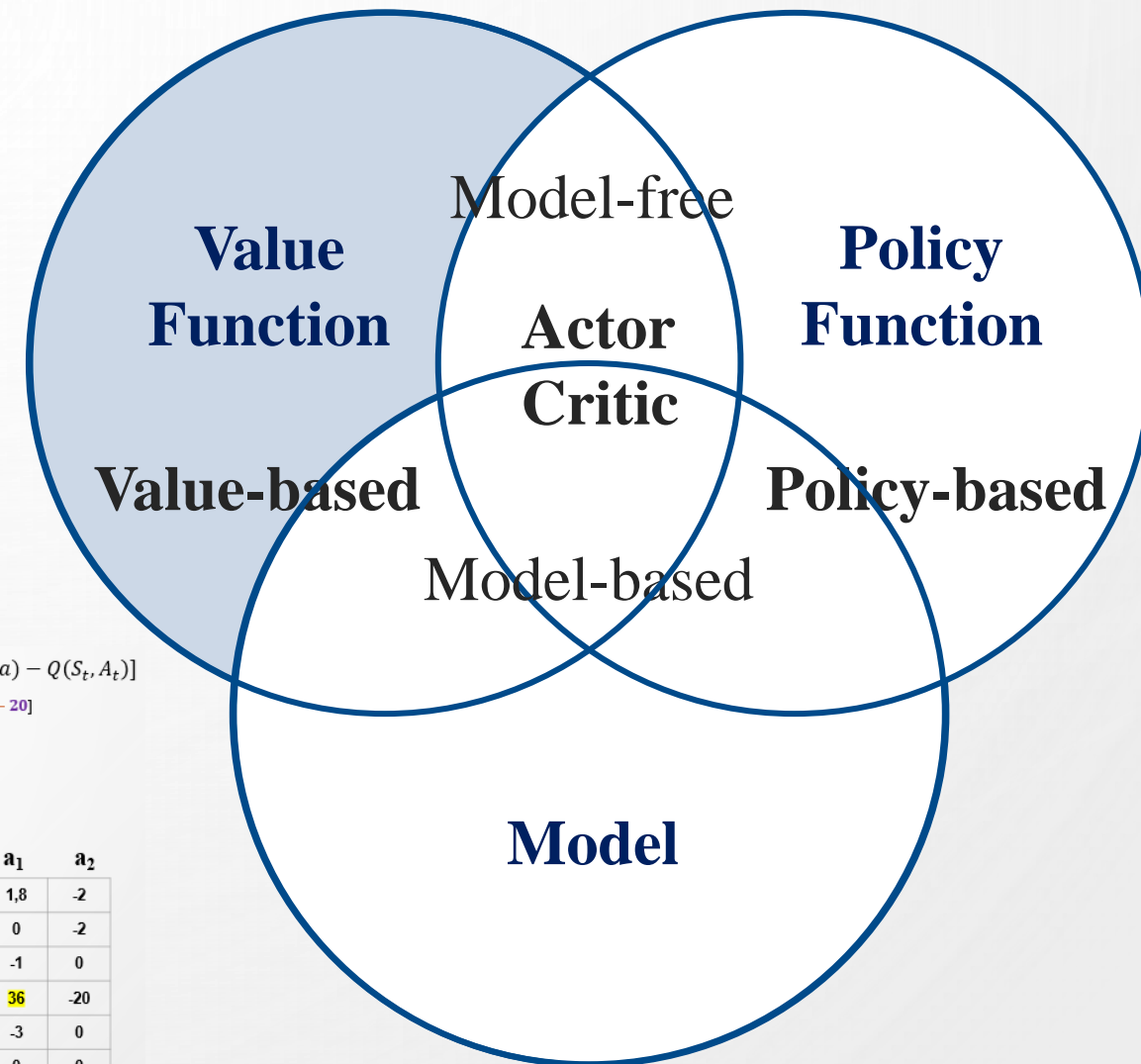
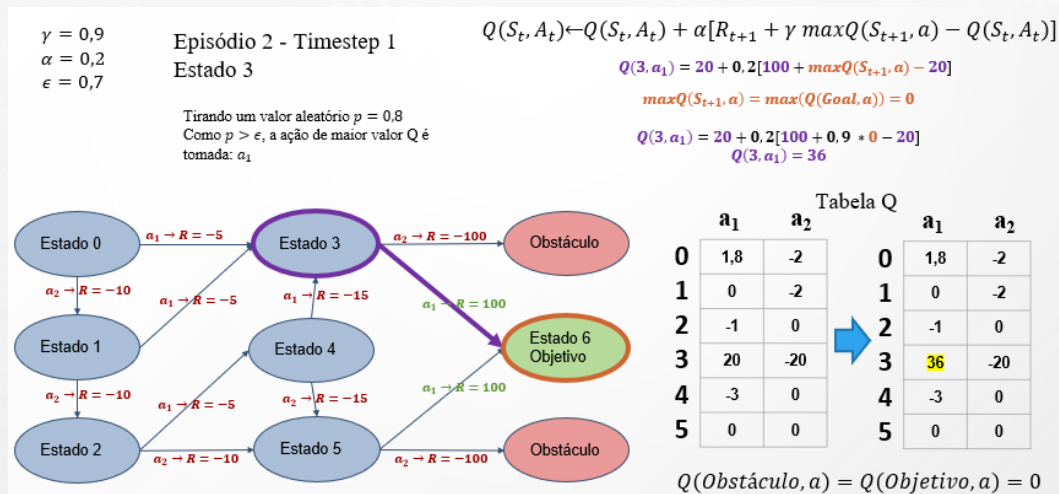


# RELEMBRANDO ÚLTIMAS AULAS

Depois foram estudados métodos de **Model-Free**  
Prediction e Control baseados nas **Funções Valor**

$V(s)$  e  $Q(s, a)$ :

- MC, TD(0), TD( $\lambda$ ) Policy Evaluation
- Monte-Carlo Control
- SARSA
- Q-Learning
- DQN
- DDQN
- Dueling DQN



# RELEMBRANDO ÚLTIMAS AULAS

Na última estudamos Algoritmos **Model-Free** baseados na aproximação da **Função Política**

$\pi_{\theta}(a|s)$ :

- REINFORCE
- REINFORCE with Baseline
- Actor-Critic
- A3C
- DDPG
- TRPO
- PPO

**Algoritmo:** One-Step Actor-Critic Policy Gradient (State Value Critic)

**Input:** Política parametrizada e diferenciável  $\pi_{\theta}(a|s)$ , Função Valor parametrizada e diferenciável  $V_w(s)$

**Parâmetro do Algoritmo:** Taxas de aprendizado  $\alpha_{\theta} > 0$  e  $\alpha_w > 0$

**Inicializar** Parâmetros  $\theta, w$  aleatoriamente

Repetir (para cada episódio):

Inicializar estado  $S_0$

$t = 0$

Repetir enquanto  $S_t$  não é terminal (para cada timestep):

Amostrar ação  $A_t \sim \pi_{\theta}(a|S_t)$

Executar ação  $A_t$  e observar  $S_{t+1}, R_{t+1}$

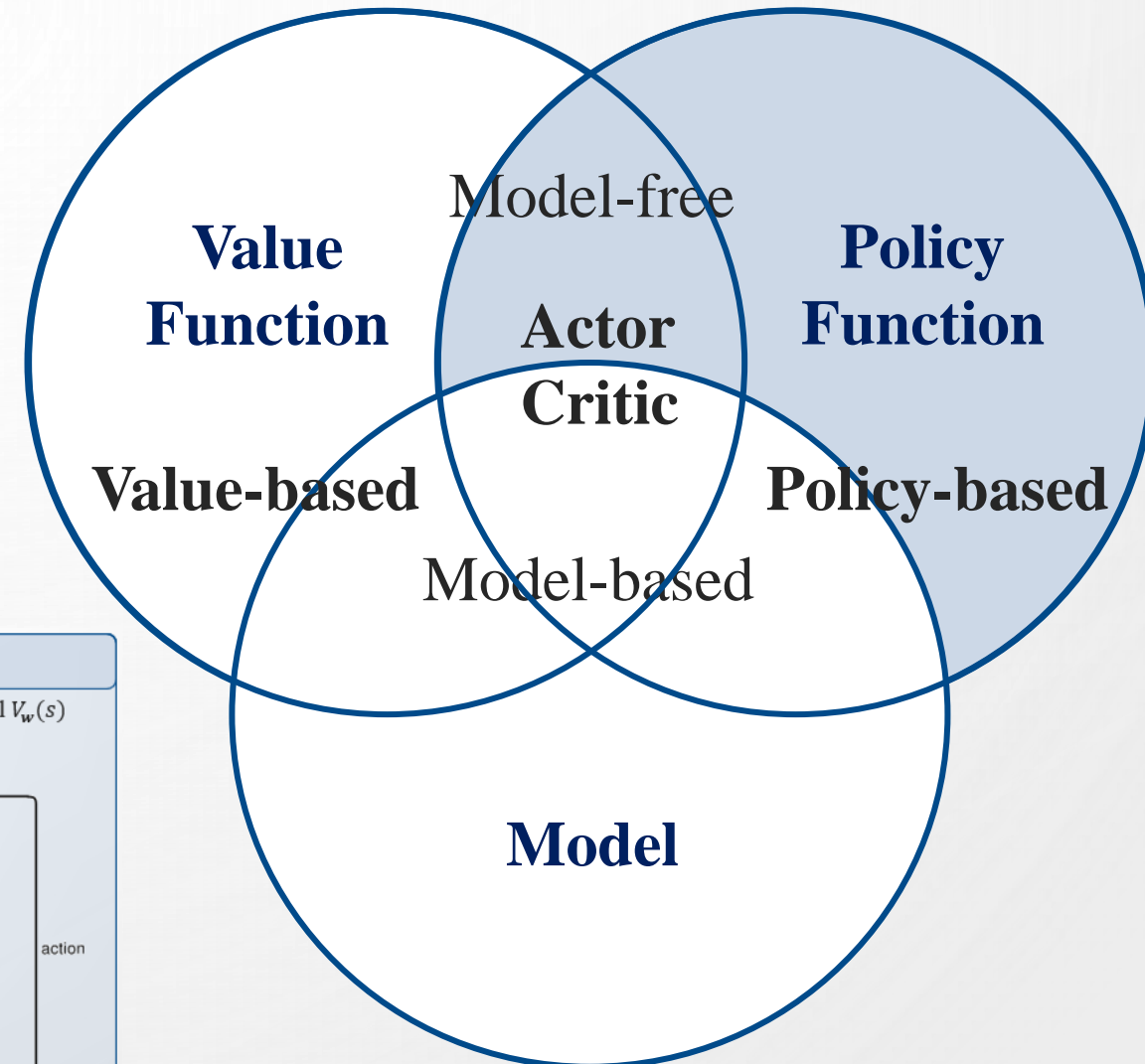
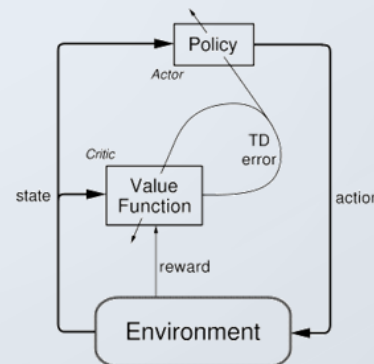
Calcular Erro:  $\delta \leftarrow R_{t+1} + \gamma V_w(S_{t+1}) - V_w(S_t)$

Atualizar Critic:  $w \leftarrow w + \alpha_w \delta \nabla_w V_w(S_t)$

Atualizar Actor:  $\theta \leftarrow \theta + \alpha_{\theta} \gamma^t \delta \nabla_{\theta} \log \pi_{\theta}(A_t|S_t)$

$t \leftarrow t + 1$

**Retorna:**  $\pi_{\theta} \approx \pi^*$

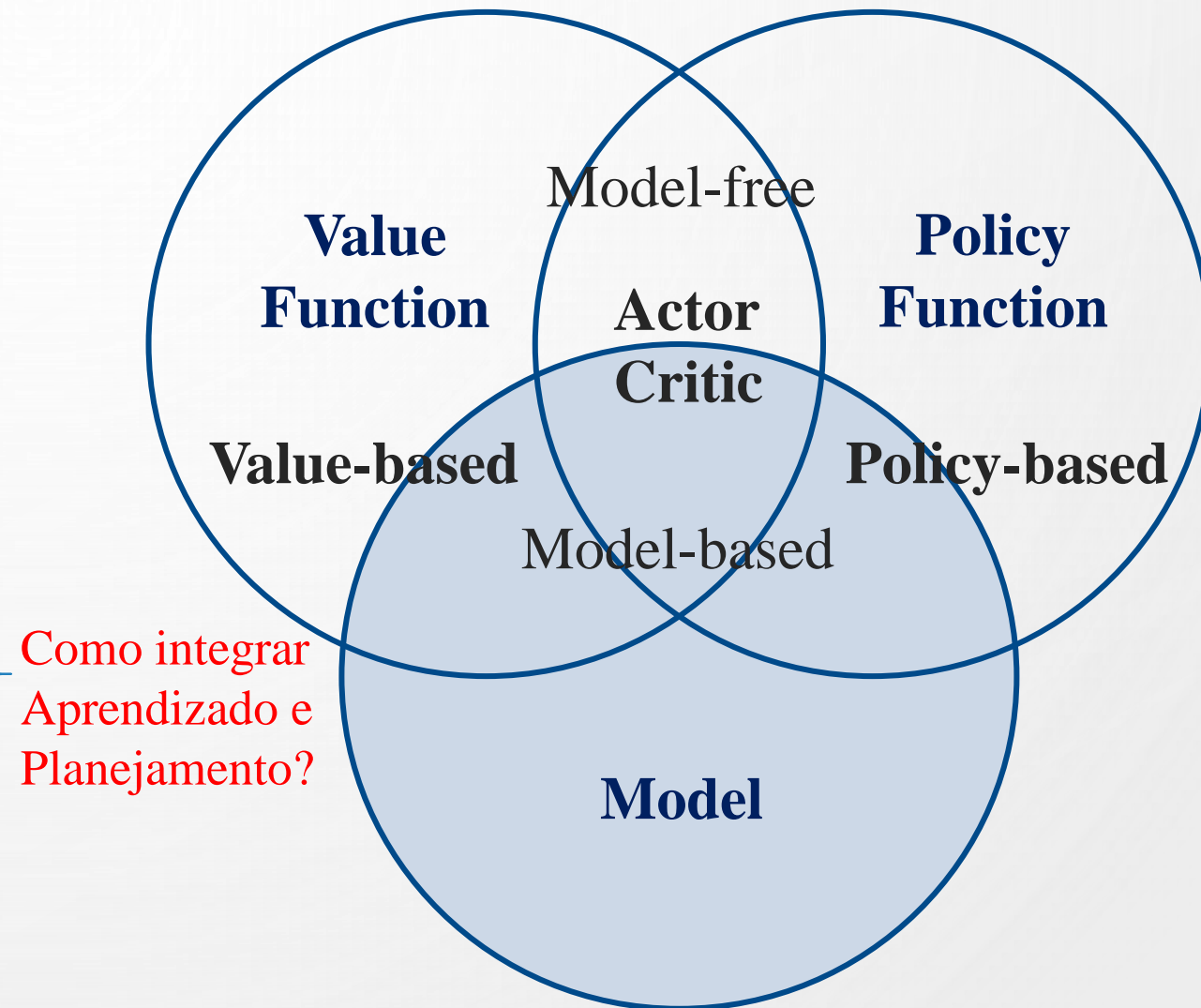




# MODEL-BASED RL

Hoje vamos ver como implementar agentes que aproximam o próprio modelo  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$  do ambiente.

- **Aprendizado:** Aprender modelo  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$  por meio de experiências reais amostradas do MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$
- **Planejamento:** Dado um modelo aproximado  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$ , resolver MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$  e obter  $Q^*$  ou  $\pi^*$ .

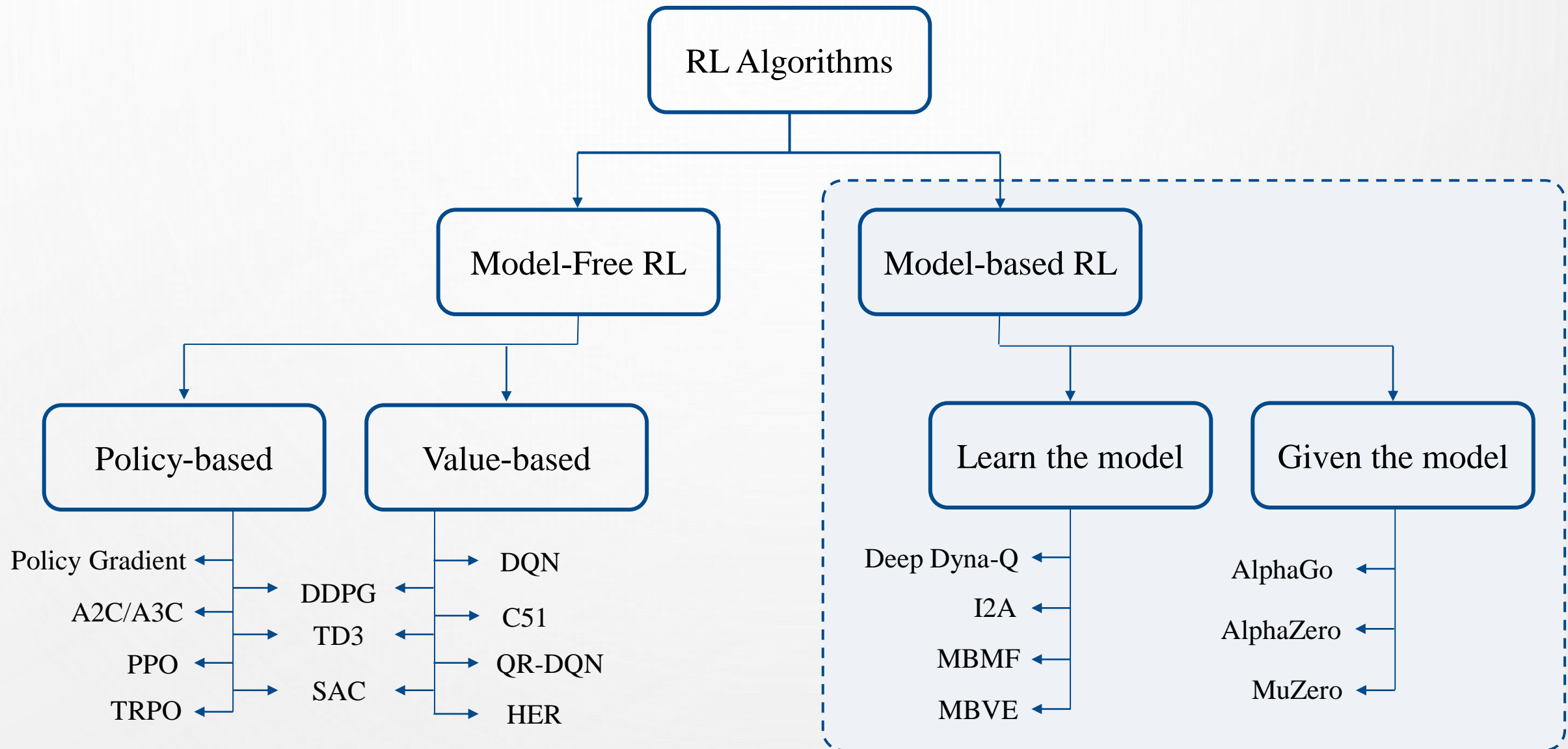


# TÓPICOS DA AULA

Em Model-Based Reinforcement Learning, em vez de aprender diretamente por experiência uma Função Valor  $Q(s, a)$  ou uma Política  $\pi(a|s)$ , o agente aprende um modelo  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$  do ambiente e o resolve por meio de Planejamento.

- Introdução a Model-Based RL
- Integrando Aprendizado e Planejamento (Dyna)
- Dyna-Q
- Deep Dyna-Q (DDQ)
- I2A
- Simulation-Based Search (Monte-Carlo Tree Search - MCTS)

# APRENDIZADO POR REFORÇO: TIPOS DE ALGORITMOS





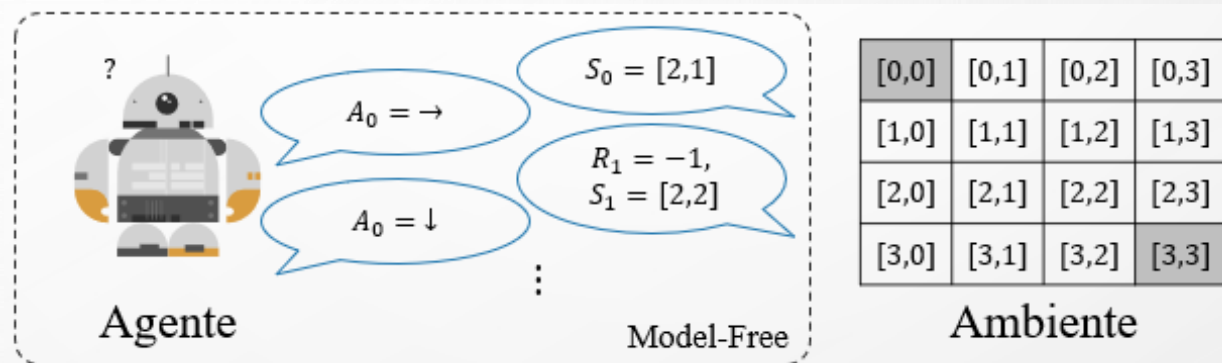
# INTRODUÇÃO A MODEL-BASED RL

Introdução a Model-Based RL

# MODEL FREE RL & MODEL-BASED RL

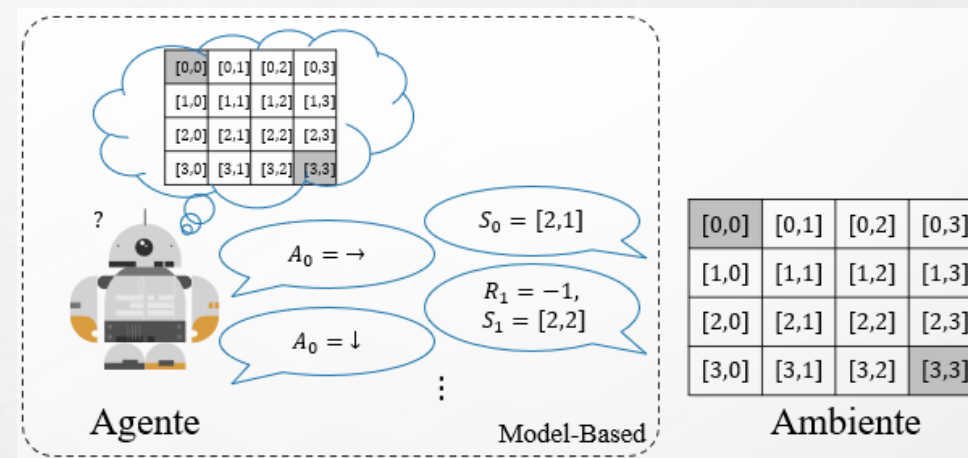
- Model-Free Reinforcement Learning

- Modelo é desconhecido
- Aprender** Função Valor (e/ou Política) a partir de interação com ambiente



- Model-Based Reinforcement Learning

- Aprender um Modelo a partir de interação com ambiente
- Planejar** Função Valor (e/ou Política) a partir de Modelo



# MODEL-BASED RL

- Model-Based Reinforcement Learning
  - Aprender um Modelo a partir de interação com ambiente
  - **Planejar** Função Valor (e/ou Política) a partir de Modelo

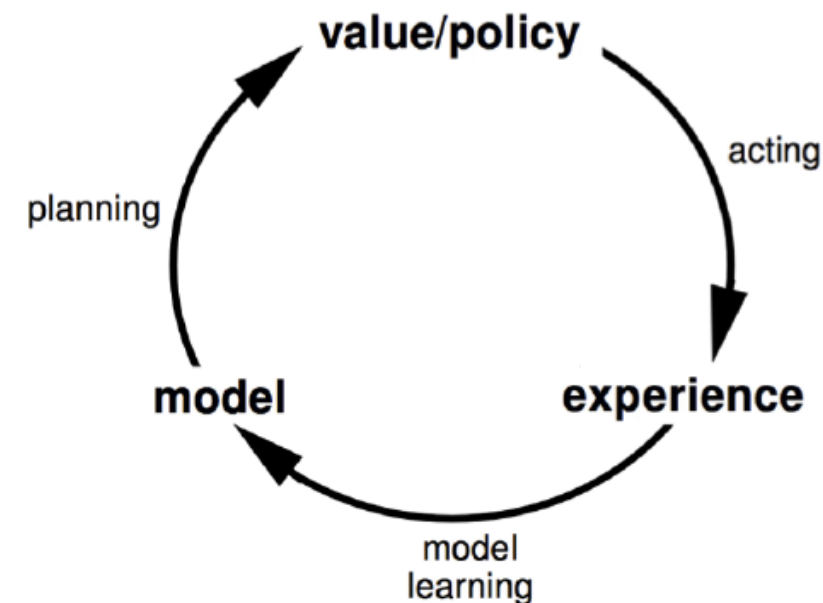
- Por que se preocupar em aproximar o modelo do ambiente?

- Vantagens:

- Maior eficiência amostral do que métodos Model-Free
- Indicado quando interações com o ambiente real são custosas
- Capacidade de raciocinar sobre incerteza do modelo (UCT: Upper Confidence Tree)

- Desvantagens:

- Duas fontes de erro: Aprendizado do Modelo e Planejamento da Função Valor/Política



Fonte: UCL Course on RL by David Silver  
(<https://www.davidsilver.uk/teaching/>)

# DEFINIÇÃO DE MODELO

Um Modelo  $M_\eta$  é uma representação de um MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$  parametrizada por  $\eta$ .

Assumindo que os Espaços de Estados  $\mathcal{S}$  e Ações  $\mathcal{A}$  são conhecidos, temos que  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$  aproxima as Funções de Transições de Estados e Recompensa:

$$\begin{aligned}\mathcal{P}_\eta &\approx \mathcal{P}, & S_{t+1} &\sim \mathcal{P}_\eta(S_{t+1}|S_t, A_t) \\ \mathcal{R}_\eta &\approx \mathcal{R}, & R_{t+1} &= \mathcal{R}_\eta(S_t, A_t)\end{aligned}$$

Em geral modelos assumem independência condicional entre transições de estados e recompensas:

$$\mathbb{P}(S_{t+1}, R_{t+1}|S_t, A_t) = \mathbb{P}(S_{t+1}|S_t, A_t) \mathbb{P}(R_{t+1}|S_t, A_t)$$

# COMO APRENDER UM MODELO

Dado um conjunto de transições (experiência):  $\{(S_t, A_t, R_{t+1}, S_{t+1})_k\}_{k=1}^K$  do ambiente real.

O aprendizado do modelo  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$  é um problema de **Aprendizado Supervisionado**

$$S_1, A_1 \rightarrow R_2, S_2$$

$$S_2, A_2 \rightarrow R_3, S_3$$

$$\vdots$$


$$S_{T-1}, A_{T-1} \rightarrow R_T, S_T$$

- O aprendizado de  $\mathcal{R}_\eta$  a partir de  $\{(S_t, A_t \rightarrow R_{t+1})_k\}_{k=1}^K$  é um problema de **regressão**
- O aprendizado de  $\mathcal{P}_\eta$  a partir de  $\{(S_t, A_t \rightarrow S_{t+1})_k\}_{k=1}^K$  é um problema de **estimativa de densidade de probabilidade**
- Podemos escolher função custo (MSE, MAE, Divergência KL) e encontrar os parâmetros  $\eta$ .

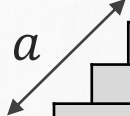
# TIPOS DE MODELOS

Modelos  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$ , de forma análoga a Funções Valor e Políticas, podem ser descritos por *lookup tables* ou aproximadores de funções:

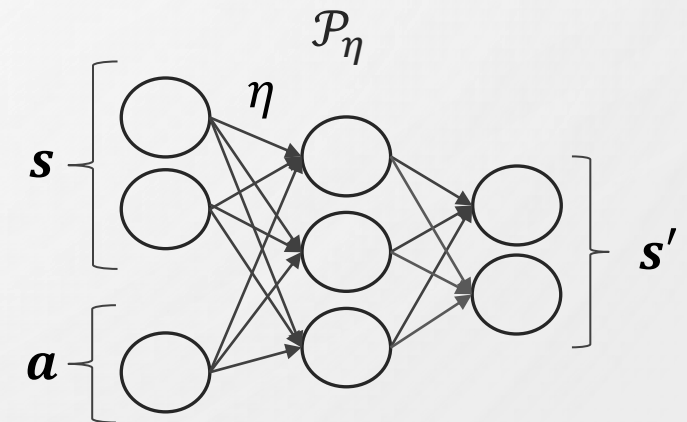
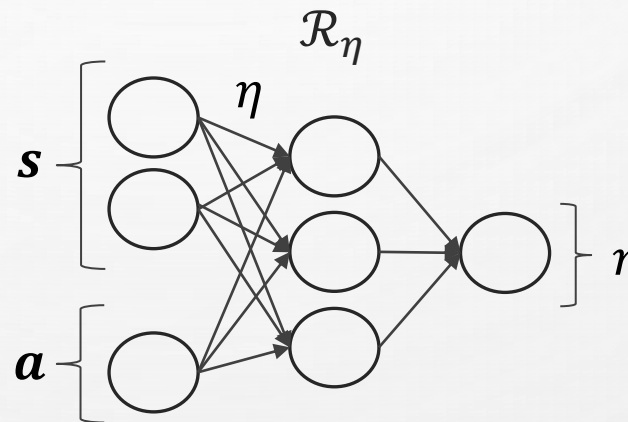
- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Mixture Model (GMM)
- Redes Neurais



$\mathcal{R}_\eta$	$a_1$	...	$a_m$
$s_1$	1	...	4
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$s_n$	3	...	2



$\mathcal{P}_\eta$	$s_1$	...	$s_n$
$s_1$	0	...	1
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$s_n$	0.8	...	0.2





# LOOKUP TABLE MODEL

Um modelo do tipo *Lookup Table* só é aplicável em problemas com Espaços de Estados e Ações  $\mathcal{S}, \mathcal{A}$  discretos e suficientemente pequenos.

Contando o número de visitas a cada par estado-ação  $N(s, a)$  temos:

$$\hat{\mathcal{P}}_{ss'}^a = \frac{1}{N(s, a)} \sum_{t=1}^T 1(S_t = s, A_t = a, S_{t+1} = s')$$

$$\hat{\mathcal{R}}_s^a = \frac{1}{N(s, a)} \sum_{t=1}^T 1(S_t = s, A_t = a) R_t$$

$\mathcal{R}_\eta$	$a_1$	...	$a_m$
$s_1$	1	...	4
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$s_n$	3	...	2

# EXEMPLO: LOOKUP TABLE MODEL

Vamos considerar o seguinte conjunto de episódios de experiência obtidos através de interação com o ambiente real:

Episódio 1:  $(A, a_1, 0, B, a_2, 1, done)$

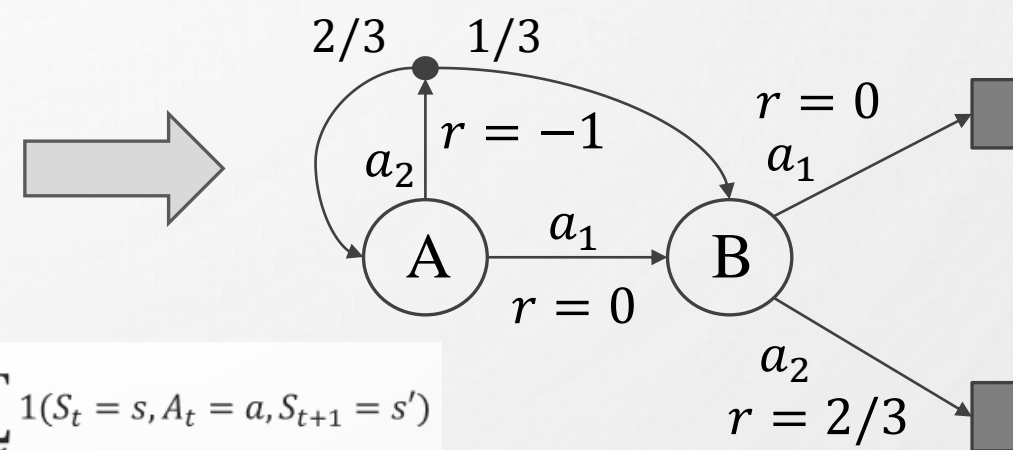
Episódio 2:  $(A, a_2, -1, A, a_1, 0, B, a_2, 1, done)$

Episódio 3:  $(A, a_2, -1, B, a_1, 0, done)$

Episódio 4:  $(A, a_2, -1, A, a_1, 0, B, a_2, 0, done)$

Episódio 5:  $(A, a_1, 0, B, a_1, 0, done)$

A partir da experiência podemos construir o seguinte modelo de MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$

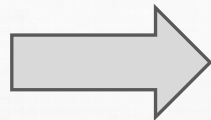
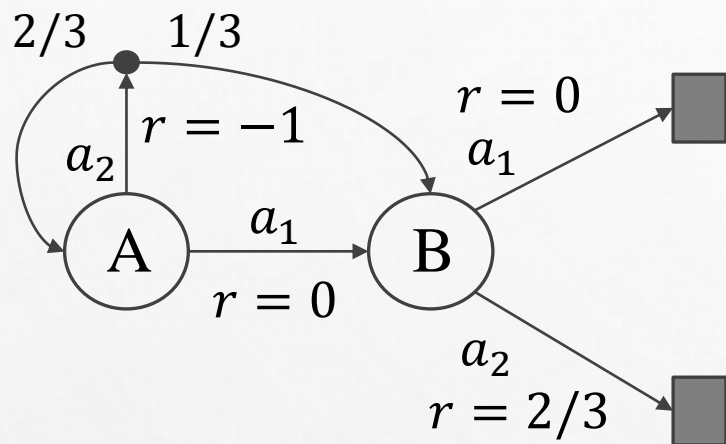


$$\hat{\mathcal{P}}_{ss'}^a = \frac{1}{N(s, a)} \sum_{t=1}^T 1(S_t = s, A_t = a, S_{t+1} = s')$$

$$\hat{\mathcal{R}}_s^a = \frac{1}{N(s, a)} \sum_{t=1}^T 1(S_t = s, A_t = a) R_t$$

# EXEMPLO: LOOKUP TABLE MODEL

A partir do modelo aprendido podemos amostrar episódios **simulados**:



Episódio 1:  $(A, a_1, 0, B, a_2, 1, done)$

Episódio 2:  $(A, a_2, -1, A, a_1, 0, B, a_2, 1, done)$

Episódio 3:  $(A, a_2, -1, B, a_1, 0, done)$

Episódio 4:  $(A, a_2, -1, A, a_1, 0, B, a_2, 0, done)$

Episódio 5:  $(A, a_1, 0, B, a_1, 0, done)$

Episódio 6:  $(A, a_1, 0, B, a_2, 1, done)$

Episódio 7:  $(A, a_2, -1, A, a_1, 0, B, a_2, 1, done)$

Episódio 8:  $(A, a_2, -1, B, a_1, 0, done)$

Episódio 9:  $(A, a_2, -1, A, a_1, 0, B, a_2, 0, done)$

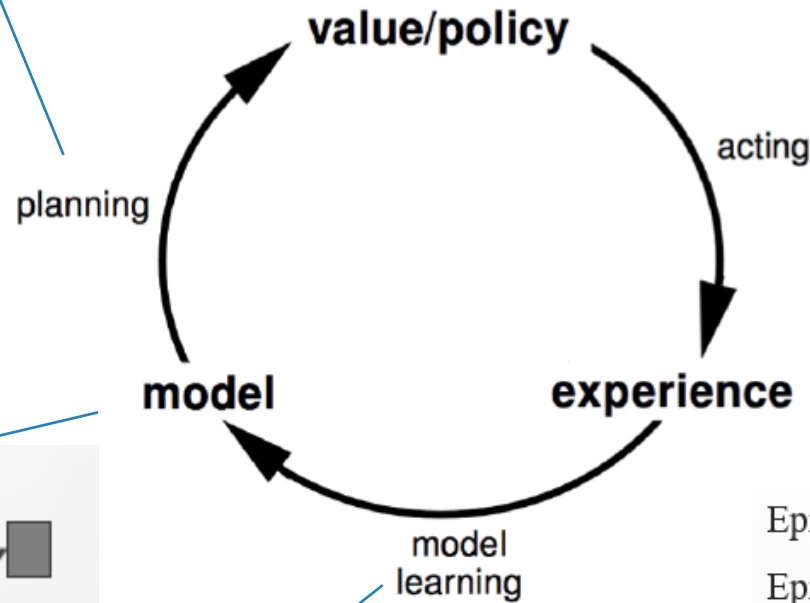
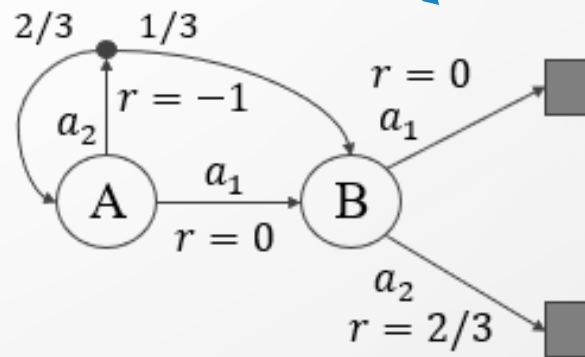
Episódio 10:  $(A, a_1, 0, B, a_1, 0, done)$

# EXEMPLO: LOOKUP TABLE MODEL

## Experiência Simulada

Episódio 1: (A,  $a_1$ , 0, B,  $a_2$ , 1, done)  
 Episódio 2: (A,  $a_2$ , -1, A,  $a_1$ , 0, B,  $a_2$ , 1, done)  
 Episódio 3: (A,  $a_2$ , -1, B,  $a_1$ , 0, done)  
 Episódio 4: (A,  $a_2$ , -1, A,  $a_1$ , 0, B,  $a_2$ , 0, done)  
 Episódio 5: (A,  $a_1$ , 0, B,  $a_1$ , 0, done)  
 Episódio 6: (A,  $a_1$ , 0, B,  $a_2$ , 1, done)  
 Episódio 7: (A,  $a_2$ , -1, A,  $a_1$ , 0, B,  $a_2$ , 1, done)  
 Episódio 8: (A,  $a_2$ , -1, B,  $a_1$ , 0, done)  
 Episódio 9: (A,  $a_2$ , -1, A,  $a_1$ , 0, B,  $a_2$ , 0, done)  
 Episódio 10: (A,  $a_1$ , 0, B,  $a_1$ , 0, done)

## Modelo



## Experiência real

Episódio 1: (A,  $a_1$ , 0, B,  $a_2$ , 1, done)  
 Episódio 2: (A,  $a_2$ , -1, A,  $a_1$ , 0, B,  $a_2$ , 1, done)  
 Episódio 3: (A,  $a_2$ , -1, B,  $a_1$ , 0, done)  
 Episódio 4: (A,  $a_2$ , -1, A,  $a_1$ , 0, B,  $a_2$ , 0, done)  
 Episódio 5: (A,  $a_1$ , 0, B,  $a_1$ , 0, done)

$$\hat{P}_{ss'}^a = \frac{1}{N(s, a)} \sum_{t=1}^T 1(S_t = s, A_t = a, S_{t+1} = s')$$

$$\hat{R}_s^a = \frac{1}{N(s, a)} \sum_{t=1}^T 1(S_t = s, A_t = a) R_t$$

# MODEL-BASED RL: PLANNING

Uma vez obtido um modelo  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$ , como resolver o MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$  por Planejamento?

## DP: VALUE ITERATION

**Algoritmo:** Iteração sobre Função Valor para obtenção de  $\pi^*, V^*$  (Atualizações Síncronas)

**Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ .

**Parâmetro do Algoritmo:**  $\theta > 0$  para determinar precisão da estimativa.

**Inicializar**  $V_0(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.

$k = 0$   
 Repetir:  
    $\Delta = 0$   
   Repetir para cada  $s \in \mathcal{S}$ :  
      $V_{k+1} \leftarrow (s) \max_{a \in \mathcal{A}} [R(s, a) + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V_k(s')]$   
      $\Delta \leftarrow \max\{\Delta, |V_{k+1}(s) - V_k(s)|\}$   
    $k \leftarrow k + 1$   
 Até que  $\Delta < \theta$   
**Retorna:**  $\pi = \arg\max_{a \in \mathcal{A}} [R(s, a) + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V_{k+1}(s')] \approx \pi^*$  e  $V_{k+1} \approx V^*$

## DP: POLICY ITERATION

**Algoritmo:** Iteração sobre Política  $\pi$  para obter  $\pi^*, V^*$  (Atualizações Síncronas)

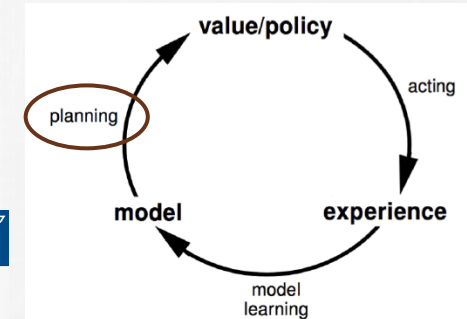
**Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ .

**Parâmetro do Algoritmo:**  $\theta > 0$  para determinar precisão da estimativa.

**Inicializar:**  $V_0(s)$  e  $\pi(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.

**1) Policy Evaluation**  
 $k = 0$   
 Repetir:  
    $\Delta = 0$   
   Repetir para cada  $s \in \mathcal{S}$ :  
      $V_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) [R(s, a) + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V_k(s')]$   
      $\Delta \leftarrow \max\{\Delta, |V_{k+1}(s) - V_k(s)|\}$   
    $k \leftarrow k + 1$   
 Até que  $\Delta < \theta$

**2) Policy Improvement**  
 $policy\_stable \leftarrow True$   
 Repetir para cada  $s \in \mathcal{S}$ :  
    $a_{old} \leftarrow \pi(s)$   
    $\pi(s) = \arg\max_{a \in \mathcal{A}} [R(s, a) + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V_{k+1}(s')]$   
   Se  $a_{old} \neq \pi(s)$ , então  $policy\_stable \leftarrow False$   
 Se  $policy\_stable$ , **Retorna**  $\pi \approx \pi^*, V_{k+1} \approx V^*$ , caso contrário volta para 1)



- Value Iteration  $\longrightarrow$
- Policy Iteration  $\longrightarrow$
- Sample-Based Planning  $\longrightarrow$  Amostrar Interações com ambiente e aplicar Model-Free RL
- Tree Search  $\longrightarrow$  MCTS (SARSA, Q-Learning, DQN, Actor-Critic)

# SAMPLE-BASED PLANNING

Sample-Based Planning consiste em utilizar o modelo aprendido  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$  **apenas para gerar amostras de experiência por simulação**, em seguida aplica-se Model-Free RL sobre as amostras:

- Amostrar transições simuladas de acordo com modelo:

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1}|S_t, A_t)$$

$$R_{t+1} = \mathcal{R}_\eta(S_t, A_t)$$

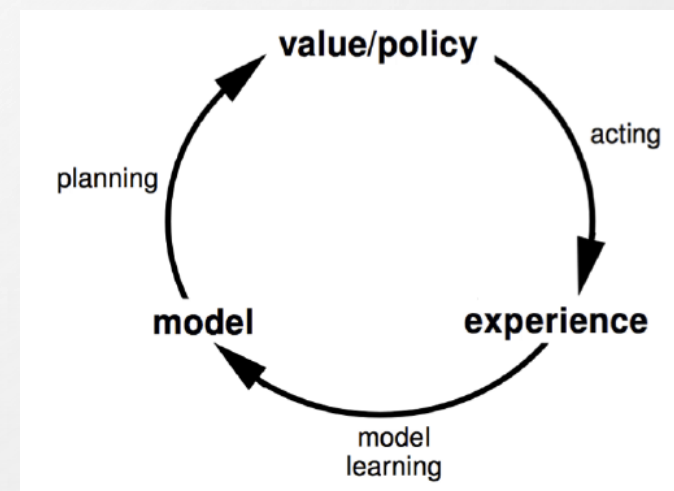
- Aplicar Model-Free RL sobre transições simuladas:
  - SARSA
  - Q-Learning/DQN
  - Actor-Critic



# MODEL-BASED RL: PRINCIPAIS PROBLEMAS

A estrutura de Model-Based RL vista até agora apresenta diversos problemas:

- Se o **modelo aprendido é diferente do real**  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\} \neq \{\mathcal{P}, \mathcal{R}\} = M$ , a performance é limitada à política ótima no MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- Quando o modelo não é preciso, o planejamento levará a **política sub-ótima** no ambiente real.
- **Soluções:**
  - Utilizar Model-Free RL para atualizar Política com mais frequência
  - Considerar incerteza do modelo explicitamente:
    - Priorizar treino do modelo em regiões de maior incerteza



# INTEGRANDO APRENDIZADO E PLANEJAMENTO (DYNA)

Integrando Aprendizado e  
Planejamento (Dyna)

Vamos considerar duas fontes de experiência:

- **Experiência Real:** Amostrada do ambiente (MDP verdadeiro  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ )

$$S' \sim \mathcal{P}_{SS'}^a$$

$$R = \mathcal{R}_S^a$$

- **Experiência Simulada:** Amostrada do modelo (MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ )

$$S' \sim \mathcal{P}_\eta(S'|S, A)$$

$$R = \mathcal{R}_\eta(S, A)$$

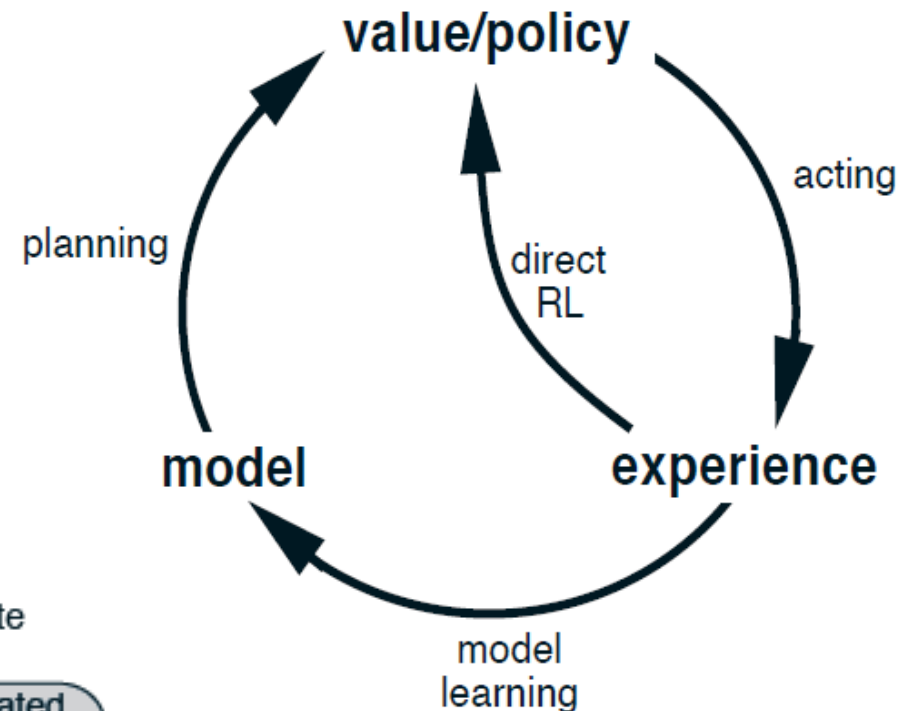
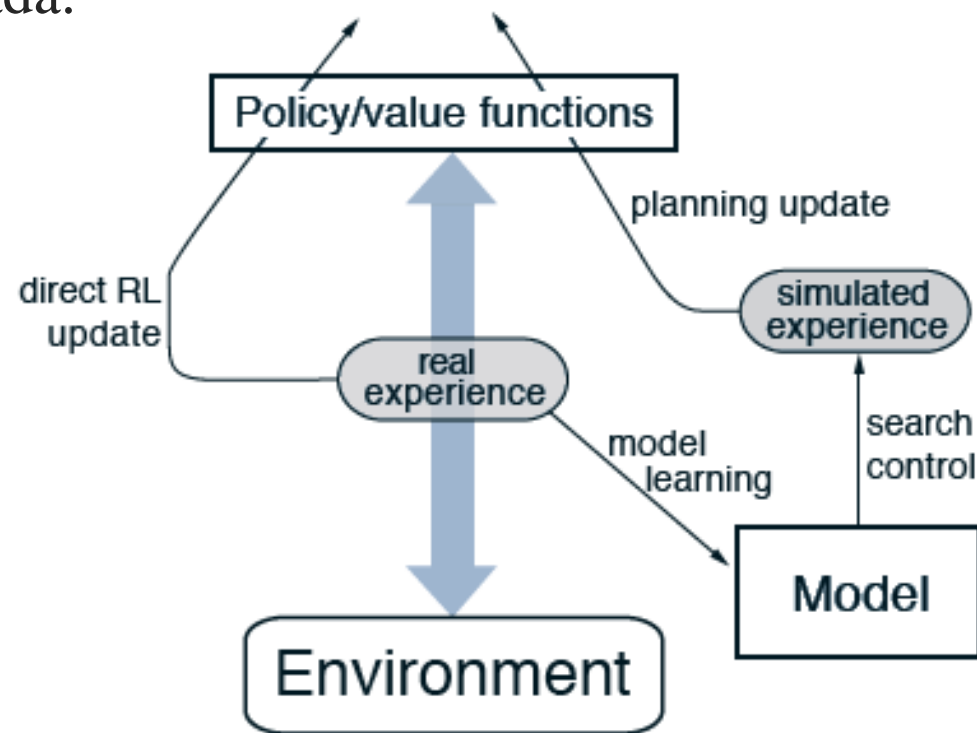
# MODEL-BASED RL: INTEGRAÇÃO ENTRE APRENDIZADO E PLANEJAMENTO

- Model-Free RL:
  - Não existe modelo treinado.
  - **Aprendizado** de Função Valor (e/ou Política) diretamente de experiência real.
- Model-Based RL (Sample-Based Planning):
  - Modelo é aprendido a partir de experiência real
  - **Planejamento** de Função Valor (e/ou Política) de experiência simulada
- Model-Based RL (Dyna):
  - Modelo é aprendido a partir de experiência real
  - **Aprendizado e Planejamento** de Função Valor (e/ou Política) a partir de experiência real e simulada

# MODEL-BASED RL: INTEGRAÇÃO ENTRE APRENDIZADO E PLANEJAMENTO

## Algoritmo Dyna: Integração entre Planejamento e Aprendizado

- Função Valor/Política é treinada a partir de experiência real e simulada.
- Arquitetura compatível com todos algoritmos Model-Free estudados (DQN, REINFORCE, DDPG, PPO, ...)



Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction*, The MIT Press (2020).

# ALGORITMO DYNA-Q: DYNA + Q-LEARNING

## Algoritmo: Dyna-Q

**Parâmetro do Algoritmo:** Número de passos de planejamento  $n$ , taxa de aprendizado  $\alpha$ .

**Inicializar**  $Q(s, a)$  e  $M_\eta(s, a) = \mathcal{P}_\eta(s, a), \mathcal{R}_\eta(s, a)$  para todo  $s \in \mathcal{S}$  e  $a \in \mathcal{A}(s)$ .

Repetir para cada episódio:

$S = s_0 \sim \mathbb{P}(s_0)$

Enquanto  $S$  não é estado terminal:

Amostrar ação  $A \sim \epsilon - greedy(Q(S, a))$

**Executar ação  $A$  e observar  $R$  e  $S'$**

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{a \in \mathcal{A}(S')} Q(S', a) - Q(S, A)]$

**Atualiza Modelo:**  $\mathcal{P}_\eta(S, A) \leftarrow S', \mathcal{R}_\eta(S, A) \leftarrow R$

Repetir  $n$  vezes:

$S_- \leftarrow$  Estado aleatório previamente visto,  $A_- \leftarrow$  Ação aleatória previamente tomada em  $S_-$

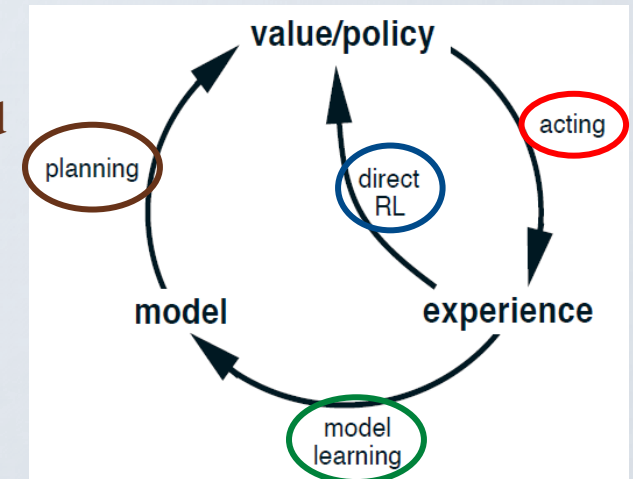
$S'_-, R_- \leftarrow M_\eta(S_-, A_-)$

$Q(S_-, A_-) \leftarrow Q(S_-, A_-) + \alpha[R_- + \gamma \max_{a \in \mathcal{A}(S'_-)} Q(S'_-, a) - Q(S_-, A_-)]$

$S \leftarrow S'$

**Retorna:**  $Q(s, a) \approx Q^*(s, a)$

Sample-based  
planning





# EXEMPLO: DYNA-Q

Considere o seguinte problema de posicionamento de um agente no Gridworld com obstáculos ilustrado na figura (Exemplo 8.1 do *Sutton&Barto*).

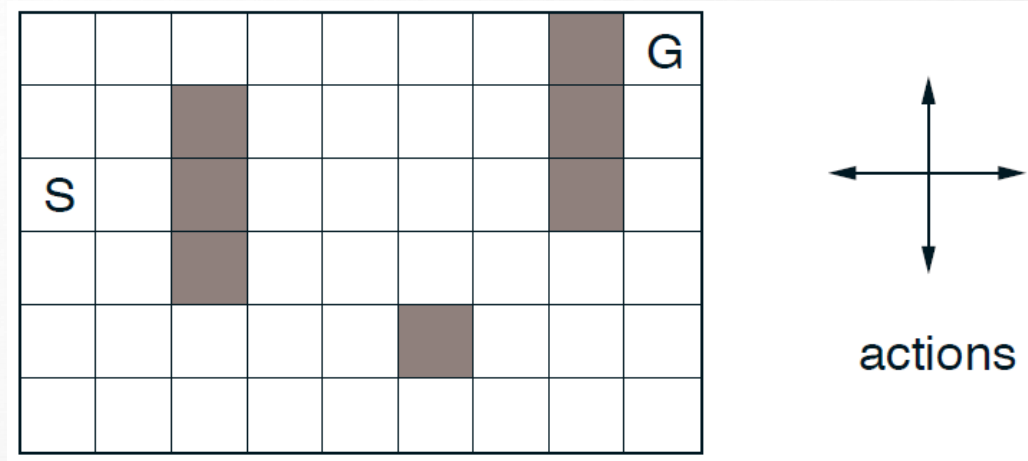
- Interação com ambiente é custosa

Escala geográfica

Tempo de locomoção

Combustível

- Model-Based RL permite o aprendizado da Função Valor Ótima  $Q^*$  com interação reduzida com ambiente real



# DynaQ\_Maze\_Sutton.ipynb

CO

DynaQ\_Maze\_Sutton.ipynb

☆

File Edit View Insert Runtime Tools Help All changes saved

Comment Share Settings L

RAM Disk Editing

Table of contents

Q

Dyna-Q: Maze (exemplo 8.1 Sutton&Barto)

Imports

<>

Environment Class: Maze

Classe DynaParams

Política  $\epsilon$ -greedy

Classe TrivialModel

Dyna-Q

Figura 8.2

Section

+ Code + Text

▼ Dyna-Q: Maze (exemplo 8.1 Sutton&Barto)

Vamos implementar o algoritmo Dyna-Q para resolver o problema descrito no exemplo 8.1 do livro "Reinforcement Learning: An Introduction" de Sutton R. e Barto, A.

O seguinte código foi adaptado de Shangtong Zhang e Kenta Shimada, repositório em:

<https://github.com/ShangtongZhang/reinforcement-learning-an-introduction>

O problema de a ser resolvido consiste no posicionamento de um agente em um ambiente do tipo GridWorld ilustrado abaixo, onde o agente inicia um episódio na posição indicada por S e o episódio termina quando o mesmo alcança a posição G. O movimento ocorre de forma determinística em função das quatro possíveis ações (N,S,E,W) e as células hachuradas indicam obstáculos sobre os quais o agente não pode se mover.


↑

←

→

↓

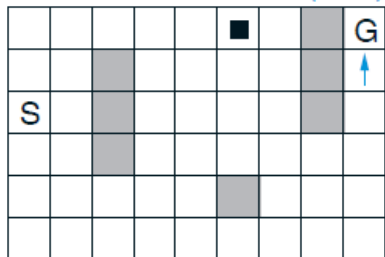
actions

# COLAB DYNA-Q

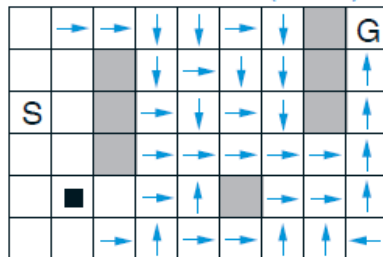
- Dyna-Q reduziu consideravelmente o número de episódios de interação com ambiente real em comparação com Q-Learning para aprendizado da função valor ótima  $Q^*$

Políticas encontradas na metade do segundo episódio:

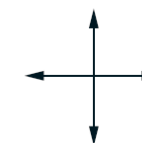
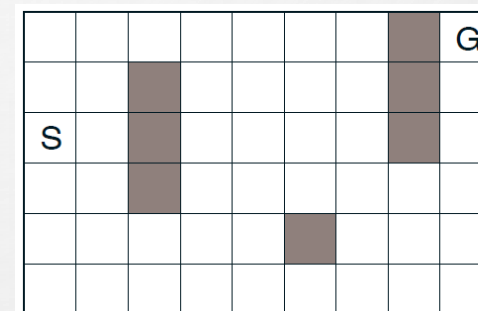
WITHOUT PLANNING ( $n=0$ )



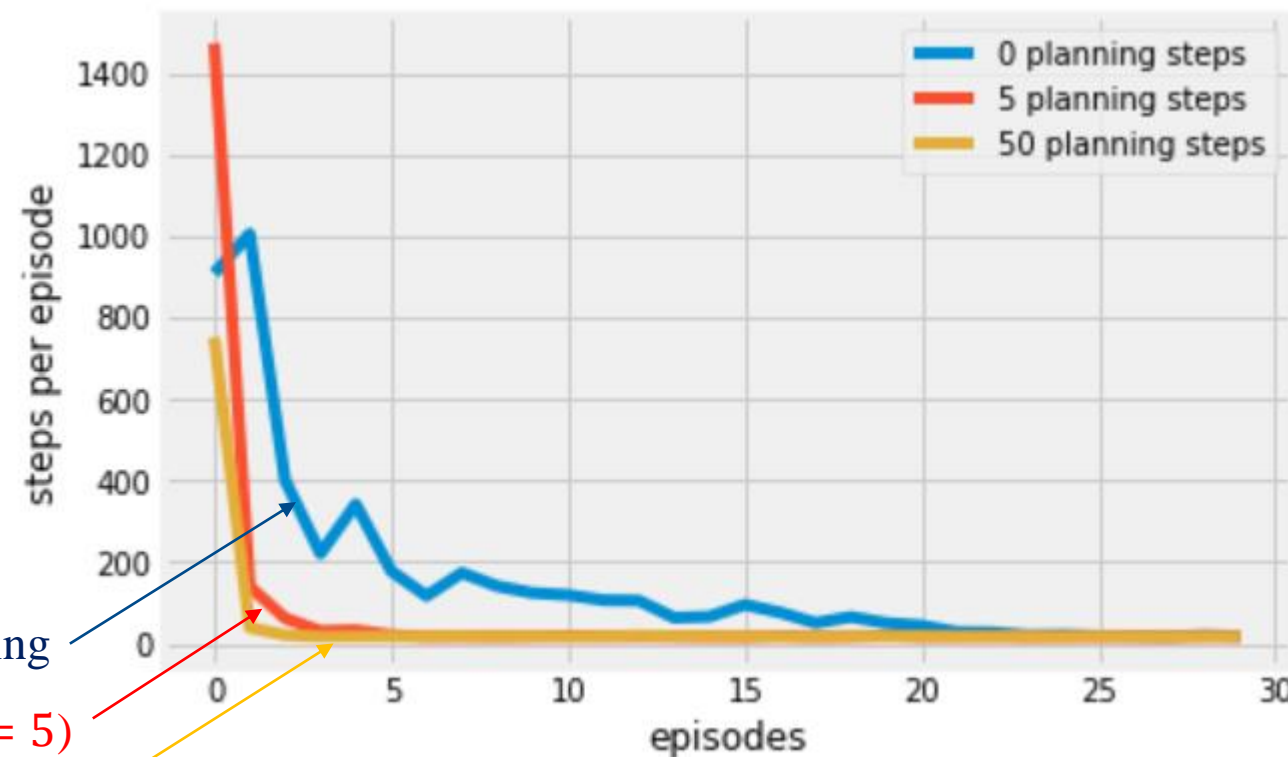
WITH PLANNING ( $n=50$ )



Dyna-Q ( $n = 50$ )



actions



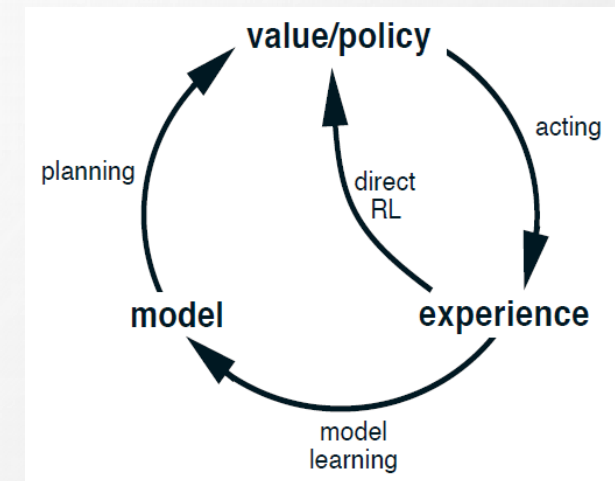
# DEEP DYNA-Q (DDQ)

Deep Dyna-Q (DDQ)

# DEEP DYNA Q: DYNA + DQN

A arquitetura de algoritmos Dyna pode ser implementada com qualquer algoritmo de Model-Free RL no passo de Planejamento.

- De forma análoga à extensão do algoritmo Q-Learning para o DQN com aproximadores de funções, podemos estender o Dyna-Q para o **Deep Dyna-Q (DDQ)** por meio do treinamento dos parâmetros  $\theta$  de uma Função Valor parametrizada  $Q_\theta(s, a)$  a partir de **experiência real e simulada** e por meio do treinamento de um modelo  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$



# DEEP DYNA Q: DYNA + DQN

## Algorithm 1 Deep Dyna-Q for Dialogue Policy Learning

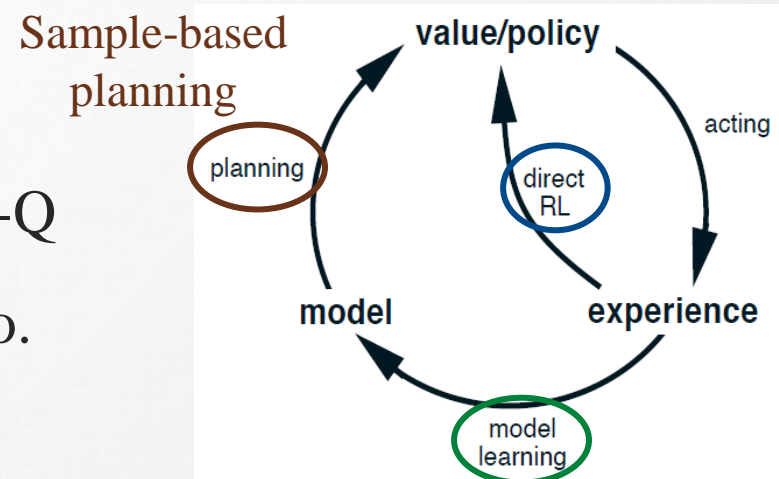
**Require:**  $N, \epsilon, K, L, C, Z$

**Ensure:**  $Q(s, a; \theta_Q), M(s, a; \theta_M)$

- 1: initialize  $Q(s, a; \theta_Q)$  and  $M(s, a; \theta_M)$  via pre-training on human conversational data
- 2: initialize  $Q'(s, a; \theta_{Q'})$  with  $\theta_{Q'} = \theta_Q$
- 3: initialize real experience replay buffer  $D^u$  using Reply Buffer Spiking (RBS), and simulated experience replay buffer  $D^s$  as empty
- 4: **for**  $u=1:N$  **do**
- 5:   **# Direct Reinforcement Learning starts**
- 6:   user starts a dialogue with user action  $a^u$
- 7:   generate an initial dialogue state  $s$
- 8:   **while**  $s$  is not a terminal state **do**
- 9:     with probability  $\epsilon$  select a random action  $a$
- 10:    otherwise select  $a = \arg\max_a Q(s, a; \theta_Q)$
- 11:    execute  $a$ , and observe user response  $a^u$  and reward  $r$
- 12:    update dialogue state to  $s'$
- 13:    store  $(s, a, r, a^u, s')$  to  $D^u$
- 14:     $s = s'$
- 15:   **end while**
- 16:   sample random minibatches of  $(s, a, r, s')$  from  $D^u$
- 17:   update  $\theta_Q$  via Z-step minibatch Q-learning according to Equation (2)
- 18:   **# Direct Reinforcement Learning ends**
- 19:   **# World Model Learning starts**
- 20:   sample random minibatches of training samples  $(s, a, r, a^u, s')$  from  $D^u$
- 21:   update  $\theta_M$  via Z-step minibatch SGD of multi-task learning
- 22:   **# World Model Learning ends**
- 23:   **# Planning starts**
- 24:   **for**  $k=1:K$  **do**
- 25:      $t = \text{FALSE}, l = 0$
- 26:     sample a user goal  $G$
- 27:     sample user action  $a^u$  from  $G$
- 28:     generate an initial dialogue state  $s$
- 29:     **while**  $t$  is  $\text{FALSE} \wedge l \leq L$  **do**
- 30:       with probability  $\epsilon$  select a random action  $a$
- 31:       otherwise select  $a = \arg\max_a Q(s, a; \theta_Q)$
- 32:       execute  $a$
- 33:       world model responds with  $a^u, r$  and  $t$
- 34:       update dialogue state to  $s'$
- 35:       store  $(s, a, r, s')$  to  $D^s$
- 36:        $l = l + 1, s = s'$
- 37:     **end while**
- 38:     sample random minibatches of  $(s, a, r, s')$  from  $D^s$
- 39:     update  $\theta_Q$  via Z-step minibatch Q-learning according to Equation (2)
- 40:   **end for**
- 41:   **# Planning ends**
- 42:   every  $C$  steps reset  $\theta_{Q'} = \theta_Q$
- 43: **end for**

A arquitetura de algoritmos Dyna pode ser implementada com qualquer algoritmo de Model-Free RL no passo de Planejamento. Se utilizarmos o **DQN** temos o algoritmo **Deep Dyna-Q**:

- Peng, J. et. al. implementaram o Deep Dyna-Q para treino de um agente gerador de diálogo.
- Interação com usuários reais é valiosa/custosa
- Utilização de Modelo permite treino de função valor  $Q_\theta(s, a)$  a partir de experiência simulada!





# DEEP DYNA Q: DYNA + DQN

## Algorithm 1 Deep Dyna-Q for Dialogue Policy Learning

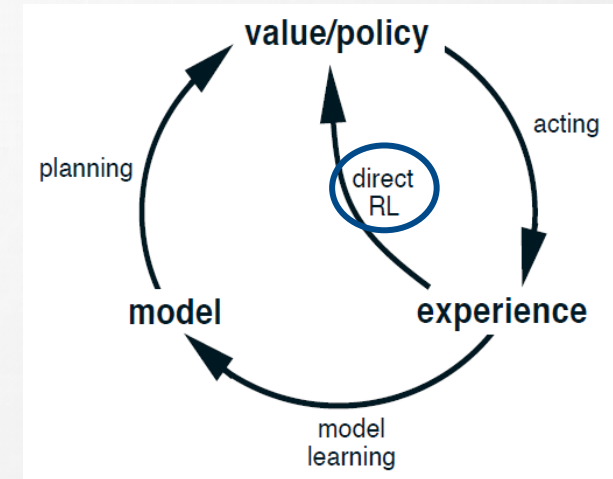
**Require:**  $N, \epsilon, K, L, C, Z$

**Ensure:**  $Q(s, a; \theta_Q), M(s, a; \theta_M)$

- 1: initialize  $Q(s, a; \theta_Q)$  and  $M(s, a; \theta_M)$  via pre-training on human conversational data
- 2: initialize  $Q'(s, a; \theta_{Q'})$  with  $\theta_{Q'} = \theta_Q$
- 3: initialize real experience replay buffer  $D^u$  using Reply Buffer Spiking (RBS), and simulated experience replay buffer  $D^s$  as empty
- 4: **for**  $u=1:N$  **do**
- 5:   *# Direct Reinforcement Learning starts*
- 6:   user starts a dialogue with user action  $a^u$
- 7:   generate an initial dialogue state  $s$
- 8:   **while**  $s$  is not a terminal state **do**
- 9:     with probability  $\epsilon$  select a random action  $a$
- 10:    otherwise select  $a = \operatorname{argmax}_{a'} Q(s, a'; \theta_Q)$
- 11:    execute  $a$ , and observe user response  $a^u$  and reward  $r$
- 12:    update dialogue state to  $s'$
- 13:    store  $(s, a, r, a^u, s')$  to  $D^u$
- 14:     $s = s'$
- 15:   **end while**
- 16:   sample random minibatches of  $(s, a, r, s')$  from  $D^u$
- 17:   update  $\theta_Q$  via Z-step minibatch Q-learning according to Equation (2)
- 18:   *# Direct Reinforcement Learning ends*
- 19:   *# World Model Learning starts*
- 20:   sample random minibatches of training samples  $(s, a, r, a^u, s')$  from  $D^u$
- 21:   update  $\theta_M$  via Z-step minibatch SGD of multi-task learning
- 22:   *# World Model Learning ends*
- 23:   *# Planning starts*
- 24:   **for**  $k=1:K$  **do**
- 25:      $t = \text{FALSE}, l = 0$
- 26:     sample a user goal  $G$
- 27:     sample user action  $a^u$  from  $G$
- 28:     generate an initial dialogue state  $s$
- 29:     **while**  $t$  is FALSE  $\wedge l \leq L$  **do**
- 30:       with probability  $\epsilon$  select a random action  $a$
- 31:       otherwise select  $a = \operatorname{argmax}_{a'} Q(s, a'; \theta_Q)$
- 32:       execute  $a$
- 33:       world model responds with  $a^u, r$  and  $t$
- 34:       update dialogue state to  $s'$
- 35:       store  $(s, a, r, s')$  to  $D^s$
- 36:        $l = l + 1, s = s'$
- 37:     **end while**
- 38:     sample random minibatches of  $(s, a, r, s')$  from  $D^s$
- 39:     update  $\theta_Q$  via Z-step minibatch Q-learning according to Equation (2)
- 40:   **end for**
- 41:   *# Planning ends*
- 42:   every  $C$  steps reset  $\theta_{Q'} = \theta_Q$
- 43: **end for**

A arquitetura de algoritmos Dyna pode ser implementada com qualquer algoritmo de Model-Free RL no passo de Planejamento. Se utilizarmos o DQN temos o algoritmo **Deep Dyna-Q**:

- 4: **for**  $n=1:N$  **do**
- 5:   *# Direct Reinforcement Learning starts*
- 6:   user starts a dialogue with user action  $a^u$
- 7:   generate an initial dialogue state  $s$
- 8:   **while**  $s$  is not a terminal state **do**
- 9:     with probability  $\epsilon$  select a random action  $a$
- 10:    otherwise select  $a = \operatorname{argmax}_{a'} Q(s, a'; \theta_Q)$
- 11:    execute  $a$ , and observe user response  $a^u$  and reward  $r$
- 12:    update dialogue state to  $s'$
- 13:    store  $(s, a, r, a^u, s')$  to  $D^u$
- 14:     $s = s'$
- 15:   **end while**
- 16:   sample random minibatches of  $(s, a, r, s')$  from  $D^u$
- 17:   update  $\theta_Q$  via Z-step minibatch Q-learning according to Equation (2)
- 18:   *# Direct Reinforcement Learning ends*



# DEEP DYNA Q: DYNA + DQN

## Algorithm 1 Deep Dyna-Q for Dialogue Policy Learning

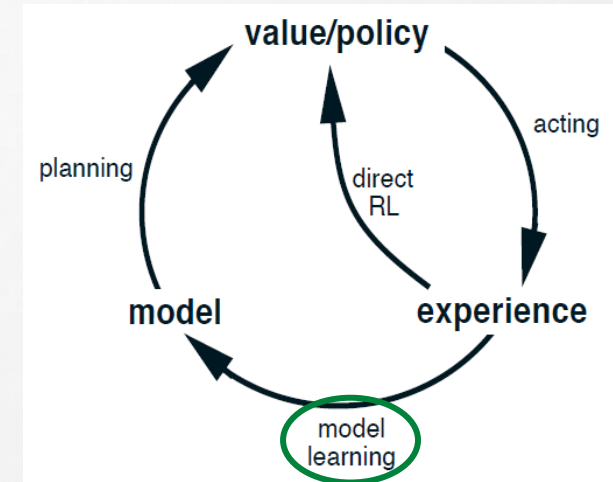
**Require:**  $N, \epsilon, K, L, C, Z$   
**Ensure:**  $Q(s, a; \theta_Q), M(s, a; \theta_M)$

- 1: initialize  $Q(s, a; \theta_Q)$  and  $M(s, a; \theta_M)$  via pre-training on human conversational data
- 2: initialize  $Q'(s, a; \theta_{Q'})$  with  $\theta_{Q'} = \theta_Q$
- 3: initialize real experience replay buffer  $D^u$  using Reply Buffer Spiking (RBS), and simulated experience replay buffer  $D^s$  as empty
- 4: **for**  $u=1:N$  **do**
- 5:   *# Direct Reinforcement Learning starts*
- 6:   user starts a dialogue with user action  $a^u$
- 7:   generate an initial dialogue state  $s$
- 8:   **while**  $s$  is not a terminal state **do**
- 9:     with probability  $\epsilon$  select a random action  $a$
- 10:    otherwise select  $a = \operatorname{argmax}_a Q(s, a; \theta_Q)$
- 11:    execute  $a$ , and observe user response  $a^u$  and reward  $r$
- 12:    update dialogue state to  $s'$
- 13:    store  $(s, a, r, a^u, s')$  to  $D^u$
- 14:     $s = s'$
- 15:   **end while**
- 16:   sample random minibatches of  $(s, a, r, s')$  from  $D^u$
- 17:   update  $\theta_Q$  via Z-step minibatch Q-learning according to Equation (2)
- 18:   *# Direct Reinforcement Learning ends*
- 19:   *# World Model Learning starts*
- 20:   sample random minibatches of training samples  $(s, a, r, a^u, s')$  from  $D^u$
- 21:   update  $\theta_M$  via Z-step minibatch SGD of multi-task learning
- 22:   *# World Model Learning ends*
- 23:   *# Planning starts*
- 24:   **for**  $k=1:K$  **do**
- 25:      $t = \text{FALSE}, l = 0$
- 26:     sample a user goal  $G$
- 27:     sample user action  $a^u$  from  $G$
- 28:     generate an initial dialogue state  $s$
- 29:     **while**  $t$  is FALSE  $\wedge l \leq L$  **do**
- 30:       with probability  $\epsilon$  select a random action  $a$
- 31:       otherwise select  $a = \operatorname{argmax}_a Q(s, a; \theta_Q)$
- 32:       execute  $a$
- 33:       world model responds with  $a^u, r$  and  $t$
- 34:       update dialogue state to  $s'$
- 35:       store  $(s, a, r, s')$  to  $D^s$
- 36:        $l = l + 1, s = s'$
- 37:     **end while**
- 38:     sample random minibatches of  $(s, a, r, s')$  from  $D^s$
- 39:     update  $\theta_{Q'}$  via Z-step minibatch Q-learning according to Equation (2)
- 40:   **end for**
- 41:   *# Planning ends*
- 42:   every  $C$  steps reset  $\theta_{Q'} = \theta_Q$
- 43: **end for**

A arquitetura de algoritmos Dyna pode ser implementada com qualquer algoritmo de Model-Free RL no passo de Planejamento. Se utilizarmos o DQN temos o algoritmo **Deep Dyna-Q**:

```

19:  # World Model Learning starts
20:  sample random minibatches of training samples
     $(s, a, r, a^u, s')$  from  $D^u$ 
21:  update  $\theta_M$  via Z-step minibatch SGD of multi-task
    learning
22:  # World Model Learning ends
  
```



# DEEP DYNA Q: DYNA + DQN

## Algorithm 1 Deep Dyna-Q for Dialogue Policy Learning

Require:  $N, \epsilon, K, L, C, Z$

Ensure:  $Q(s, a; \theta_Q), M(s, a; \theta_M)$

1: initialize  $Q(s, a; \theta_Q)$  and  $M(s, a; \theta_M)$  via pre-training on human conversational data  
 2: initialize  $Q'(s, a; \theta_{Q'})$  with  $\theta_{Q'} = \theta_Q$   
 3: initialize real experience replay buffer  $D^u$  using Reply Buffer Spiking (RBS), and simulated experience replay buffer  $D^s$  as empty

4: for  $u=1:N$  do

5:   # Direct Reinforcement Learning starts

6:   user starts a dialogue with user action  $a^u$

7:   generate an initial dialogue state  $s$

8:   while  $s$  is not a terminal state do

9:     with probability  $\epsilon$  select a random action  $a$

10:    otherwise select  $a = \arg\max_{a'} Q(s, a'; \theta_Q)$

11:    execute  $a$ , and observe user response  $a^u$  and reward  $r$

12:    update dialogue state to  $s'$

13:    store  $(s, a, r, a^u, s')$  to  $D^u$

14:     $s = s'$

15:   end while

16:   sample random minibatches of  $(s, a, r, s')$  from  $D^u$

17:   update  $\theta_Q$  via Z-step minibatch Q-learning according to Equation (2)

18:   # Direct Reinforcement Learning ends

19:   # World Model Learning starts

20:   sample random minibatches of training samples  $(s, a, r, a^u, s')$  from  $D^u$

21:   update  $\theta_M$  via Z-step minibatch SGD of multi-task learning

22:   # World Model Learning ends

23:   # Planning starts

24:   for  $k=1:K$  do

25:      $t = \text{FALSE}, l = 0$

26:     sample a user goal  $G$

27:     sample user action  $a^u$  from  $G$

28:     generate an initial dialogue state  $s$

29:     while  $t$  is FALSE  $\wedge l \leq L$  do

30:       with probability  $\epsilon$  select a random action  $a$

31:       otherwise select  $a = \arg\max_{a'} Q(s, a'; \theta_Q)$

32:       execute  $a$

33:       world model responds with  $a^u, r$  and  $t$

34:       update dialogue state to  $s'$

35:       store  $(s, a, r, s')$  to  $D^s$

36:        $l = l + 1, s = s'$

37:     end while

38:     sample random minibatches of  $(s, a, r, s')$  from  $D^s$

39:     update  $\theta_Q$  via Z-step minibatch Q-learning according to Equation (2)

40:   end for

41:   # Planning ends

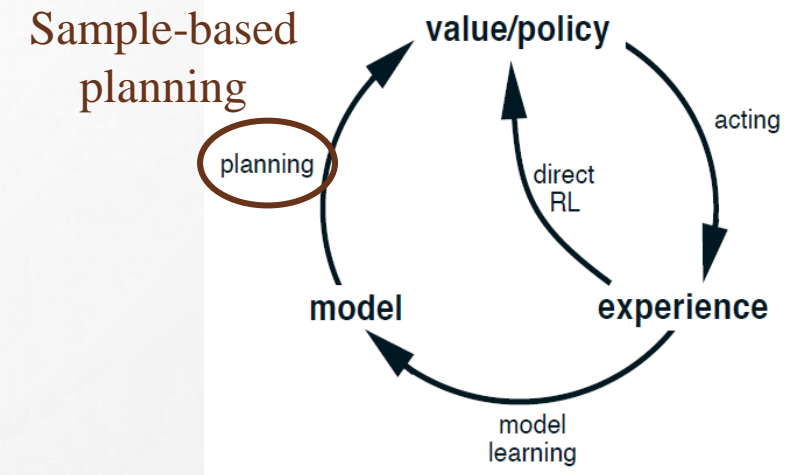
42:   every  $C$  steps reset  $\theta_{Q'} = \theta_Q$

43: end for

A arquitetura de algoritmos Dyna pode ser implementada com qualquer algoritmo de Model-Free RL no passo de Planejamento. Se utilizarmos o DQN temos o algoritmo **Deep Dyna-Q**:

```

23:  # Planning starts
24:  for k=1:K do
25:    t = FALSE, l = 0
26:    sample a user goal G
27:    sample user action a^u from G
28:    generate an initial dialogue state s
29:    while t is FALSE ^ l ≤ L do
30:      with probability ε select a random action a
31:      otherwise select a = argmax_{a'} Q(s, a'; θ_Q)
32:      execute a
33:      world model responds with a^u, r and t
34:      update dialogue state to s'
35:      store (s, a, r, s') to D^s
36:      l = l + 1, s = s'
37:    end while
38:    sample random minibatches of (s, a, r, s') from D^s
39:    update θ_Q via Z-step minibatch Q-learning according to Equation (2)
40:  end for
41:  # Planning ends
  
```



Colab Deep Dyna-Q



# IMAGINATION-AUGMENTED AGENTS (I2A)

Imagination-Augmented Agents (I2A)

# IMAGINATION-AUGMENTED AGENTS (I2A)

Métodos Model-Based funcionam bem quando:

- O modelo do ambiente é conhecido (AlphaGo, AlphaZero)
- O modelo do ambiente é facilmente aprendido (Dyna)

A partir do modelo conhecido/aprendido qualquer técnica de Planejamento pode ser utilizada para treinar a Função Valor/Política do agente.

O que fazer com **ambientes mais complexos** (observações dadas por imagens)?

- Em vez de amostrar experiências simuladas aleatórias e **prescrever** uma técnica de Planejamento (como o Dyna), o I2A gera trajetórias completas de experiência futura simulada e **aprende como combiná-las com experiência real** para treinar uma política



# IMAGINATION-AUGMENTED AGENTS (I2A)

O I2A é um algoritmo model-based que aprende como codificar e combinar experiência simulada futura (denominada *imaginação*) com experiência real (Model-Free) para treinar uma função política  $\pi_\theta$ .

- Em vez de utilizar uma técnica de planejamento fixa (Dyna), I2A aprende como melhor interpretar trajetórias simuladas.
- Experiência simulada é interpretada para construir planos implícitos.

arXiv:1707.06203v2 [cs.LG] 14 Feb 2018

## Imagination-Augmented Agents for Deep Reinforcement Learning

Théophane Weber\* Sébastien Racanière\* David P. Reichert\* Lars Boesingh  
Arthur Guez Danilo Rezende Adria Puigdomènech Badia Oriol Vinyals  
Nicolas Heess Yujia Li Ruqian Pascanu Peter Battaglia  
Demis Hassabis David Silver Dan Wierstra  
DeepMind

### Abstract

We introduce Imagination-Augmented Agents (I2As), a novel architecture for deep reinforcement learning combining model-free and model-based aspects. In contrast to most existing model-based reinforcement learning and planning methods, which prescribe how a model should be used to arrive at a policy, I2As learn to interpret predictions from a learned environment model to construct implicit plans in arbitrary ways, by using the predictions as additional context in deep policy networks. I2As show improved data efficiency, performance, and robustness to model misspecification compared to several baselines.

### 1 Introduction

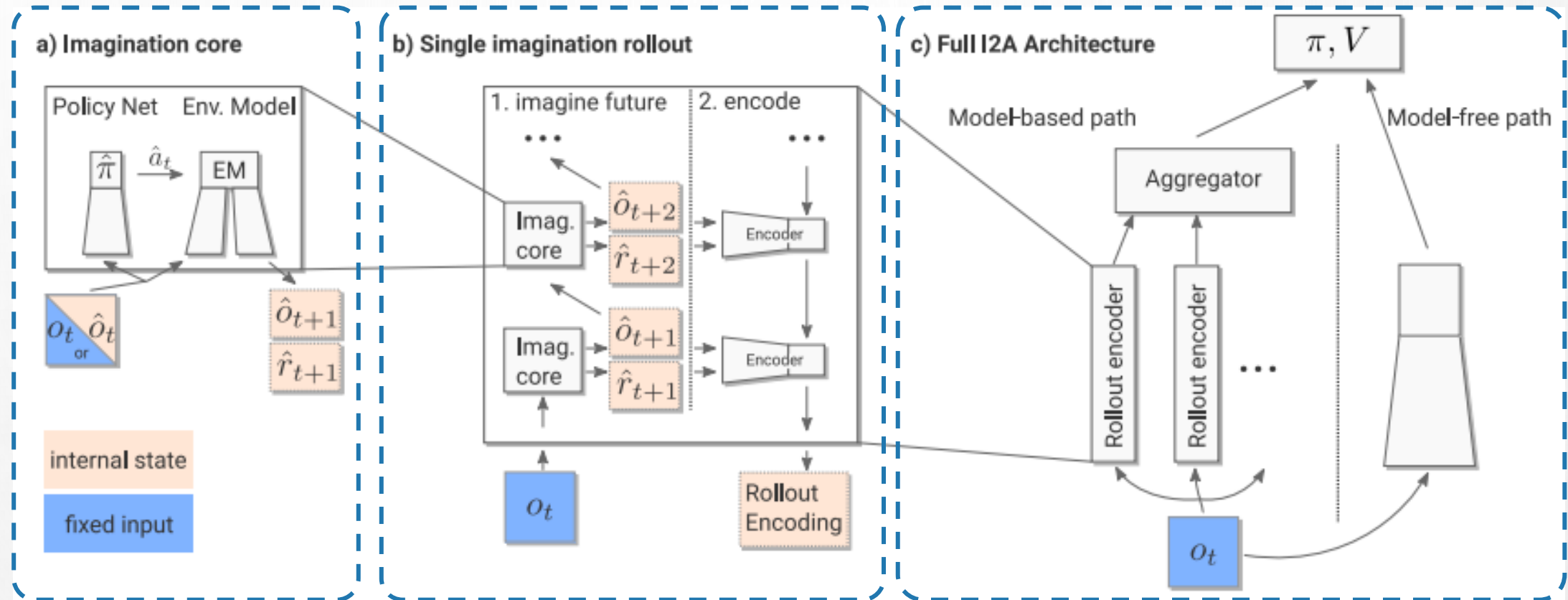
A hallmark of an intelligent agent is its ability to rapidly adapt to new circumstances and “achieve goals in a wide range of environments” [1]. Progress has been made in developing capable agents for numerous domains using deep neural networks in conjunction with model-free reinforcement learning (RL) [2–4], where raw observations directly map to values or actions. However, this approach usually requires large amounts of training data and the resulting policies do not readily generalize to novel tasks in the same environment, as it lacks the behavioral flexibility constitutive of general intelligence.

Model-based RL aims to address these shortcomings by endowing agents with a model of the world, synthesized from past experience. By using an internal model to reason about the future, here also referred to as *imagining*, the agent can seek positive outcomes while avoiding the adverse consequences of trial-and-error in the real environment – including making irreversible, poor decisions. Even if the model needs to be learned first, it can enable better generalization across states, remain valid across tasks in the same environment, and exploit additional unsupervised learning signals, thus ultimately leading to greater data efficiency. Another appeal of model-based methods is their ability to scale performance with more computation by increasing the amount of internal simulation.

The neural basis for imagination, model-based reasoning and decision making has generated a lot of interest in neuroscience [5–7]; at the cognitive level, model learning and mental simulation have been hypothesized and demonstrated in animal and human learning [8–11]. Its successful deployment in artificial model-based agents however has hitherto been limited to settings where an exact transition model is available [12] or in domains where models are easy to learn – e.g. symbolic environments or low-dimensional systems [13–16]. In complex domains for which a simulator is not available to the agent, recent successes are dominated by model-free methods [2, 17]. In such domains, the performance of model-based agents employing standard planning methods usually suffers from model errors resulting from function approximation [18, 19]. These errors compound during planning, causing over-optimism and poor agent performance. There are currently no planning

\*Equal contribution, corresponding authors: {theophane, sracaniere, reichert}@google.com.

# IMAGINATION-AUGMENTED AGENTS (I2A)



Núcleo de Imaginação:

- Política de rollout  $\hat{\pi}$
- Modelo EM (rede convolucional).

Geração de rollout futuros a partir de observação atual:

Geração de  $n$  trajetórias  $\hat{\tau}_1, \dots, \hat{\tau}_n$  a partir de núcleo de imaginação.

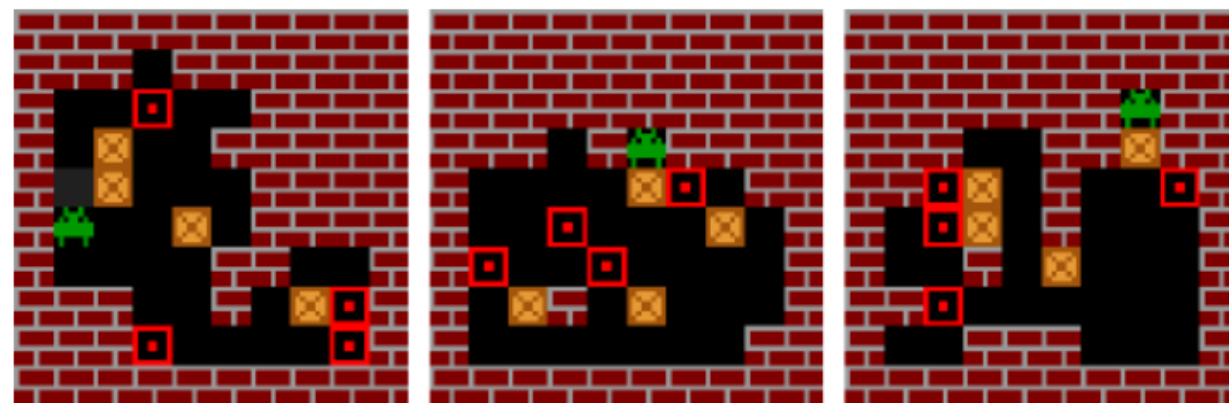
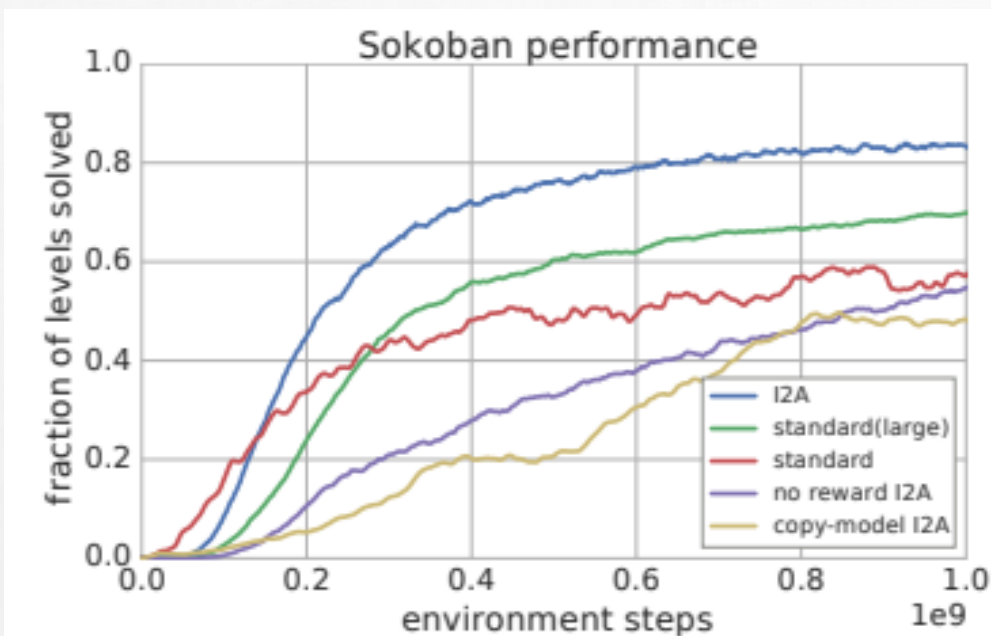
Treinamento de política:

Rollouts são codificados por meio de um *rollout encoder*  $\epsilon(\hat{\tau})$  implementado por uma rede LSTM e combinados com experiência real para treinar política  $\pi$  (rede convolucional).

# IMAGINATION-AUGMENTED AGENTS (I2A)

Agente I2A é treinado em ambiente com níveis gerados aleatoriamente de *puzzle* Sokoban, em que o agente deve empurrar caixas até posições especificadas evitando erros irreversíveis.

- Abordagem Model-Based permite que agente preveja e evite erros irreversíveis.



Weber, T.; et. al. *Imagination-Augmented Agents for Deep Reinforcement Learning*, DeepMind, 2018.

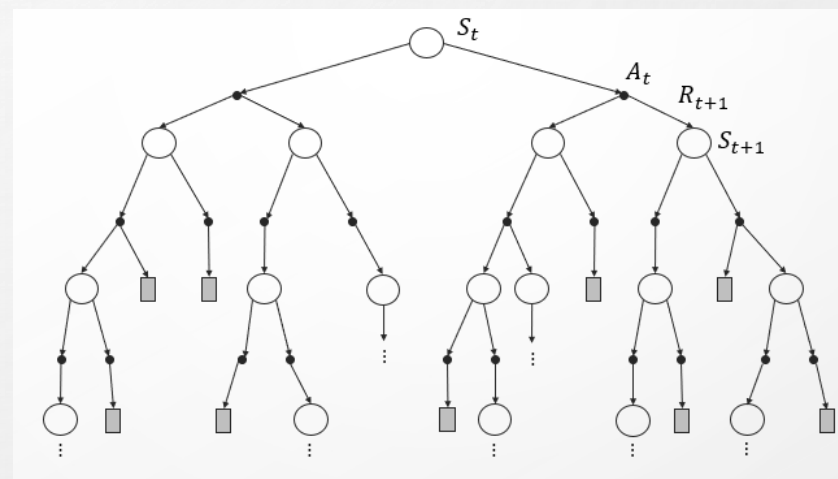
# SIMULATION-BASED SEARCH

Simulation-Based Search

# SIMULATION-BASED SEARCH

Os métodos vistos até agora fazem uso do Modelo aprendido  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$  para treinar uma função valor  $Q_w(s, a)$  ou política  $\pi_\theta(a|s)$  sobre o espaço de estados completo  $\mathcal{S}$ .

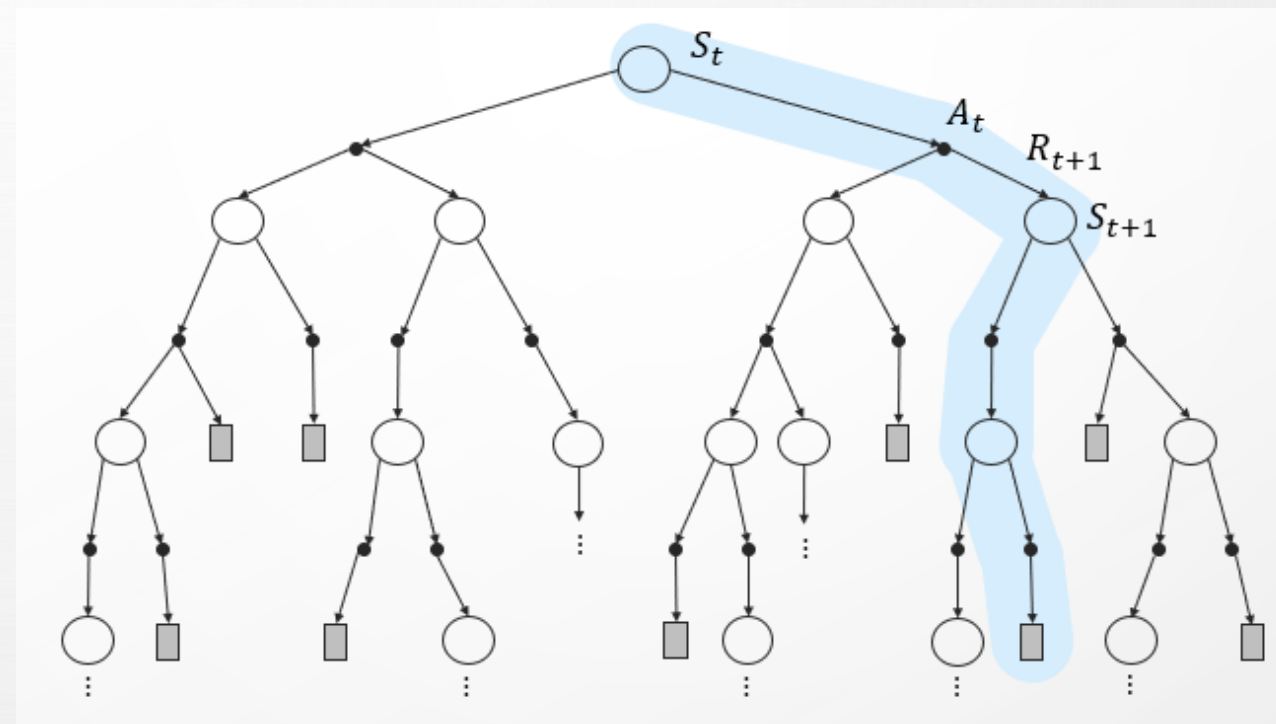
- Por que não utilizar experiência simulada para **resolver somente o sub-MDP definido a partir do estado atual  $S_t$** ?
- Simulation-Based Search faz uso do modelo para gerar experiência simulada e construir uma árvore de transições de estados a partir de  $S_t$ , aproxima  $Q(S_t, a)$  e executa, no ambiente real, a ação  $a = \arg \max_{a \in \mathcal{A}} Q(S_t, a)$ .
- Somente aplicável a estados e ações discretos.
- Melhor opção quando o modelo  $M = \{\mathcal{P}, \mathcal{R}\}$  é conhecido.  
(AlphaGo, AlphaZero)





# SIMULATION-BASED SEARCH

- Simular  $K$  episódios  $\left\{S_t^{(k)}, A_t^{(k)}, R_{t+1}^{(k)}, \dots, S_T^k\right\}_{k=1}^K \sim M_\eta$  a partir de estado atual  $S_t$
- Aplicar Model-Free RL em episódios simulados
  - Monte-Carlo Control  $\rightarrow$  Monte-Carlo Search
  - SARSA  $\rightarrow$  TD Search





# MONTE-CARLO SEARCH SIMPLES (POLÍTICA DE SIMULAÇÃO $\pi$ FIXA)

- Dado um modelo  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$  e uma política de simulação  $\pi$  fixa.
- Para cada ação  $a \in \mathcal{A}$ 
  - Simular K episódios a partir de estado atual  $s_t$  e ação  $a$ :

$$\left\{ s_t, a, R_{t+1}^{(k)}, \dots, S_T^k \right\}_{k=1}^K \sim M_\eta, \pi$$

- Avaliar ações por meio de retorno médio (Monte-Carlo Evaluation):

$$\hat{Q}(s_t, a) = \frac{1}{K} \sum_{k=1}^K G_t \approx Q_\pi(s_t, a)$$

- Selecionar e executar ação com maior valor:

$$a = \arg \max_{a \in \mathcal{A}} Q(s_t, a)$$

# MONTE-CARLO TREE SEARCH (EVALUATION)

- Dado um modelo  $M_\eta = \{\mathcal{P}_\eta, \mathcal{R}_\eta\}$
- Simular K episódios a partir de estado atual  $s_t$  usando política atual de simulação  $\pi$ :

$$\left\{ s_t, A_t^{(k)}, R_{t+1}^{(k)}, \dots, S_T^k \right\}_{k=1}^K \sim M_\eta, \pi$$

- Construir uma árvore de busca contendo estados e ações visitados.
- **Avaliar** pares estado-ação  $\hat{Q}(s, a)$  por meio de retorno médio de episódios a partir de  $s, a$

$$\hat{Q}(s, a) = \frac{1}{N(s, a)} \sum_{k=1}^K \sum_{u=t}^{T_k} 1(S_u = s, A_u = a) G_u \approx Q_\pi(s, a)$$

- Após busca, selecionar ação com maior valor:

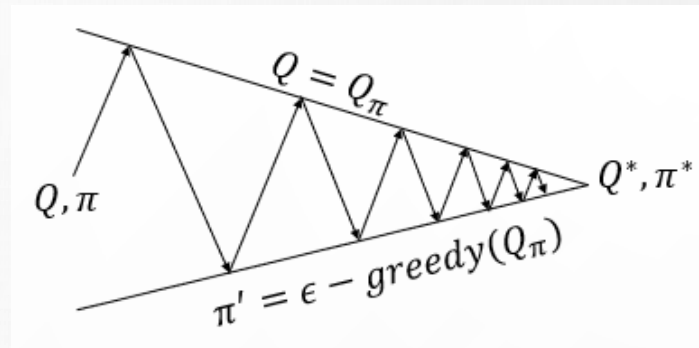
$$A_t = \arg \max_{a \in \mathcal{A}} Q(S_t, a)$$

# MONTE-CARLO TREE SEARCH

Monte-Carlo Tree Search (MCTS):

Em MCTS a **política de simulação  $\pi$  melhora**

- Cada simulação possui duas fases (in-tree, out-of-tree):
  - Tree Policy (improves): Tomar ações que maximizam  $Q(s, a)$ .
  - Default Policy (fixed): Tomar ações aleatoriamente.
- Repetir iterativamente (para cada simulação):
  - Avaliar  $Q(s, a)$  por meio de Monte-Carlo Evaluation (Policy Evaluation)
  - Melhorar política de simulação  $\pi$  ( $\epsilon$ -greedy Policy Improvement)

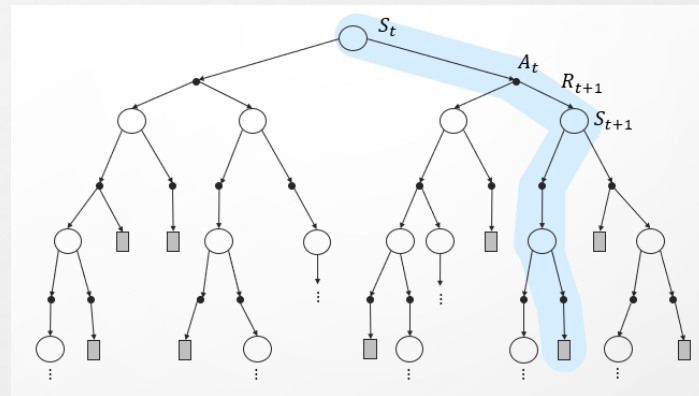


Nós já visitados na árvore (in-tree)

Nós novos (out-of-tree)

Essencialmente algoritmo de **Monte-Carlo Control** (aula 4)  
aplicado a **experiência simulada**.

Converge para árvore de busca ótima (Função Valor Ótima  $Q^*(s, a)$ )



# MONTE-CARLO TREE SEARCH

Principais Vantagens de Monte-Carlo Tree Search:

- Avalia apenas estados necessários para sub-MDP a partir de estado atual (mais eficiente).
- Aproximação de Funções Valor **reduz espaço de busca da árvore completa**:
  - Redução de largura de busca: A partir de  $Q(S_t, a)$  considera-se somente  $m$  ações de maior valor.
  - Redução de profundidade de busca: A partir de  $V(S_t)$  não é preciso simular episódios até obter retorno final  $G_t$ , o valor do estado é uma estimativa do retorno.
- Eficiência computacional (Paralelização).

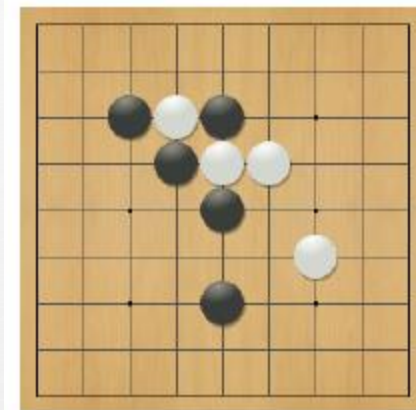
# ALPHAGO: MONTE-CARLO TREE SEARCH

AlphaGo (Monte-Carlo Tree Search)

# ALPHAGO

O jogo Go possui 2500 anos e sua resolução é considerada um grande desafio para a IA.

- Isso se deve a dimensão de seu espaço de estados e ações, levando a uma quantidade de jogos diferentes da ordem de  $10^{360}$
- Assim, métodos tradicionais de IA baseados em regras são inviáveis e métodos tradicionais de busca em árvore historicamente falharam
- **Objetivo do jogo:** Posicionar peças alternadamente de modo a capturar maior território que oponente no final do jogo. Peças cercadas por oponente são eliminadas.





# ALPHAGO: AVALIAÇÃO DE ESTADOS

Um estado representa uma configuração específica do tabuleiro. Como determinar o valor de um estado  $s$  qualquer?

- Função de Recompensa:

$$R_t = 0, \text{ for non-terminal steps } t < T$$

$$R_T = \begin{cases} 1, & \text{if Black wins} \\ 0, & \text{if White wins} \end{cases}$$

- Política  $\pi_\theta = \langle \pi_B, \pi_W \rangle$  seleciona ações para ambos jogadores (self-play)
- Função Valor:

$$V_\pi(s) = \mathbb{E}_\pi[R_T | S_t = s] = \mathbb{P}[\text{Black wins} | S_t = s]$$

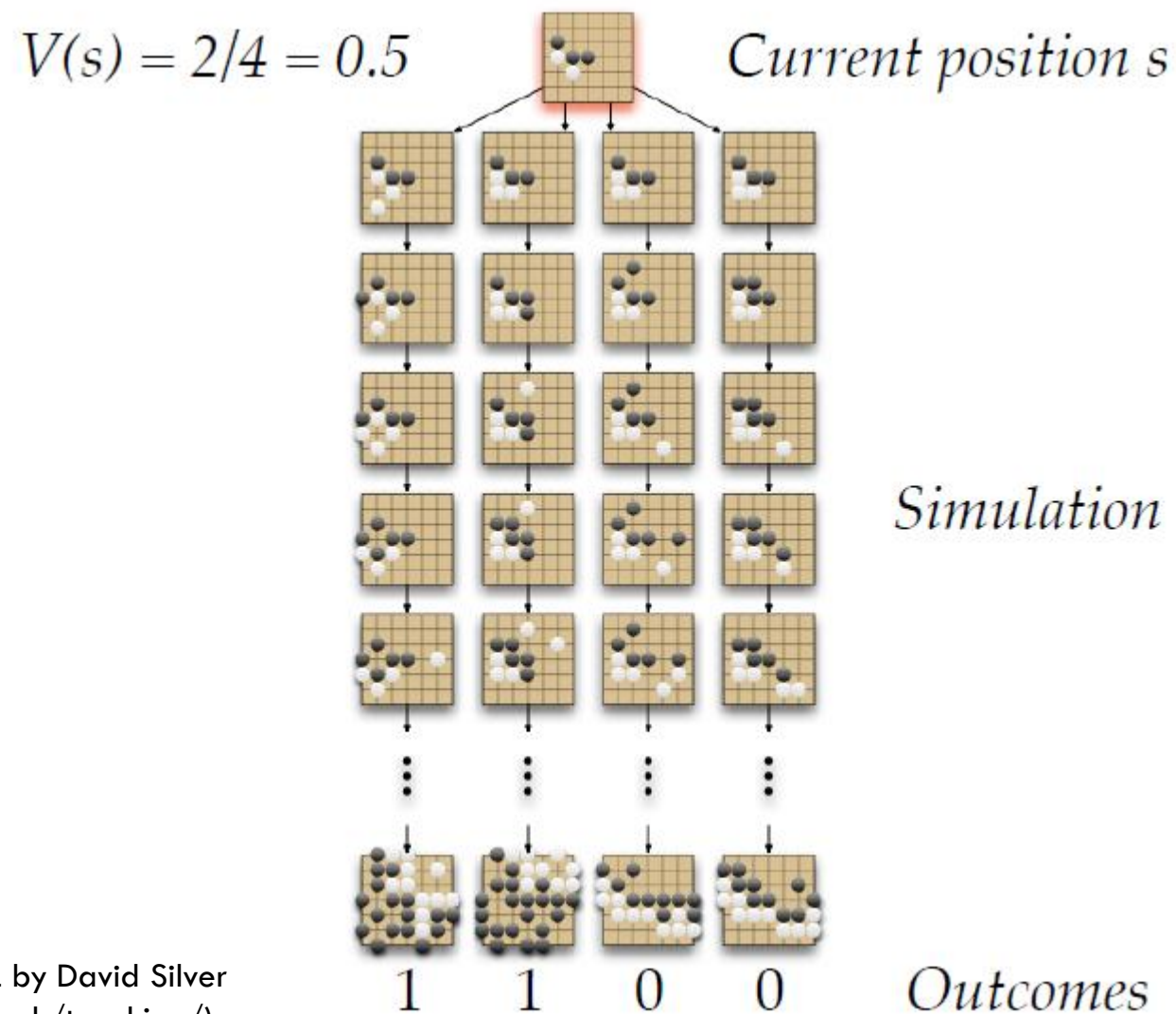
$$V^*(s) = \max_{\pi_B} \min_{\pi_W} V_\pi(s)$$

# ALPHAGO: MONTE-CARLO EVALUATION

A partir de um estado atual  $S_t$  é aplicada a política atual  $\pi_\theta(a|s)$  para simular episódios de experiência (até o final do episódio ou truncada até leaf-node e utilizando  $V(s_{leaf})$ )

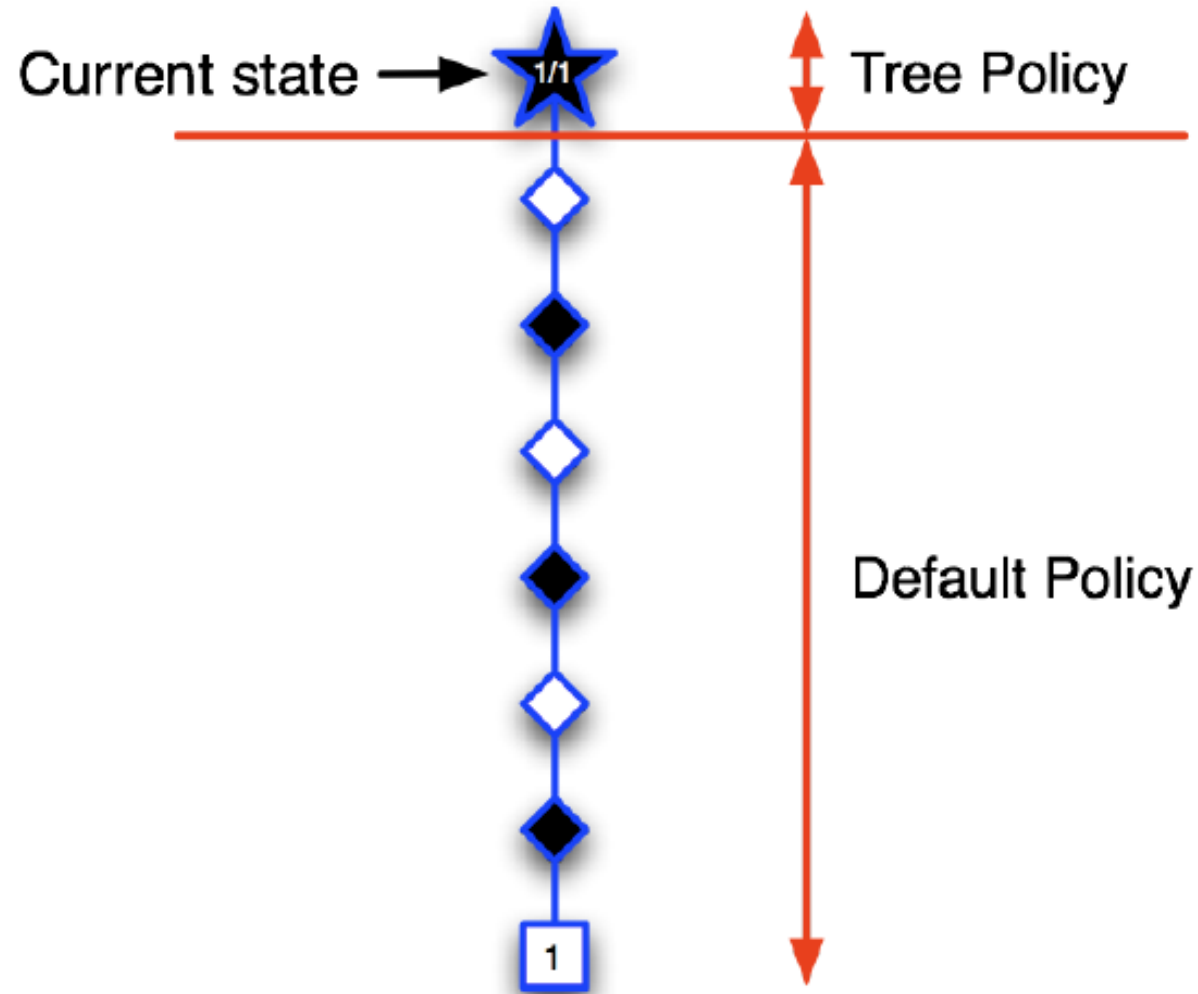
- Valor de estado atual  $V_\pi(S_t)$  pode ser estimado como a média dos retornos  $G_t$  de cada episódio simulado.

Fonte: UCL Course on RL by David Silver  
(<https://www.davidsilver.uk/teaching/>)



# ALPHAGO: MONTE-CARLO EVALUATION

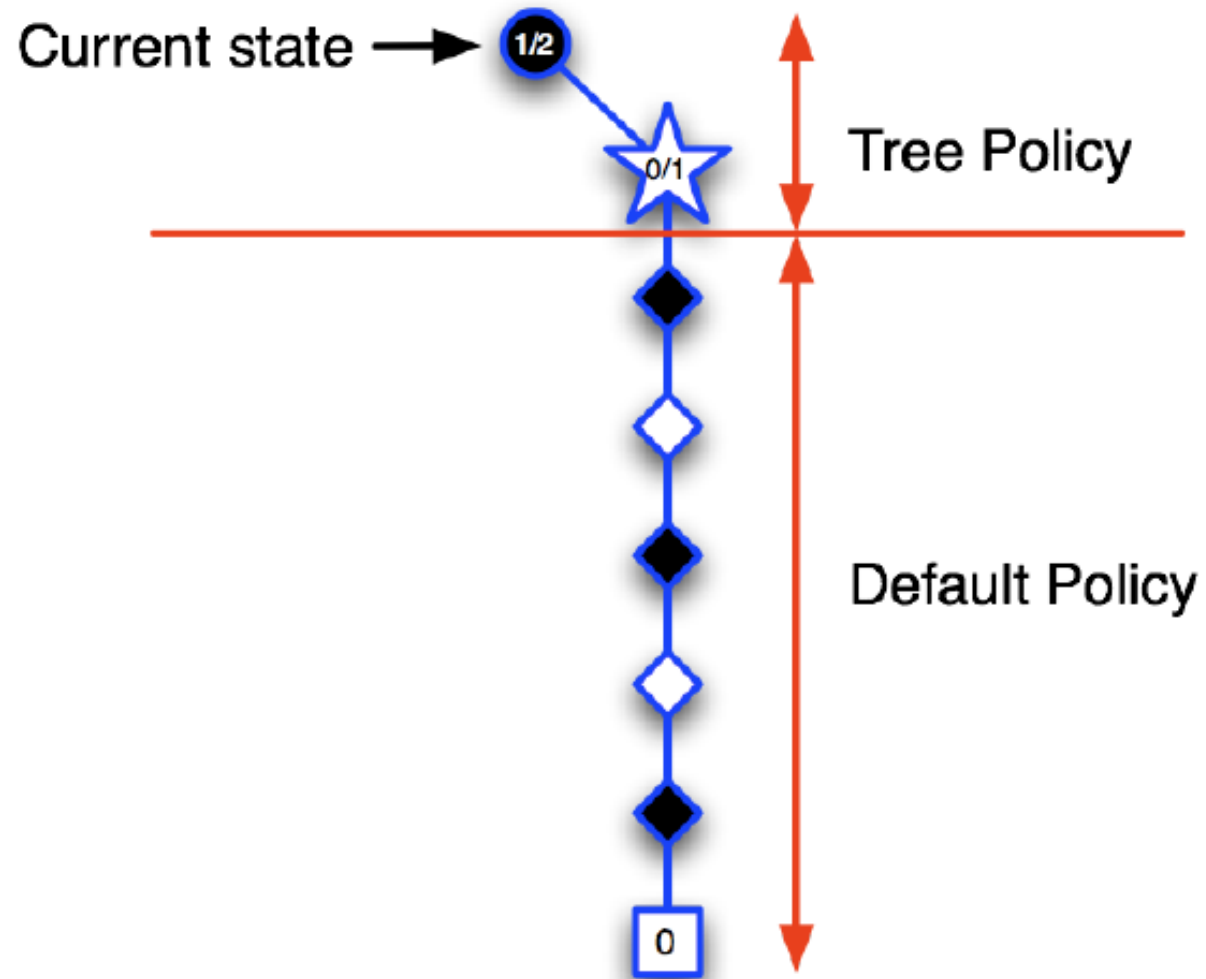
$k = 1$



Fonte: UCL Course on RL by David Silver  
(<https://www.davidsilver.uk/teaching/>)

# ALPHAGO: MONTE-CARLO EVALUATION

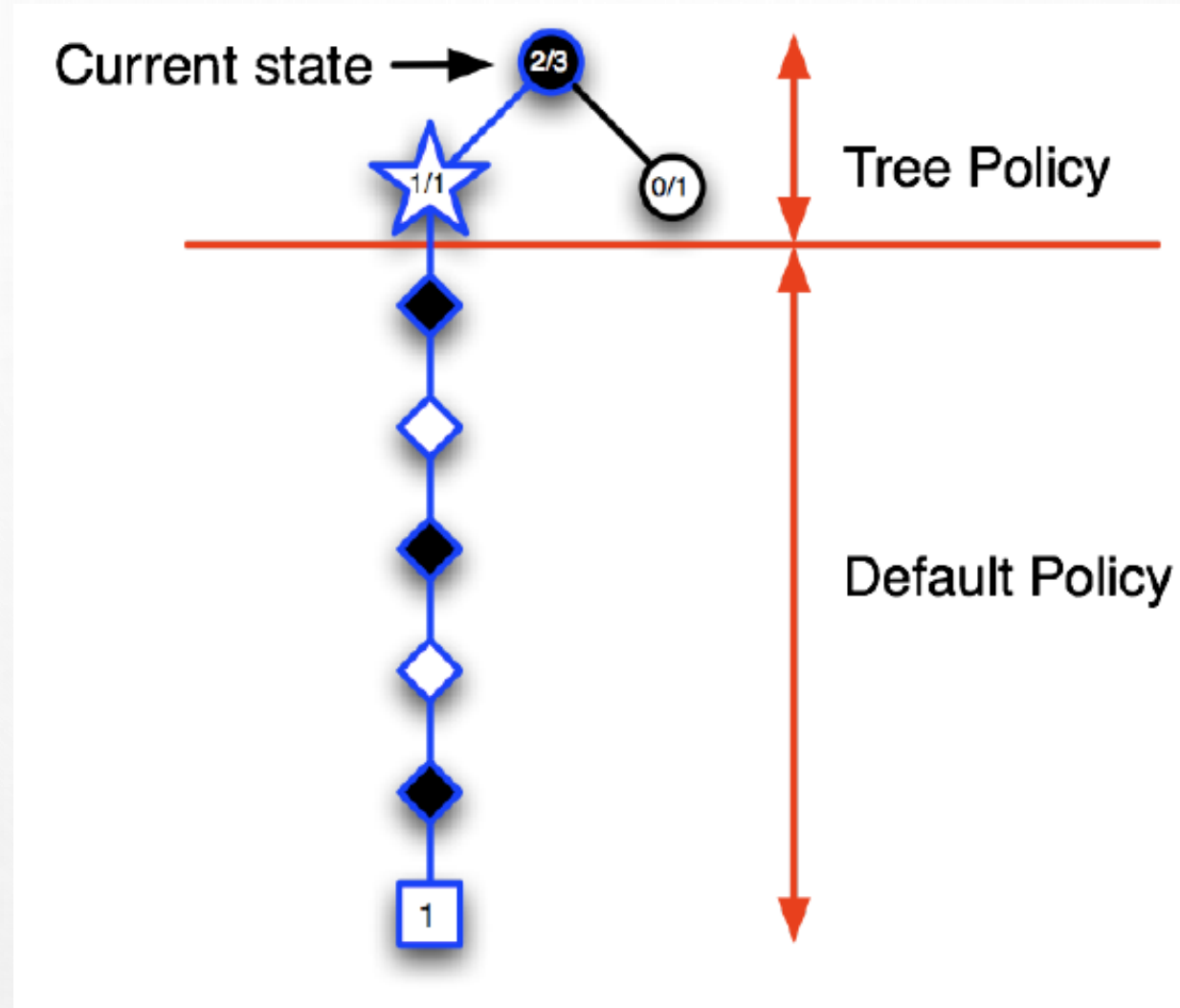
$k = 2$



Fonte: UCL Course on RL by David Silver  
(<https://www.davidsilver.uk/teaching/>)

# ALPHAGO: MONTE-CARLO EVALUATION

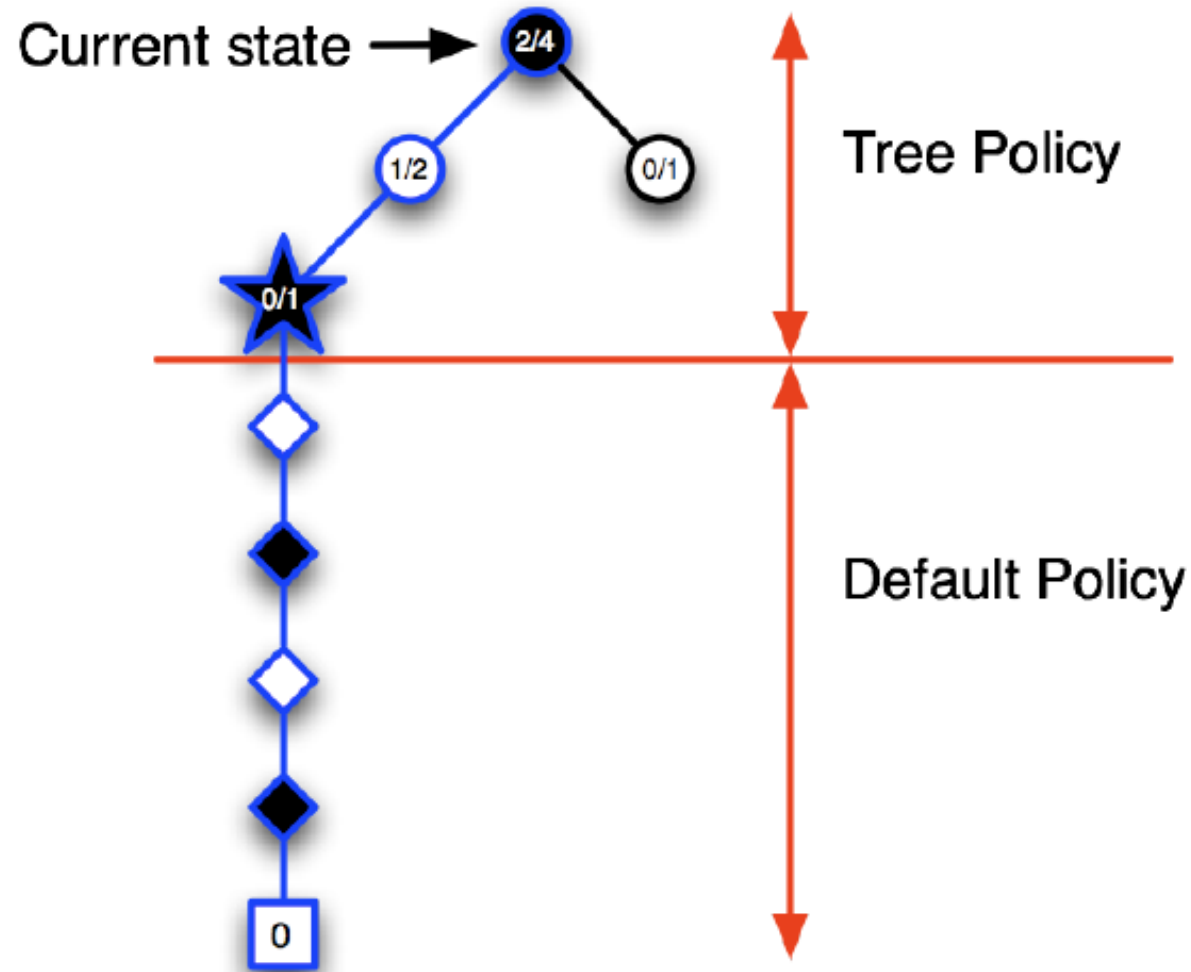
$k = 3$



Fonte: UCL Course on RL by David Silver  
(<https://www.davidsilver.uk/teaching/>)

# ALPHAGO: MONTE-CARLO EVALUATION

$k = 4$

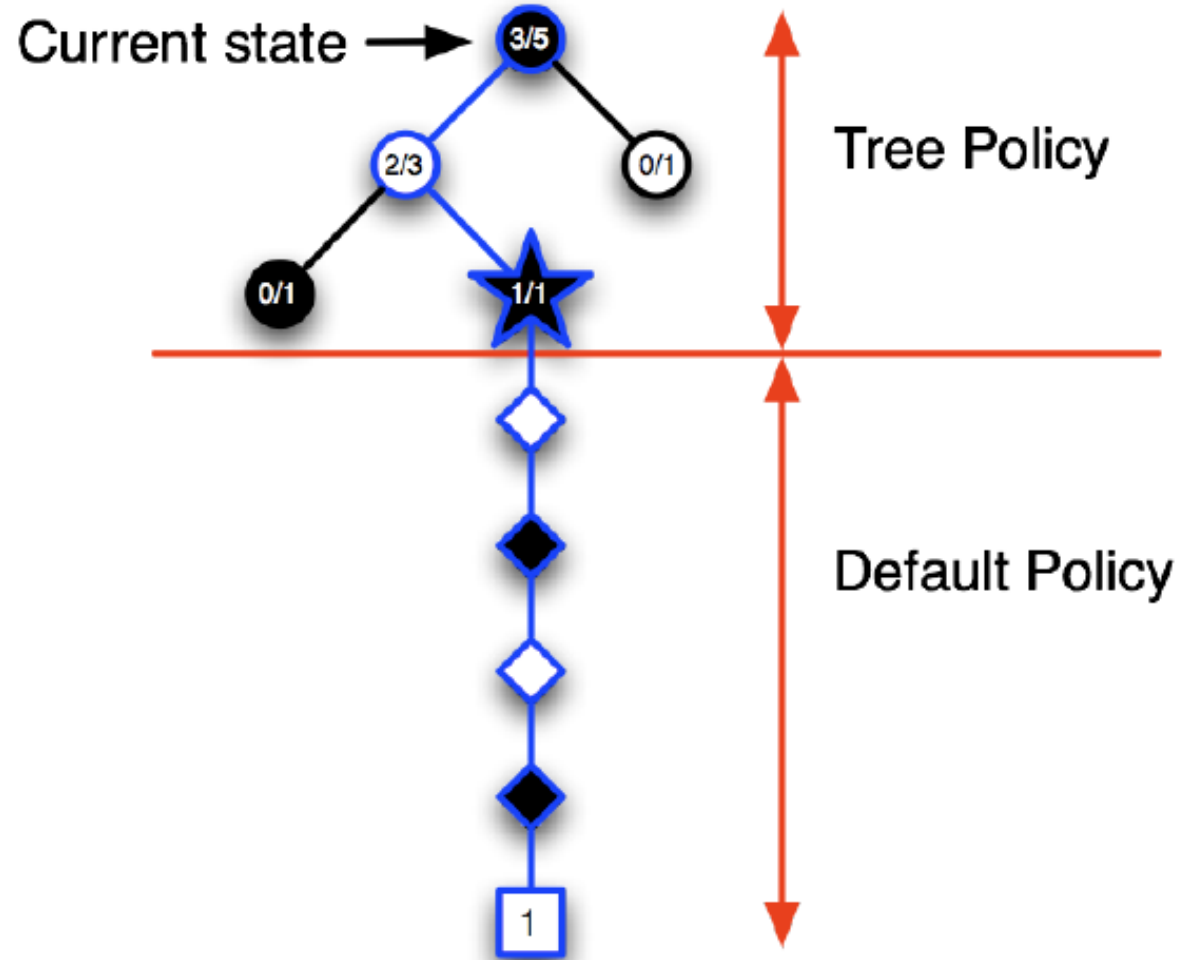


Fonte: UCL Course on RL by David Silver  
(<https://www.davidsilver.uk/teaching/>)



# ALPHAGO: MONTE-CARLO EVALUATION

$k = 5$

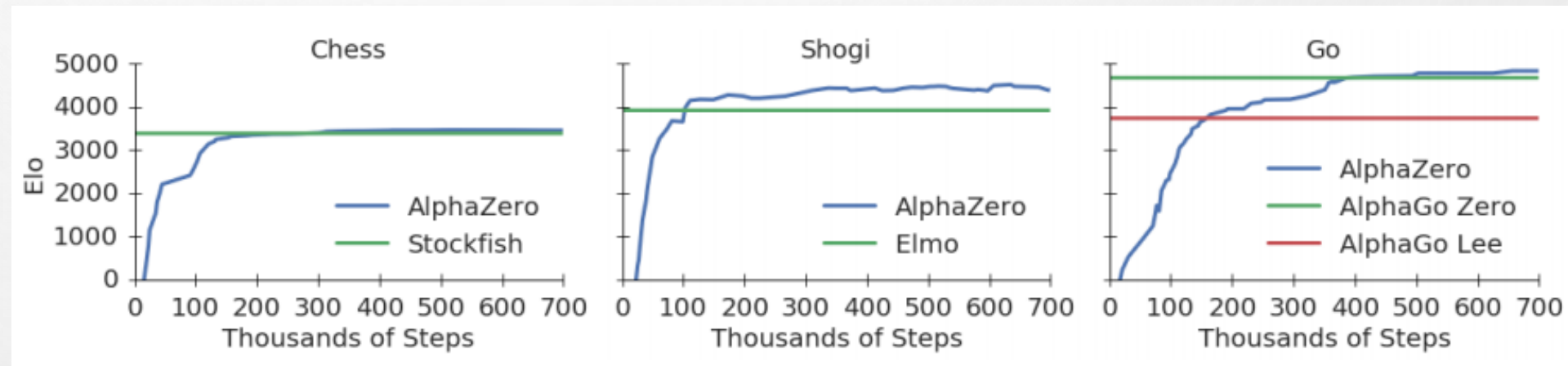


Fonte: UCL Course on RL by David Silver  
(<https://www.davidsilver.uk/teaching/>)

# ALPHAZERO

**AlphaZero** procura **generalizar o algoritmo AlphaGo** para outros ambientes e apresenta as seguintes diferenças:

- Não há treino inicial para imitar política de ações de humanos
  - Aprendizado completamente por meio de *self-play*, a partir de comportamento aleatório
- Não utiliza *hand-crafted features*, recebe apenas posições de peças em tabuleiro.
- Funções Valor e Política são combinadas em uma única rede neural.
- Busca não faz uso de *rollouts* aleatórios de Monte-Carlo, utiliza a própria política.



# CONCLUSÕES

- Métodos Model-Based são indicados quando:
  - O modelo do MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$  é conhecido (jogos/regras)
  - Interações com ambiente real são custosas e é mais eficiente treinar um modelo aproximado e aprender por simulação
- Atualmente, métodos modernos procuram aplicar Model-Based RL em problemas com estados de alta dimensão (maior complexidade de modelos):
  - Deep Dyna-Q, I2A, SOLAR, ...
- Model-Based RL é considerado o futuro dos algoritmos de Aprendizado por Reforço  
(<https://www.crowdcast.io/e/gpq66ivq/1> 22:45, 26:07, Khipu2021)

# REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction*, The MIT Press (2020). [Cp. 8, 16]
- [2] <https://julien-vitay.net/deeprl/ModelBased.html>
- [3] <https://github.com/ShangtongZhang/reinforcement-learning-an-introduction/blob/master/chapter08/maze.py>
- [4] Peng J, Williams RJ. *Efficient Learning and Planning Within the Dyna Framework*. *Adaptive Behavior*. 1993;1(4):437-454
- [5] Peng, J, et. al. *Deep Dyna-Q: Integrating Planning for Task Completion Dialogue Policy Learning*, 2018.
- [6] Nagabandi, A.; et. al. *Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning*, University of California, Berkeley, 2017.
- [7] Weber, T.; et. al. *Imagination-Augmented Agents for Deep Reinforcement Learning*, DeepMind, 2018.
- [8] <https://github.com/MiuLab/DDQ/tree/f65611c2358581bb72be61b5b389b1e3c046b73d>
- [9] [https://github.com/ray-project/ray/blob/master/rllib/contrib/alpha\\_zero/examples/train\\_cartpole.py](https://github.com/ray-project/ray/blob/master/rllib/contrib/alpha_zero/examples/train_cartpole.py)
- [10] Khipu Event Series: Reinforcement Learning, 2021. (<https://www.crowdcast.io/e/gpq66ivq/1>)
- [11] Silver, D., et. al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*, 2017

# DÚVIDAS TRABALHO $T_2$

Objetivo do trabalho:

- Criar um agente DDPG com biblioteca keras-rl2 e treiná-lo em ambiente FinRL fornecido para *portfolio management* de 15 ações entre as mais negociadas da bolsa brasileira no período entre 2008 e 2020.

Tarefas:

- A) Criação do ambiente *StockPortfolioEnv*
- B) Criação de modelos de agente  $\pi_\theta$  e crítico  $Q_w$ .
- C) Inicialização e compilação de agente DDPGAgent.
- D) Treino e visualização de resultados.





Muito obrigado a todos!

Dúvidas