Inteligência Artificial para Robótica Móvel

Aprendizado por Reforço com Aproximação de Função e Gradiente de Política

Professor: Marcos Maximo

Roteiro

- Revisão;
- Aproximação da Função Valor;
- Predição com Aproximação;
- Controle com Aproximação;
- *Deep Q-Networks* (DQN);
- *Policy Gradient* (PG).

Revisão

Aprendizado por Reforço

- Relembrando:

$$\text{MDP: } (S, A, p, r, \gamma)$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$$

Aprendizado por Reforço Baseado em Modelo

- Predição: avaliar uma política.
- Controle: determinar a política ótima.
- Métodos de programação dinâmica se baseiam nas equações de Bellman:

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) r(s,a) + \sum_{a \in A} \sum_{s' \in S} \pi(a|s) p(s'|s,a) v_\pi(s')$$

$$v_*(s) = \max_{a \in A} \left(r(s,a) + \sum_{s' \in S} p(s'|s,a) v_*(s') \right)$$

Aprendizado por Reforço Livre de Modelo (Predição)

- Monte Carlo:

$$V(S_t) = V(S_t) + \alpha(G_t - V(S_t))$$

- *Temporal-Difference* (TD):

$$V(S_t) = V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- TD de n passos:

$$V(S_t) = V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n V(S_{t+n})$$

- $TD(\lambda)$:

$$V(S_t) = V(S_t) + \alpha(G_t^{\lambda} - V(S_t))$$

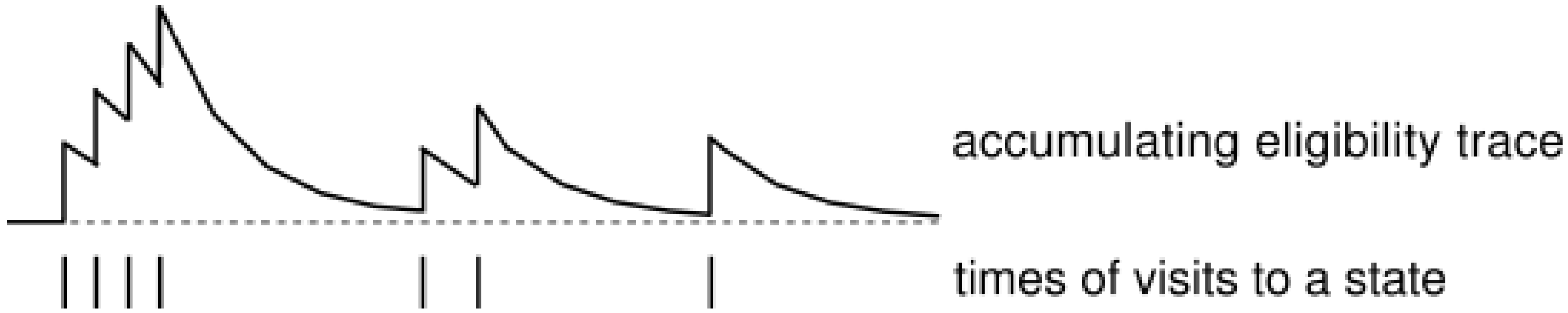
Eligibility Traces

- Forward view $TD(\lambda)$:

$$V(S_t) = V(S_t) + \alpha(G_t^\lambda - V(S_t))$$

- Backward view $TD(\lambda)$:

$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$
$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$
$$V(S_t) = V(S_t) + \alpha\delta_t E_t(S_t)$$



Aprendizado por Reforço Livre de Modelo (Controle)

- Monte Carlo (*on-policy*):

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$$

- Sarsa (*on-policy*):

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

- Q-Learning (*off-policy*):

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t)\right)$$

Aproximação da Função Valor

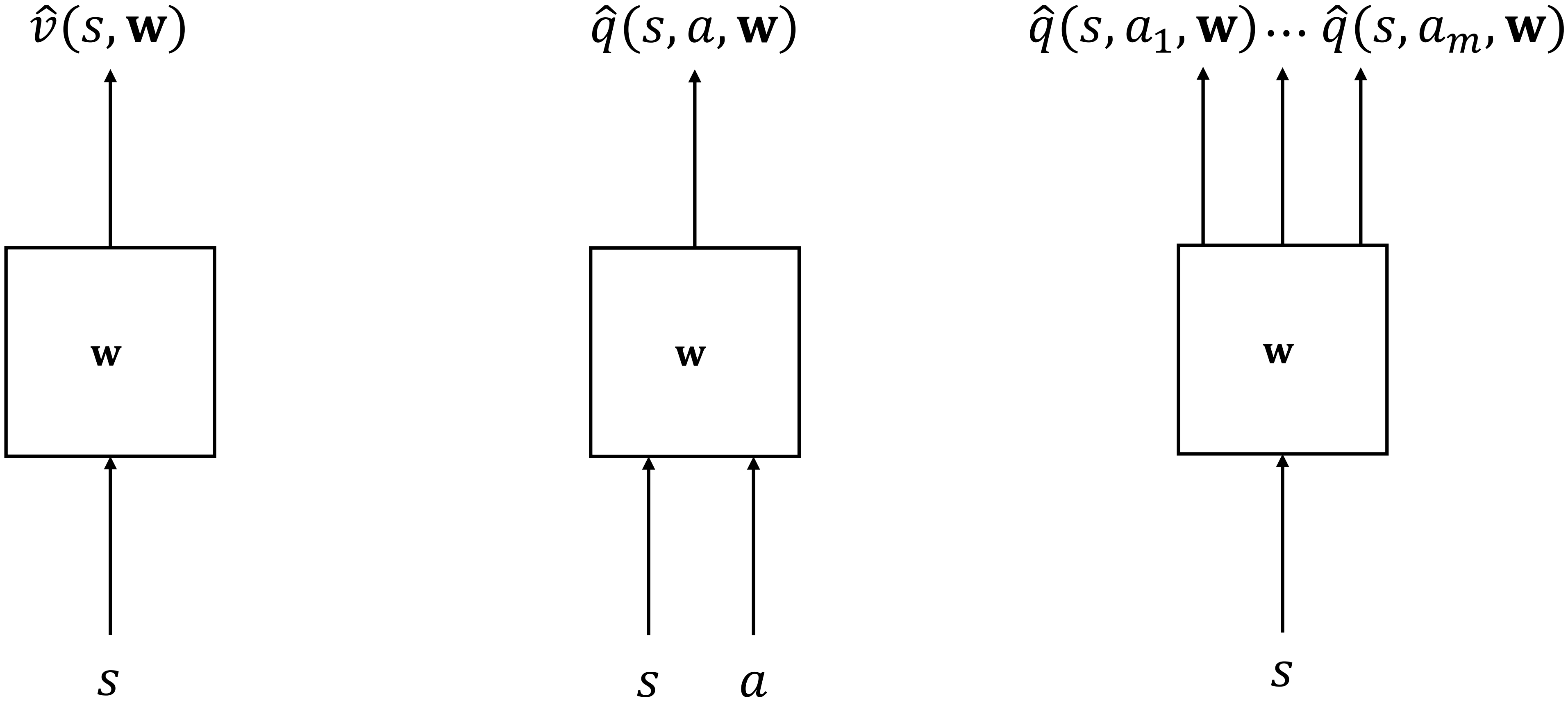
Aproximação da Função Valor

- Até então, usamos tabelas para armazenar $v(s)$ ou $q(s, a)$.
- Representação perfeita se espaço de estados e ações forem discretos.
- Porém, alguns MDPs possuem muitos estados:
  - Backgammon: 10^{20} estados.
  - Go: 10^{170} estados.
  - Futebol de robôs: espaço de estados contínuo.
- Além disso, abordagem tabular considera que estados vizinhos podem ser muito diferentes entre si…

Aproximação da Função Valor

- Solução para MDPs grandes consiste em usar aproximador de função:
$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$
- Ao invés de guardar uma tabela para cada estado ou par estado-ação, basta guardar os parâmetros $\mathbf{w}$.
- Vantagem bônus: aproximação permite generalizar aprendizado para estados próximos.

Tipos de Aproximação da Função Valor



Tipos de Aproximadores de Função

- Combinação linear de *features*.
- Função não-linear.
- Redes neurais.
- *Deep Reinforcement Learning*: redes neurais profundas.

Predição com Aproximação

Aproximação da Função Valor

- Objetivo: encontrar um vetor de parâmetros $\mathbf{w}$ que minimiza o erro quadrático em relação ao valor verdadeiro $v_\pi(s)$:

$$J(\mathbf{w}) = E_\pi\left[\left(v_\pi(s) - \hat{v}(s,\mathbf{w})\right)^{\mathbf{2}}\right] = \sum_{s\in S} \mu(s)\left(v_\pi(s) - \hat{v}(s,\mathbf{w})\right)^{\mathbf{2}}$$

em que $E_\pi[.]$ faz uma média considerando a distribuição de estados esperada ao se seguir a política π no MDP em questão.

- No caso geral de $\hat{v}(S,\mathbf{w})$, pode-se usar Descida de Gradiente para resolver esse problema de otimização:

$$\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$$

$$\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_{\mathbf{w}}J(\mathbf{w}) = \alpha E_\pi[\left(v_\pi(s) - \hat{v}(s,\mathbf{w})\right)\nabla_{\mathbf{w}}\,\hat{v}(s,\mathbf{w})]$$

Aproximação da Função Valor

- Não é possível calcular essa esperança diretamente: não temos $v_\pi(s)$.
- Porém, é possível calcular através de amostras:

$$\mathbf{w} = \mathbf{w} + \alpha\big(V_t - \hat{v}(S_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\, \hat{v}(S_t, \mathbf{w})$$

em que V_t é uma amostra de $v_\pi(S_t)$, e.g. $V_t = G_t$

- Nesse caso, realiza-se Descida de Gradiente Estocástica (SGD)!
- Como comentado antes, SGD converge para o mínimo.

Vetor de *Features*

- Representa o estado por um vetor de *features*:

$$\mathbf{x}(s) = \begin{bmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{bmatrix}$$

- Exemplos:
  - Coordenadas do robô em um tabuleiro.
  - Distâncias do robô até certos pontos do mapa.
  - Posições da bola e de todos os jogadores no campo de futebol.

Aproximação da Função Valor Linear

- Função valor é combinação linear das *features*:

$$\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w} = \sum_{j=1}^{n} x_j(s) w_j$$

- Função objetivo quadrática nos parâmetros **w:**

$$J(\mathbf{w}) = E_\pi[(v_\pi(s) - \mathbf{x}(s)^T \mathbf{w})^2]$$

- SGD converge para ótimo global.
- Regra de atualização:

$$\Delta\mathbf{w} = \alpha\big(V_t - \hat{v}(S_t, \mathbf{w})\big)\mathbf{x}(S_t)$$

Features de Tabela

- Perceba que a abordagem tabular é um caso especial da aproximação de função valor linear.

$$\mathbf{x}(s) = \begin{bmatrix} \mathbf{1}(s = s_1) \\ \vdots \\ \mathbf{1}(s = s_n) \end{bmatrix}$$

$$\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} \mathbf{1}(s = s_j) w_j$$

$$\hat{v}(s_i, \mathbf{w}) = w_i$$

Aproximação da Função Valor

$$\mathbf{w} = \mathbf{w} + \alpha\big(V_t - \hat{v}(S_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\, \hat{v}(S_t, \mathbf{w})$$

- Várias formas de conseguir a amostra V_t.
- Monte Carlo (MC):

$$\mathbf{w} = \mathbf{w} + \alpha\big(G_t - \hat{v}(S_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\, \hat{v}(S_t, \mathbf{w})$$

- TD(0):

$$\mathbf{w} = \mathbf{w} + \alpha\big(R_{t+1} + \gamma\, \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\, \hat{v}(S_t, \mathbf{w})$$

- $TD(\lambda)$:

$$\mathbf{w} = \mathbf{w} + \alpha\left(G_t^{\lambda} - \hat{v}(S_t, \mathbf{w})\right)\nabla_{\mathbf{w}}\, \hat{v}(S_t, \mathbf{w})$$

MC com Aproximação da Função Valor

$$\mathbf{w} = \mathbf{w} + \alpha\big(G_t - \hat{v}(S_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\, \hat{v}(S_t, \mathbf{w})$$

- G_t é amostra não-enviesada, mas ruidosa de $v_\pi(S_t)$.
- No fundo, é equivalente a aplicar aprendizado supervisionado em dados "anotados":

$$< S_1, G_1 >, < S_2, G_2 >, \ldots, < S_T, G_T >$$

- Alta variância, logo precisa de muitas amostras.
- Se aproximação linear, então converge para ótimo global.
- Em geral, converge para ótimo local.

TD(0) com Aproximação da Função Valor

$$\mathbf{w} = \mathbf{w} + \alpha\big(R_{t+1} + \gamma\ \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\ \hat{v}(S_t, \mathbf{w})$$
$$= \alpha\delta_t\nabla_{\mathbf{w}}\ \hat{v}(S_t, \mathbf{w})$$

- Alvo TD $R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w})$ é amostra enviesada (mas menos ruidosa) de $v_\pi(S_t)$.
- Se aproximação linear, então converge para (próximo do) ótimo global.
- Pode divergir se usar aproximação não-linear.

$TD(\lambda)$ com Aproximação da Função Valor

$$\mathbf{w} = \mathbf{w} + \alpha\left(G_t^\lambda - \hat{v}(S_t,\mathbf{w})\right)\nabla_{\mathbf{w}}\,\hat{v}(S_t,\mathbf{w})$$

- λ-Return G_t^λ é amostra enviesada de $v_\pi(S_t)$.
- *Backward view* $TD(\lambda)$ com aproximação da função valor:

$$\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})$$
$$E_t = \gamma\lambda E_{t-1} + \mathbf{x}(S_t)$$
$$\Delta\mathbf{w} = \alpha\delta_t E_t$$

Controle com Aproximação

Aproximação da Função Ação-Valor

- Semelhante à aproximação da função valor:

$$J(\mathbf{w}) = E_\pi\left[\left(q_\pi(S,A) - \hat{q}(S,A,\mathbf{w})\right)^{\mathbf{2}}\right]$$

- Usando SGD:

$$\mathbf{w} = \mathbf{w} + \alpha\left(q_t - \hat{q}(S_t,A_t,\mathbf{w})\right)\nabla_{\mathbf{w}}\,\hat{q}(S_t,A_t,\mathbf{w})$$

Aproximação da Função Ação-Valor

$$\mathbf{w} = \mathbf{w} + \alpha\big(q_t - \hat{q}(S_t, A_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\, \hat{q}(S_t, A_t, \mathbf{w})$$

- Várias formas de conseguir a amostra q_t.
- Monte Carlo (MC):

$$\mathbf{w} = \mathbf{w} + \alpha\big(G_t - \hat{q}(S_t, A_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\, \hat{q}(S_t, A_t, \mathbf{w})$$

- TD(0):

$$\mathbf{w} = \mathbf{w} + \alpha\big(R_{t+1} + \gamma\, \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\, \hat{q}(S_t, A_t, \mathbf{w})$$

- $TD(\lambda)$:

$$\mathbf{w} = \mathbf{w} + \alpha\left(Q_t^{\lambda} - \hat{q}(S_t, A_t, \mathbf{w})\right)\nabla_{\mathbf{w}}\, \hat{q}(S_t, A_t, \mathbf{w})$$

- *Backward view* $TD(\lambda)$.

Iteração de Política com Aproximação

- Iniciar com aproximação para função ação-valor e política ε-greedy.
- Loop:
  - Avaliar a política:
$$\mathbf{w} = \mathbf{w} + \alpha\big(q_t - \hat{q}(S_t, A_t, \mathbf{w})\big)\nabla_{\mathbf{w}}\, \hat{q}(S_t, A_t, \mathbf{w})$$
  - Aprimorar política ε-*greedy*.
$$\pi'(a|s) = \varepsilon\text{-}\textit{greedy}(\hat{q}(s, a, \mathbf{w}))$$

Convergência dos Algoritmos de Predição

On/Off-Policy	**Algoritmo**	**Tabular**	**Linear**	**Não-linear**
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	X
	$TD(\lambda)$	✓	✓	X
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	X	X
	$TD(\lambda)$	✓	X	X

Convergência dos Algoritmos de Predição

- Lembrando regra de atualização de SGD com TD:

$$\Delta \mathbf{w} = \alpha E_\pi\left[\left(v_\pi(s) - \hat{v}(s, \mathbf{w})\right)\nabla_{\mathbf{w}}\, \hat{v}(s, \mathbf{w})\right]$$

$$\Delta \mathbf{w} = \alpha\left(R_{t+1} + \gamma\, \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})\right)\nabla_{\mathbf{w}}\, \hat{v}(S_t, \mathbf{w})$$

- Por conta disso, TD não é gradiente de nenhuma função objetivo.
- Por isso, TD tem convergência comprometida.
- Há uma versão recente chamada *Gradient TD* que resolve isso e tem convergência garantida em todos os casos para predição.

Convergência dos Algoritmos de Controle

Algoritmo	Tabular	Linear	Não-linear
MC	✓	✓*	X
Sarsa	✓	✓*	X
Q-Learning	✓	X	X
Gradient Q-Learning	✓	✓	X

- * oscila em torno do ótimo, sem convergir.

Deep Q-Networks (DQN)

Deep Q-Networks (DQN)

- Trabalho famoso do DeepMind que atingiu desempenho super-humano em jogos de Atari.

Deep Q-Networks (DQN)

- Estabiliza Q-Learning com redes neurais profundas.
- Usou CNN para estimar função ação-valor $q(s, a)$.
- Entrada são os pixels da tela.
- Saída é $\hat{q}(s, a)$ para 18 ações (posições da alavanca e botões).
- Rede *end-to-end*.
- Recompensa é mudança de pontuação em cada passo.
- Entrada na verdade consiste dos 4 últimos *frames* (permite estimar velocidade).

Deep Q-Networks (DQN)

- DQN usa *experience replay* e *fixed Q-targets.*
- Experience replay: armazenar transições $(S_t, A_t, R_{t+1}, S_{t+1})$ em um *replay buffer* D.
- Numa atualização do treinamento, amostrar *mini-batch* aleatório de transições (s, a, r, s') de D.
- Dados fornecidos para treinamento da rede não são mais sequenciais.
- Isso ajuda muito a estabilizar o treinamento! ♣♦♥♠
- Explicação intuitiva: convergência de redes neurais garantida apenas se dados forem descorrelacionados.

Deep Q-Networks (DQN)

- *Fixed Q-targets* se baseia em criar uma cópia da rede durante o treinamento e fixar seus pesos em $\mathbf{w}^{-}$.
- Assim, treina-se a rede usando o seguinte *loss*:

$$L_i(\mathbf{w}) = E_{s,a,r,s' \sim D_i}\left[\left(r + \gamma \max_{a' \in A} \hat{q}_{\mathbf{w}^-}(s', a') - \hat{q}_{\mathbf{w}}(s, a)\right)^2\right]$$

- Explicação intuitiva: redes neurais tem problema de convergência se dados são não-estacionários.

Policy Gradient (PG)

Policy Gradient (PG)

- Os métodos que vimos até então são baseados em estimar a função valor ou a função ação-valor.
- Então, a política é gerada a partir da função (ação-)valor, geralmente usando ε-greedy.
- Vimos aproximar a função (ação-)valor usando parametrização:

$$v_\pi(s) \approx \hat{v}(s, \mathbf{w})$$
$$q_\pi(s, a, \mathbf{w}) \approx \hat{q}(s, a, \mathbf{w})$$

Policy Gradient (PG)

- Aproximação de função permite trabalhar com espaço de estados contínuo.
- Porém, espaço de ações ainda continua discreto.
- Abordagem policy gradient foca em parametrizar a política:
$$\pi_{\boldsymbol{\theta}}(s, a) = P(A_t = a | S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta})$$
- *Deep Learning*: política vai ser rede neural também.

RL Baseado em Valor e Baseado em Política

- Baseado em valor:
  - Aprender função valor.
  - Política implícita (e.g. ε-greedy).
- Baseado em política:
  - Nenhuma função valor.
  - Aprender política.
- *Actor-Critic*:
  - Aprender função valor (*critic*).
  - Aprender política (*actor*).

Policy Gradient (PG)

- Vantagens:
  - Melhores propriedades de convergência.
  - Efetivo em espaços de ações de alta dimensionalidade ou contínuos.
  - Pode aprender políticas estocásticas.
- Desvantagens:
  - Tipicamente converge para ótimo local.
  - Avaliar política tem alta variância.

Policy Gradient (PG)

- Costuma-se usar Subida de Gradiente (*Gradient Ascent*) em cima de função objetivo $J(\boldsymbol{\theta})$.
- Tarefas episódicas: $J(\boldsymbol{\theta}) = J_1(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(S_0) = E_{\pi_{\boldsymbol{\theta}}}[G_0]$.
- Tarefas continuadas:

$$J(\boldsymbol{\theta}) = J_{avg-V}(\boldsymbol{\theta}) = \sum_{s \in S} \mu_{\pi_{\boldsymbol{\theta}}}(s) v_{\pi_{\boldsymbol{\theta}}}(s)$$

$$J(\boldsymbol{\theta}) = J_{avg-R}(\boldsymbol{\theta}) = \sum_{s \in S} \mu_{\pi_{\boldsymbol{\theta}}}(s) \sum_{a \in A} \pi_{\boldsymbol{\theta}}(s,a) r(s,a)$$

- $\mu(s)$ é a distribuição dos estados da cadeia de Markov seguindo $\pi_{\boldsymbol{\theta}}$.

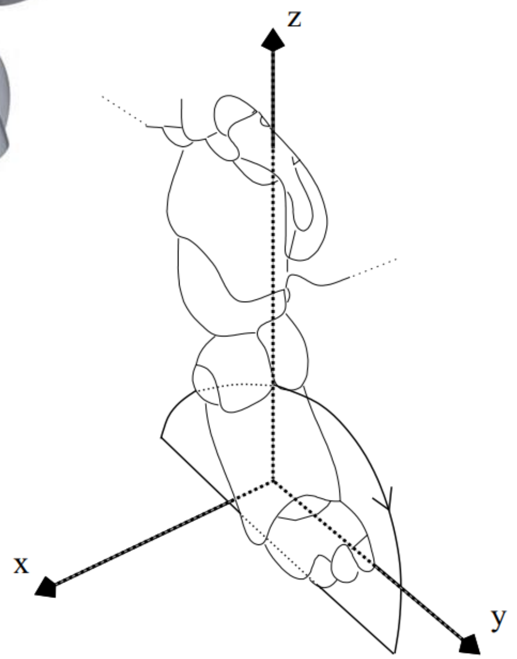
Policy Gradient (PG)

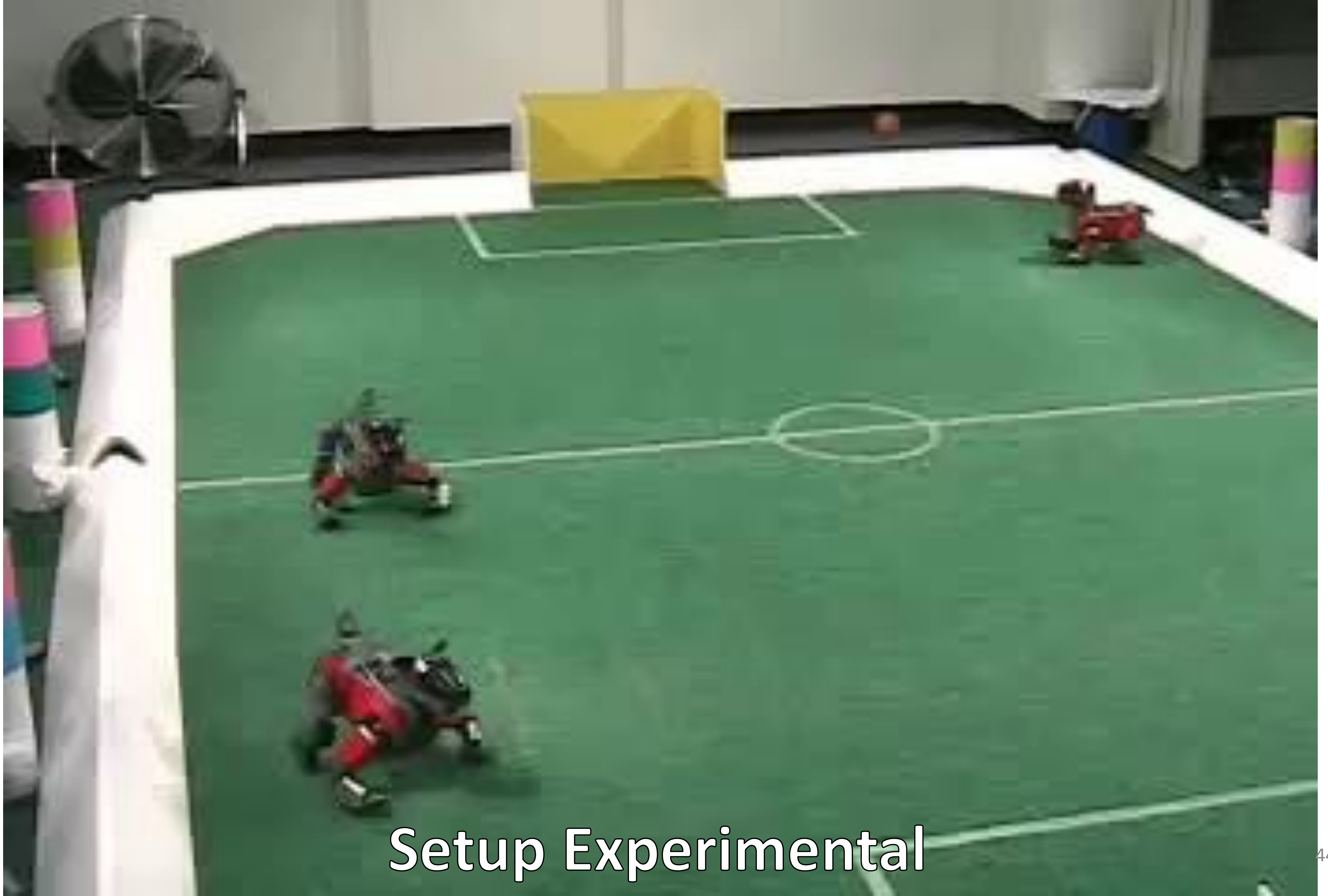
- RL baseado em política é um problema de otimização:
$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, J(\boldsymbol{\theta})$$
- Pode-se usar algoritmos que não são baseados em gradiente:
  - *Hill climbing*, *Nelder-mead*, PSO, CMA-ES etc.
- Otimização baseada em gradiente costuma ser mais eficiente.

Estudo de Caso: Aprendizado de Caminhada do AIBO (Nath Kohl e Peter Stone, 2004)

- Aprendizado em robô real!
- *Policy Gradient* calculando gradiente numericamente.
- Episódio: atravessar o campo.
- Usa *landmarks* para se localizer.
- 1000 episódios até aprender.
- Política: parâmetros da trajetória (meia elipse) do pé.

$$l, h, x, y$$





Setup Experimental

Inicial

Final

Policy Gradient Theorem

- Para qualquer política $\pi_{\boldsymbol{\theta}}(s, a)$.
- Para qualquer uma das funções objetivo: $J = J_1$, $J = J_{avg-R}$ ou $J = \frac{1}{1-\gamma} J_{avg-V}$.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto E_{\pi_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s, a) q_{\pi_{\boldsymbol{\theta}}}(s, a)\right]$$

- $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s, a)$ é chamada ***score function***.
- $\log \pi_{\boldsymbol{\theta}}(s, a)$ aparece devido ao seguinte truque:

$$\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s, a) = \pi_{\boldsymbol{\theta}}(s, a) \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s, a)}{\pi_{\boldsymbol{\theta}}(s, a)} = \pi_{\boldsymbol{\theta}}(s, a) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s, a)$$

REINFORCE (Monte Carlo PG)

- Usando Q_t como amostra para $q_{\pi_{\boldsymbol{\theta}}}(s, a)$:

$$\Delta\boldsymbol{\theta} = \alpha \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S_t, A_t)\, G_t$$

REINFORCE:

Inicializar $\boldsymbol{\theta}$ arbitrariamente.

Loop (para cada episódio):

Gerar trajetória $S_0, A_0, R_1, S_1, A_1, \dots, S_{T-1}, A_{T-1}, R_T \sim \pi_{\boldsymbol{\theta}}$

Loop (para cada passo do episódio $t = 0,1, \dots, T-1$):

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha\gamma^t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S_t, A_t)\, G_t$$

Política Gaussiana

- Um tipo de política comumente usada em PG é a gaussiana:
$$a \sim N(\mu(s), \sigma^2)$$
- *Score function*:
$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s, a) = \frac{(a - \mu(s))\nabla_{\boldsymbol{\theta}}\mu(s)}{\sigma^2}$$
- $\mu(s)$ pode ser dado por uma combinação linear de *features*:
$$\mu(s) = \boldsymbol{\phi}(s)^T \boldsymbol{\theta}$$
- Ou pode ser a saída de uma rede neural!
- Pode-se manter σ fixo ou parametrizá-lo (para aprender σ).

Reduzindo a Variância com um *Critic*

- Monte Carlo tem alta variância.
- Usar um critic para estimar a função ação-valor:

$$\hat{q}_{\mathbf{w}}(s,a) \approx q_{\pi_{\boldsymbol{\theta}}}(s,a)$$

- Actor-critic mantém dois conjuntos de parâmetros:
  - *Critic* atualiza parâmetros $\mathbf{w}$ da aproximação da função ação-valor.
  - *Actor* atualiza parâmetros $\boldsymbol{\theta}$ na direção sugerida pelo *critic*.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto E_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s,a) \hat{q}_{\mathbf{w}}(s,a)]$$
$$\Delta\boldsymbol{\theta} = \alpha \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s,a) \hat{q}_{\mathbf{w}}(s,a)$$

Estimando a Função Ação-Valor

- Como antes, tem-se várias formas de estimar a função valor.
- Monte Carlo.
- *Temporal-Difference learning*.
- $TD(\lambda)$.

Action-Value Actor-Critic

Inicializar $\boldsymbol{\theta}$ e $\mathbf{w}$ arbitrariamente.

Loop (para cada episódio):

  Inicializar S

  Escolher ação $A \sim \pi_{\boldsymbol{\theta}}$

  Loop (para cada passo do episódio $t = 0,1,\ldots,T-1$):

    Tomar ação A, observar S', R

    Escolher ação $A' \sim \pi_{\boldsymbol{\theta}}$

    $\delta = R + \gamma\ \hat{q}_{\mathbf{w}}(S', A') - \hat{q}_{\mathbf{w}}(S, A)$

    $\mathbf{w} = \mathbf{w} + \alpha_{\boldsymbol{w}} \delta \nabla \hat{q}_{\mathbf{w}}(S, A)$

    $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \gamma^{t} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S_t, A_t)\ \hat{q}_{\mathbf{w}}(S, A)$

    $S = S'; A = A'$

  Até fim do episódio

Reduzindo Variância usando um *Baseline*

- Pode-se subtrair um *baseline* da função ação-valor em *Policy Gradient* (reduz variância):

$$E_{\pi_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s,a) q_{\pi_{\boldsymbol{\theta}}}(s,a)\right] = E_{\pi_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s,a)(q_{\pi_{\boldsymbol{\theta}}}(s,a) - B)\right]$$

- Isso é verdade pois pode-se mostrar que:

$$E_{\pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s,a)\, B] = 0$$

- Um bom baseline é a função valor: $B(s) = v_{\pi_{\boldsymbol{\theta}}}(s)$.
- Define-se então a função vantagem (*advantage function*):

$$a_{\pi_{\boldsymbol{\theta}}} = q_{\pi_{\boldsymbol{\theta}}}(s,a) - v_{\pi_{\boldsymbol{\theta}}}(s)$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto E_{\pi_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s,a) a_{\pi_{\boldsymbol{\theta}}}(s,a)\right]$$

Estimativa da Função Vantagem

- Uma forma de estimar a função vantagem é usar um estimador para $v_{\pi_{\boldsymbol{\theta}}}(s)$ e outro $q_{\pi_{\boldsymbol{\theta}}}(s,a)$:

$$\hat{v}_{\mathbf{v}}(s) \approx v_{\pi_{\boldsymbol{\theta}}}(s)$$
$$\hat{q}_{\mathbf{w}}(s,a) \approx q_{\pi_{\boldsymbol{\theta}}}(s,a)$$
$$\hat{a}(s,a) = \hat{q}_{\mathbf{w}}(s,a) - \hat{v}_{\mathbf{v}}(s)$$

- Então atualizar ambos os estimadores.
- Problema: duplica quantidade de parâmetros.

Estimativa da Função Vantagem

- Para a função valor verdade $v_{\pi_{\boldsymbol{\theta}}}(s)$, o erro TD $\delta_{\pi_{\boldsymbol{\theta}}}$:

$$\delta_{\pi_{\boldsymbol{\theta}}} = r + \gamma v_{\pi_{\boldsymbol{\theta}}}(s') - v_{\pi_{\boldsymbol{\theta}}}(s)$$

é um estimador não-enviesado da função vantagem. No caso, s' foi obtido a partir da ação a.

$$E_{\pi_\theta}\left[\delta_{\pi_{\boldsymbol{\theta}}}\middle|s,a\right] = E_{\pi_\theta}\left[r + \gamma v_{\pi_{\boldsymbol{\theta}}}(s')\middle|s,a\right] - v_{\pi_{\boldsymbol{\theta}}}(s) = q_{\pi_{\boldsymbol{\theta}}}(s,a) - v_{\pi_{\boldsymbol{\theta}}}(s)$$
$$= a_{\pi_{\boldsymbol{\theta}}}(s,a)$$

- Assim:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto E_{\pi_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s,a)\, \delta_{\pi_{\boldsymbol{\theta}}}\right]$$

- De modo, que basta ter $\hat{v}_{\mathbf{v}}(s)$: $\delta_{\mathbf{v}} = r + \gamma \hat{v}_{\mathbf{v}}(s') - \hat{v}_{\mathbf{v}}(s)$

Advantage TD Actor-Critic

Inicializar $\boldsymbol{\theta}$ e $\mathbf{v}$ arbitrariamente.

Loop (para cada episódio):

Inicializar S

Escolher ação $A \sim \pi_{\boldsymbol{\theta}}$

Loop (para cada passo do episódio $t = 0,1,\ldots,T-1$):

Tomar ação A, observar S', R

Escolher ação $A' \sim \pi_{\boldsymbol{\theta}}$

$\delta = R + \gamma\ \hat{v}_{\mathbf{v}}(S') - \hat{v}_{\mathbf{v}}(S)$

$\mathbf{v} = \mathbf{v} + \alpha_{\mathbf{v}} \delta \nabla \hat{v}_{\mathbf{v}}(S, A)$

$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \gamma^t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S_t, A_t)\ \delta$

$S = S'$; $A = A'$

Até fim do episódio

Actor-Critic Policy Gradient

- Ideias anteriores podem ser aproveitadas!
- GAE (*Generalized Advantage Estimation*): ideia do $TD(\lambda)$ para função vantagem.
- Também existe versão *backward view*.
- Pode-se usar *experience replay*.
- Pode-se melhorar o SGD usando Adam.

Para Saber Mais

- Curso do David Silver (aulas 6 e 7): http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.
- Capítulos 9, 10, 11, 12 e 13 do livro: SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction, second edition*. The MIT Press, 2017.
- OpenAI Spinning Up: https://spinningup.openai.com/en/latest/
- Nath Kohl and Peter Stone. Reinforcement Learning for Fast Quadrupedal Locomotion. ICRA 2004: http://www.cs.utexas.edu/users/pstone/Papers/bib2html-links/icra04.pdf

Laboratório 13

Laboratório 13

- Mountain car com Aproximação da Função Valor e/ou *Policy Gradient*.
- Uso do OpenAI Gym.