



APRENDIZADO POR REFORÇO

Aula 6: Policy-Based Methods

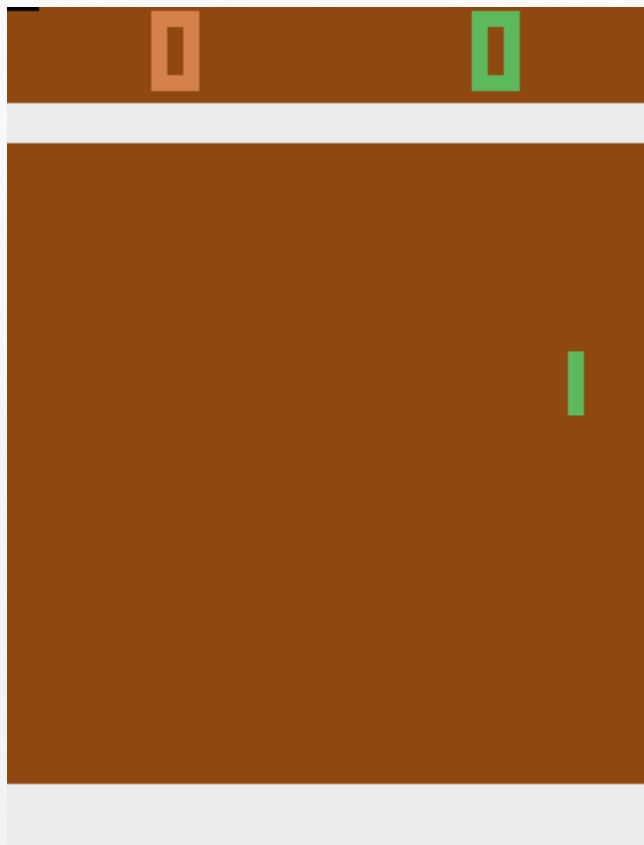
Lucas Pereira Cotrim

Marcos Menon José

lucas.cotrim@maua.br

marcos.jose@maua.br

DÚVIDAS TRABALHO T_1



- Tarefa A)
Determinar dimensões dos espaços de observações e ações.
- Tarefa B)
Implementar um agente aleatório e verificar sua performance
- Tarefa C)
Implementar o agente DQN para o PONG
- Tarefa D)
Implementar o Loop de Treino para o DQN



RELEMBRANDO ÚLTIMA AULA

Na última aula vimos:

- Como utilizar aproximadores de funções para parametrizar as Funções Valor $V_\pi(s)$ e $Q_\pi(s, a)$:

$$V_w(s) \approx V_\pi(s)$$

$$Q_w(s, a) \approx Q_\pi(s, a)$$

- A política era derivada a partir da Função Valor por comportamento $\epsilon - greedy$.

Na aule de hoje:

- Vamos parametrizar diretamente a função política $\pi(a|s)$:

$$\pi_\theta(s, a) = \mathbb{P}[a|s; \theta]$$

- O objetivo será aprender os parâmetros θ que levam à política π_θ com maior função Valor V_{π_θ} .

TÓPICOS DA AULA

- Introdução a Policy-Based Methods
- Policy Gradient
 - Finite Difference Methods
 - Monte-Carlo Policy Gradient (REINFORCE)
 - Actor-Critic Methods
 - One-Step Actor-Critic
 - Asynchronous Advantage Actor-Critic (A3C)
 - Deep Deterministic Policy Gradient (DDPG)
 - Trust-Region Policy Optimization (TRPO)
 - Proximal Policy Optimization (PPO)



Sutton&Barto, Cp.13

INTRODUÇÃO A POLICY-BASED METHODS

Introdução a Policy-Based Methods

INTRODUÇÃO A POLICY-BASED METHODS

Value-Based:

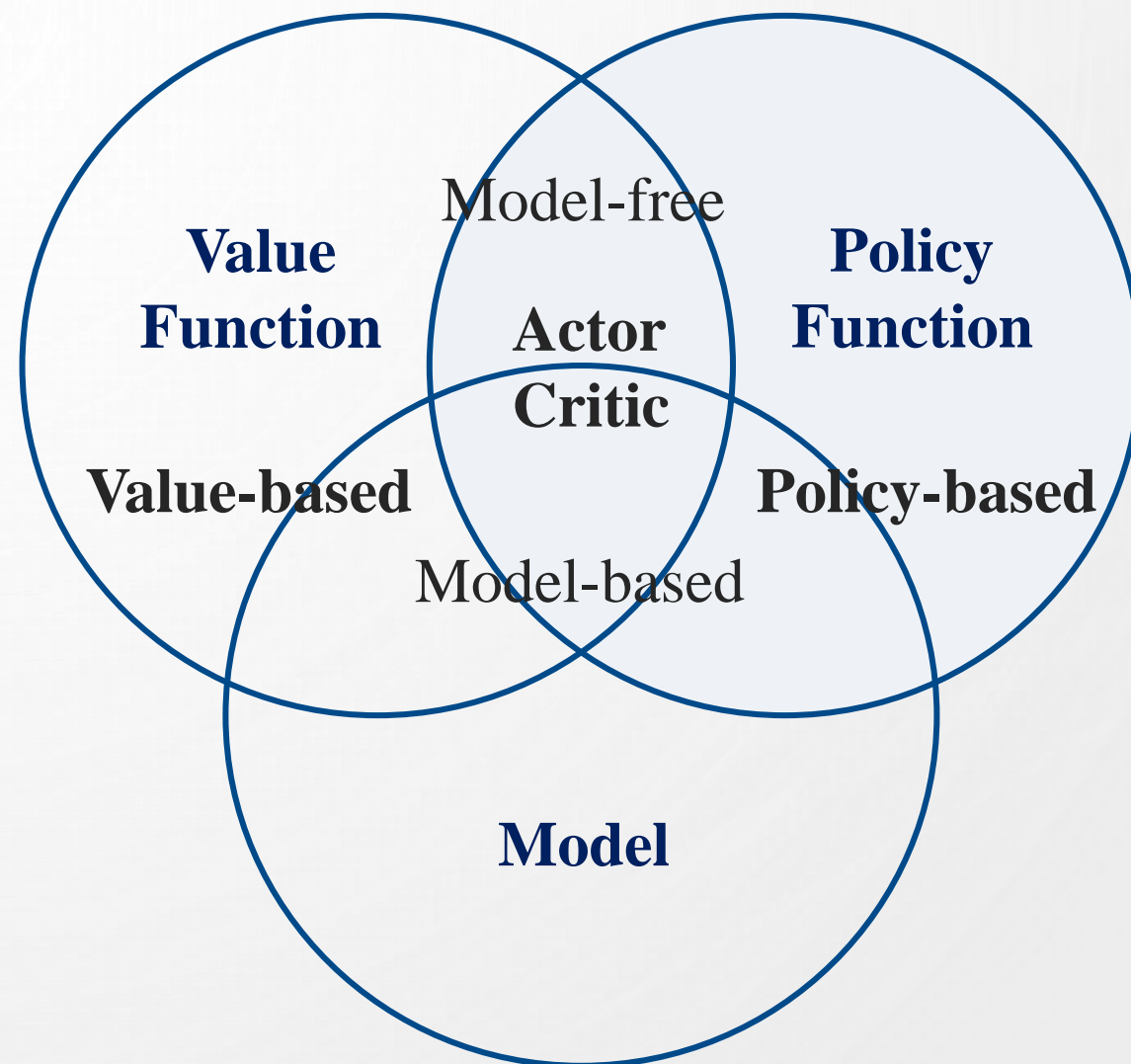
- Aprender Função Valor $V^*(s)$ ou $Q^*(s, a)$
- Política implícita (ϵ -greedy)

Policy-Based:

- Não há Função Valor
- Política π^* é aprendida

Actor-Critic:

- Tanto Função Valor quanto Função Política são parametrizadas e aprendidas



INTRODUÇÃO A POLICY-BASED METHODS

Vantagens:

- Melhores propriedades de convergência (convergência mais “suave”).
- Eficiente em espaços de ações \mathcal{A} de alta dimensão ou contínuos.
- Capacidade de aprender políticas estocásticas.
- Intuitivamente mais próximo ao aprendizado humano.

Desvantagens:

- Frequente convergência a ótimos locais.
- Avaliação de política é tipicamente ineficiente e apresenta alta variância.

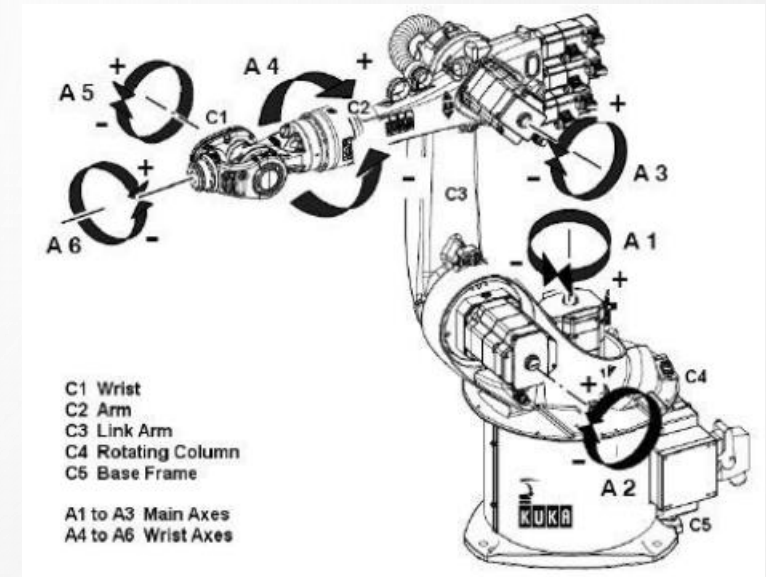
INTRODUÇÃO A POLICY-BASED METHODS

Espaços de Ações Contínuos

Em diversos problemas o agente pode interagir com o ambiente por meio de ações contínuas:

- Robótica (Controle de Atuadores)
- Carros Autônomos (Controle de aceleração, direção, freio)
- Business Management (Alocação de Recursos)
- NLP (Question Answering/Dialogue Generation)

Métodos Value-Based só conseguem lidar com esses problemas por meio da **discretização do espaço \mathcal{A}** .



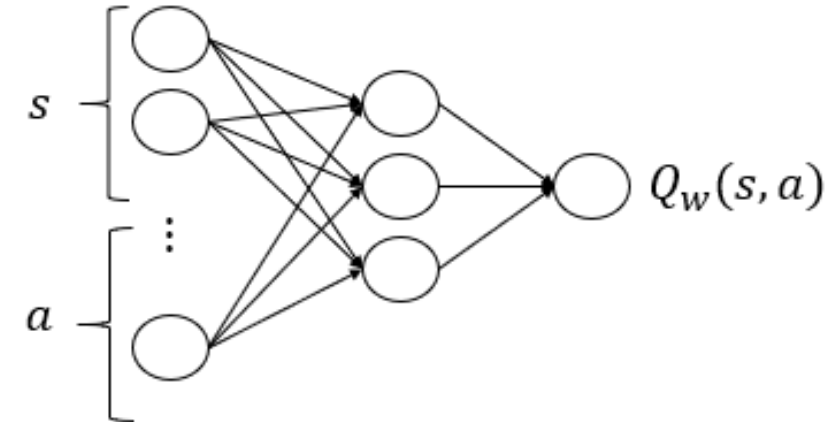
A: Where are you going? (1)
B: I'm going to the restroom. (2)

A: how old are you? (1)
B: I'm 16. (2)

INTRODUÇÃO A POLICY-BASED METHODS

Espaços de Ações Contínuos

Por que **métodos Value-Based** não podem ser aplicados em problemas de ações contínuas?



$$w \leftarrow w + \alpha \underbrace{\left[r_{t+1} + \gamma \max_{a'} Q_w(s_{t+1}, a') - Q_w(s_t, a_t) \right]}_{\text{target } y_i} \nabla_w Q(s_t, a_t)$$

Para encontrar $\max_{a \in \mathcal{A}} Q_w(s, a)$ seria preciso resolver um problema de **otimização não linear** a cada iteração!

Isso acontece pois não podemos simplesmente calcular $Q(s, a)$ para todas ações $a \in \mathcal{A}$ e tomar o máximo.

Métodos Policy-Based aproximam diretamente uma política estocástica $\pi_\theta(s, a)$ e simplesmente a **atualizam para aumentar as probabilidades de se tomar ações boas.**

INTRODUÇÃO A POLICY-BASED METHODS

Políticas Estocásticas

Por que aprender uma política não determinística?

- Em alguns problemas, a política ótima π^* é estocástica.
- Uma política determinística $\pi(s)$ é um caso particular de uma política estocástica $\pi(a|s)$.
- Ao treinar uma política estocástica, o comportamento do agente varia de forma mais suave.



Em um jogo de pedra-papel-tesoura:

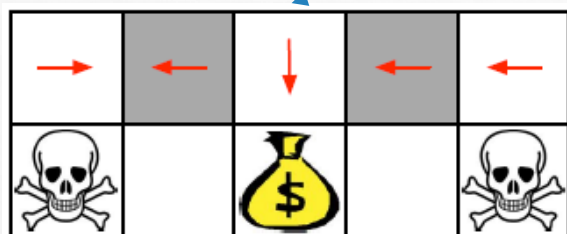
- Uma política determinística $a = \pi(s)$ é facilmente explorada pelo adversário.
- Uma política uniformemente aleatória $\pi(a|s)$ é ótima.

INTRODUÇÃO A POLICY-BASED METHODS

Políticas Estocásticas

Considere o seguinte ambiente, no qual o agente não é capaz de diferenciar os estados em cinza.

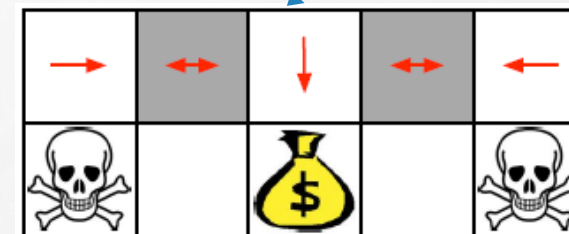
Value-Based Method



Política determinística ótima

- Mesma ação em ambos estados cinza
- Agente pode ficar preso e nunca chegar no destino.

Policy-Based Method



Política estocástica ótima

- 50% de probabilidade de \leftarrow , \rightarrow nos estados em cinza.
- Agente rapidamente chegará no objetivo

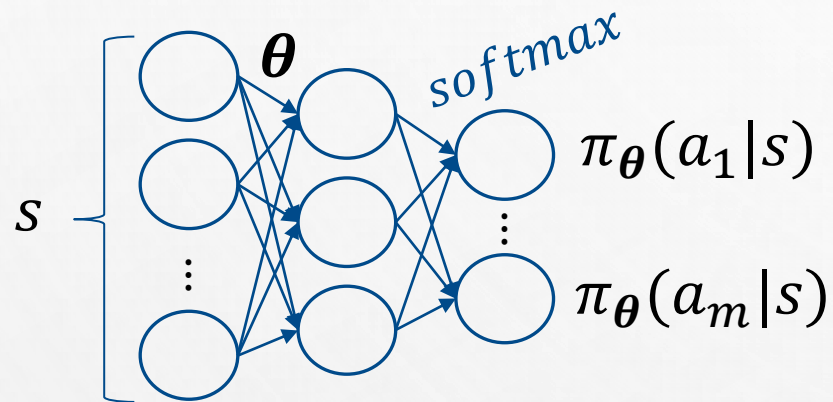
Em problemas parcialmente observáveis a política ótima pode ser não-determinística.

APROXIMADORES DE FUNÇÕES (AÇÕES DISCRETAS E CONTÍNUAS)

Em métodos Policy-Based parametrizamos a Função Política $\pi_{\theta}(s, a) = \mathbb{P}[a|s; \theta]$ diretamente.

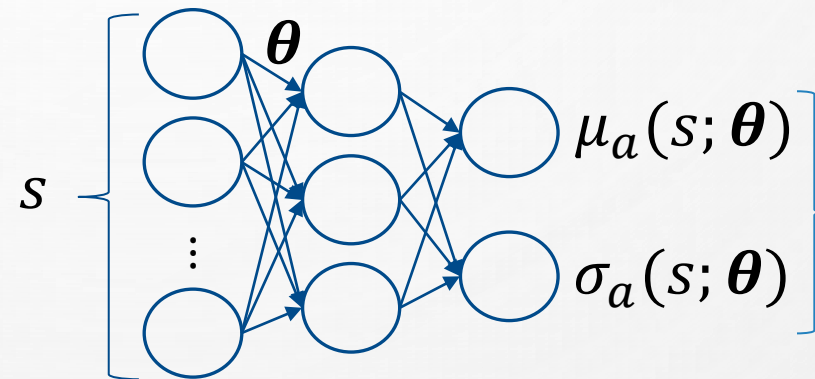
- Como representar ações discretas e contínuas?

Ações Discretas

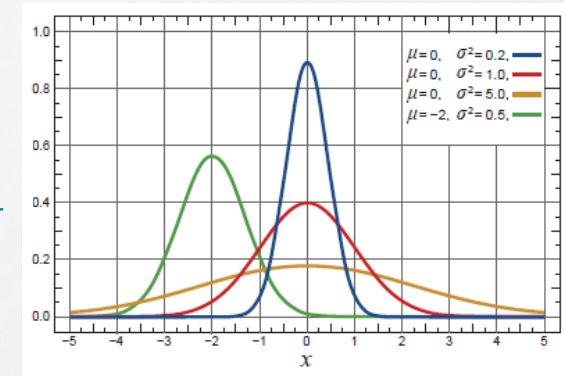


$$\sum_{a \in \mathcal{A}} \pi_{\theta}(a | s) = 1$$

Ações Contínuas



$$\pi_{\theta}(a | s) = \frac{1}{\sigma_a(s; \theta) \sqrt{2\pi}} \exp \left(-\frac{(a - \mu_a(s; \theta))^2}{2\sigma_a(s; \theta)^2} \right)$$



FUNDAMENTOS DE POLICY-BASED METHODS

Objetivo: Dada política parametrizada $\pi_{\theta}(a|s)$, encontrar os melhores parâmetros θ .

Problema: Como determinar a qualidade de uma política π_{θ} ?

Em problemas episódicos podemos definir uma função Performance $J(\theta)$ dada pelo valor do estado inicial:

$$J(\theta) = V_{\pi_{\theta}}(s_0) = \mathbb{E}_{\pi_{\theta}}[G_t | S_t = s_0] = \mathbb{E}_{\pi_{\theta}}[R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T | S_t = s_0]$$

Em problemas de horizonte infinito podemos avaliar o valor médio $J(\theta) = \sum_s d_{\pi_{\theta}}(s) V_{\pi_{\theta}}(s)$, mas na aula de hoje vamos focar em problemas episódicos.

FUNDAMENTOS DE POLICY-BASED METHODS

Métodos de Aprendizado por Reforço do tipo Policy-Based constituem um problema de **Otimização**:

$$\max_{\theta} J(\theta)$$

Encontrar parâmetros θ que maximizam $J(\theta) = V_{\pi_{\theta}}(s_0)$.

- Gradient-Free Methods:
 - Hill Climbing
 - Simplex, Amoeba, Nelder Mead
 - Algoritmos Genéticos
 - Cross-Entropy Method (CEM)
- Gradient-Based Methods (maior eficiência amostral)
 - **Gradient Descent**
 - Gradiente Conjugado

Policy Gradient

POLICY GRADIENT

Seja $J(\boldsymbol{\theta})$ uma Função Performance que mede a qualidade da política $\pi_{\boldsymbol{\theta}}(a|s)$.

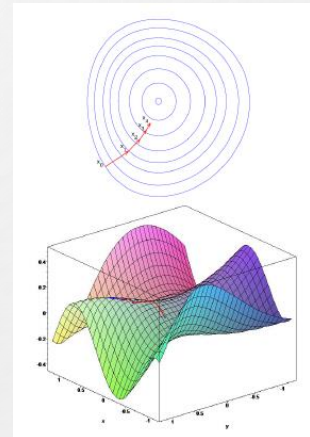
Algoritmos do tipo Policy Gradient procuram por um máximo local de $J(\boldsymbol{\theta})$ por meio do método do gradiente ascendente com relação aos parâmetros $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$$

$$\Delta\boldsymbol{\theta} = \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

onde $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ é o gradiente da Performance J com relação aos parâmetros $\boldsymbol{\theta}$ (Policy Gradient)

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{pmatrix} \partial J(\boldsymbol{\theta}) / \partial \theta_1 \\ \vdots \\ \partial J(\boldsymbol{\theta}) / \partial \theta_N \end{pmatrix}$$



FINITE DIFFERENCE METHODS

Métodos de Diferenças Finitas avaliam o Policy Gradient $\nabla J(\boldsymbol{\theta})$ da política $\pi_{\boldsymbol{\theta}}(a|s)$ por meio de aproximações de primeira ordem das derivadas parciais $\partial J(\boldsymbol{\theta}) / \partial \theta_i$:

Para cada dimensão $k = \{1, \dots, N\}$:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_k} \approx \frac{J(\boldsymbol{\theta} + \delta \mathbf{u}_k) - J(\boldsymbol{\theta})}{\delta}$$

- Vantagem: Funciona para políticas arbitrárias (não diferenciáveis)
- Desvantagem: Extremamente ineficiente, estimativas com elevado ruído.

Monte-Carlo Policy Gradient (algoritmo REINFORCE)

NOTAÇÃO: SCORE FUNCTION

Vamos calcular o gradiente $\nabla_{\theta} J(\theta)$ da Performance J de forma **analítica**. Para isso, precisamos ter uma política π_{θ} **diferenciável** onde π_{θ} é não nula.

Se π_{θ} é dado por um aproximador de funções diferenciável (Softmax, Gaussian, Redes Neurais) com pesos θ :

- Podemos calcular o gradiente $\nabla_{\theta} \pi_{\theta}(a|s)$ de forma analítica.
- Likelihood Ratio:

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

- **Score Function:** $\nabla_{\theta} \log \pi_{\theta}(a|s)$ é o gradiente do logaritmo natural da política $\pi_{\theta}(a|s)$

COMO CALCULAR O GRADIENTE $\nabla_{\theta} J(\theta)$

- Vamos utilizar a seguinte função Performance: $J(\theta) = V_{\pi_{\theta}}(s_0)$

- A partir da definição da Função Valor V , temos:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_{\theta}}[G_t | S_t = s_0] \\ &= \mathbb{E}_{\pi_{\theta}}[R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T | S_t = s_0] \end{aligned}$$

- Podemos calcular o valor esperado por meio da política $\pi_{\theta}(a|s)$:

$$= \sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) Q_{\pi_{\theta}}(s_0, a)$$

- Denotando uma trajetória qualquer como $\tau = \{s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T, s_T\}$:

$$J(\theta) = \sum_{\tau} \mathbb{P}_{\pi_{\theta}}(\tau) G_0(\tau)$$

onde $\mathbb{P}_{\pi_{\theta}}(\tau)$ é a probabilidade da trajetória τ ocorrer sob a política π_{θ} e $G_0(\tau)$ é o retorno obtido na trajetória τ a partir do estado inicial s_0 , $G_0(\tau) = R_1 + \gamma R_2 + \dots + \gamma^{T-1} R_T$

COMO CALCULAR O GRADIENTE $\nabla_{\theta} J(\theta)$

- A partir da formulação da Performance $J(\theta) = \sum_{\tau} \mathbb{P}_{\pi_{\theta}}(\tau) G_0(\tau)$ vamos calcular o gradiente $\nabla_{\theta} J(\theta)$:

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \sum_{\tau} \mathbb{P}_{\pi_{\theta}}(\tau) G_0(\tau) \\ &= \sum_{\tau} \nabla_{\theta} \mathbb{P}_{\pi_{\theta}}(\tau) G_0(\tau) \\ &= \sum_{\tau} \mathbb{P}_{\pi_{\theta}}(\tau) G_0(\tau) \frac{\nabla_{\theta} \mathbb{P}_{\pi_{\theta}}(\tau)}{\mathbb{P}_{\pi_{\theta}}(\tau)} \\ &= \sum_{\tau} \mathbb{P}_{\pi_{\theta}}(\tau) G_0(\tau) \nabla_{\theta} \log \mathbb{P}_{\pi_{\theta}}(\tau)\end{aligned}\tag{1}$$

The diagram includes two dashed blue ellipses. The first ellipse encloses the fraction $\frac{\nabla_{\theta} \mathbb{P}_{\pi_{\theta}}(\tau)}{\mathbb{P}_{\pi_{\theta}}(\tau)}$ in the third line, with a blue arrow pointing to the text "Score Function". The second ellipse encloses the term $\nabla_{\theta} \log \mathbb{P}_{\pi_{\theta}}(\tau)$ in the fourth line.

- Note que o retorno $G_0(\tau)$ depende apenas da trajetória, não dos parâmetros θ da política.
- A Função Score $\nabla_{\theta} \log \mathbb{P}_{\pi_{\theta}}(\tau)$ aparece após manipulação algébrica.

COMO CALCULAR O GRADIENTE $\nabla_{\theta} J(\theta)$

- Vamos agora calcular o gradiente da Score Function da probabilidade de uma trajetória τ com relação aos parâmetros θ da política de ações π_{θ} que gerou esta trajetória:

$$\begin{aligned}
 \nabla_{\theta} \log \mathbb{P}_{\pi_{\theta}}(\tau) &= \nabla_{\theta} \log \left[\overset{\text{Initial State Distribution}}{\mu(s_0)} \prod_{t=0}^{T-1} \overset{\text{Policy}}{\pi_{\theta}(a_t | s_t)} \overset{\text{Environment Dynamics}}{\mathbb{P}(s_{t+1} | s_t, a_t)} \right] \\
 &= \nabla_{\theta} \left[\log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t | s_t) + \log \mathbb{P}(s_{t+1} | s_t, a_t) \right] \\
 &= \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \quad \text{Model-Free!} \quad (2)
 \end{aligned}$$

- Ao tomar o gradiente ∇_{θ} eliminamos a necessidade de conhecer a dinâmica do ambiente!

COMO CALCULAR O GRADIENTE $\nabla_{\theta} J(\theta)$

- Substituindo (2) em (1) temos:

$$\nabla_{\theta} J(\boldsymbol{\theta}) = \sum_{\tau} \left[\mathbb{P}_{\pi_{\theta}}(\tau) G_0(\tau) \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta} \left(a_t^{(\tau)} | s_t^{(\tau)} \right) \right]$$

- Mas como lidar com a soma sobre todas **trajetórias τ** possíveis e suas **probabilidades $\mathbb{P}_{\pi_{\theta}}(\tau)$** ?
- **Solução:** Amostrar grande número de trajetórias (Monte-Carlo Policy Gradient)

$$\nabla_{\theta} J(\boldsymbol{\theta}) \approx \frac{1}{N_{\tau}} \sum_{j=1}^{N_{\tau}} \left[G_0(\tau_j) \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta} \left(a_t^{(\tau_j)} | s_t^{(\tau_j)} \right) \right]$$

POLICY GRADIENT THEOREM

- O Policy Gradient Theorem generaliza o resultado obtido nos últimos slides para qualquer Função Performance $J(\theta)$:

Policy Gradient Theorem:

$$\nabla_{\theta} J(\theta) \propto \sum_{s \in \mathcal{S}} \mu(s) \sum_{a \in \mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q_{\pi_{\theta}}(s, a)$$

- Vamos determinar a Lei de Atualização dos parâmetros θ a partir do Policy Gradient Theorem.

Sutton&Barto Cp. 13, pp. 325

Proof of the Policy Gradient Theorem (episodic case)

With just elementary calculus and re-arranging of terms, we can prove the policy gradient theorem from first principles. To keep the notation simple, we leave it implicit in all cases that π is a function of θ , and all gradients are also implicitly with respect to θ . First note that the gradient of the state-value function can be written in terms of the action-value function as

$$\begin{aligned} \nabla v_{\pi}(s) &= \nabla \left[\sum_a \pi(a|s) q_{\pi}(s, a) \right], \quad \text{for all } s \in \mathcal{S} && \text{(Exercise 3.18)} \\ &= \sum_a \left[\nabla \pi(a|s) q_{\pi}(s, a) + \pi(a|s) \nabla q_{\pi}(s, a) \right] && \text{(product rule of calculus)} \\ &= \sum_a \left[\nabla \pi(a|s) q_{\pi}(s, a) + \pi(a|s) \nabla \sum_{s', r} p(s', r|s, a) (r + v_{\pi}(s')) \right] \\ &&& \text{(Exercise 3.19 and Equation 3.2)} \\ &= \sum_a \left[\nabla \pi(a|s) q_{\pi}(s, a) + \pi(a|s) \sum_{s'} p(s'|s, a) \nabla v_{\pi}(s') \right] && \text{(Eq. 3.4)} \\ &= \sum_a \left[\nabla \pi(a|s) q_{\pi}(s, a) + \pi(a|s) \sum_{s'} p(s'|s, a) \right. \\ &\quad \left. \sum_{a'} [\nabla \pi(a'|s') q_{\pi}(s', a') + \pi(a'|s') \sum_{s''} p(s''|s', a') \nabla v_{\pi}(s'')] \right] \\ &= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \Pr(s \rightarrow x, k, \pi) \sum_a \nabla \pi(a|x) q_{\pi}(x, a), \end{aligned}$$

after repeated unrolling, where $\Pr(s \rightarrow x, k, \pi)$ is the probability of transitioning from state s to state x in k steps under policy π . It is then immediate that

$$\begin{aligned} \nabla J(\theta) &= \nabla v_{\pi}(s_0) \\ &= \sum_s \left(\sum_{k=0}^{\infty} \Pr(s_0 \rightarrow s, k, \pi) \right) \sum_a \nabla \pi(a|s) q_{\pi}(s, a) \\ &= \sum_s \eta(s) \sum_a \nabla \pi(a|s) q_{\pi}(s, a) && \text{(box page 199)} \\ &= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|s) q_{\pi}(s, a) \\ &= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_{\pi}(s, a) && \text{(Eq. 9.3)} \\ &\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_{\pi}(s, a) && \text{(Q.E.D.)} \end{aligned}$$

LEI DE ATUALIZAÇÃO DE PARÂMETROS θ

- A partir do Policy Gradient Theorem temos:

$$\nabla_{\theta} J(\theta) \propto \sum_{s \in \mathcal{S}} \mu(s) \sum_{a \in \mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q_{\pi_{\theta}}(s, a)$$

- A distribuição $\mu(s)$ representa a frequência com que o estado s é visitado com a política π , assim, podemos escrever:

$$\begin{aligned} &= \mathbb{E}_{\pi_{\theta}} \left[\sum_{a \in \mathcal{A}} Q_{\pi_{\theta}}(S_t, a) \nabla_{\theta} \pi_{\theta}(a|S_t) \right] \\ &= \mathbb{E}_{\pi_{\theta}} \left[\sum_{a \in \mathcal{A}} \pi_{\theta}(a|S_t) Q_{\pi_{\theta}}(S_t, a) \frac{\nabla_{\theta} \pi_{\theta}(a|S_t)}{\pi_{\theta}(a|S_t)} \right] \\ &= \mathbb{E}_{\pi_{\theta}} \left[G_t \frac{\nabla_{\theta} \pi_{\theta}(A_t|S_t)}{\pi_{\theta}(A_t|S_t)} \right] \end{aligned}$$

Lei de Atualização
de Parâmetros θ

$$\theta \leftarrow \theta + \alpha G_t \frac{\nabla_{\theta} \pi_{\theta}(A_t|S_t)}{\pi_{\theta}(A_t|S_t)}$$

Score Function:
 $\nabla_{\theta} \log \pi_{\theta}(A_t|S_t)$

LEI DE ATUALIZAÇÃO DE PARÂMETROS θ

- A Lei de Atualização dos parâmetros θ é baseada no gradiente da função Performance $J(\theta)$:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$$

$$\theta \leftarrow \theta + \alpha \underbrace{G_t}_{\text{retorno}} \underbrace{\frac{\nabla_{\theta} \pi_{\theta}(A_t | S_t)}{\pi_{\theta}(A_t | S_t)}}_{\text{probabilidade da ação ter sido tomada}}$$

- A Atualização dos Parâmetros θ ocorre na direção do gradiente da política (direção que aumenta a probabilidade da mesma ação ser tomada neste estado) e é proporcional ao retorno, assim quanto maior o retorno obtido maior a chance da ação ser tomada no futuro.
- A divisão pela probabilidade da ação ter sido tomada é feita para normalizar as atualizações e não priorizar ações mais frequentes.

REINFORCE: MONTE-CARLO POLICY GRADIENT

Algoritmo: REINFORCE (Monte-Carlo Policy Gradient)

Input: Política parametrizada e diferenciável $\pi_{\theta}(a|s)$

Parâmetro do Algoritmo: Taxa de aprendizado $\alpha > 0$

Inicializar Parâmetros θ da política parametrizada $\pi_{\theta}(a|s)$ aleatoriamente

Repetir para cada episódio:

Gerar episódio $\tau = \{S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T\} \sim \pi_{\theta}(a|s)$

Repetir para cada timestep $t = 0, 1, \dots, T - 1$ do episódio τ :

Calcular Retorno: $G_t \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$

Atualizar Função Política: $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_{\theta} \log \pi_{\theta}(A_t|S_t)$

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

REINFORCE: MONTE-CARLO POLICY GRADIENT

- Lei de Atualização de parâmetros utilizando *Cross-Entropy Loss Function*:

$$Loss = - \sum_{i=1}^m y_i \log \hat{y}_i$$

- Para cada ação transição $(S_t, A_t, R_{t+1}, S_{t+1}, done)$ do episódio

$$d = [OneHotEncoding(A_t) - \pi_{\theta}(\cdot | S_t)]$$

$$target = \pi_{\theta}(\cdot | S_t) + \alpha G_t d$$

- Ao minimizar a *Cross-Entropy Loss Function* para os alvos definidas acima, o Keras executa a Atualização do REINFORCE $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_{\theta} \log \pi_{\theta}(A_t | S_t)$

$$\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Probability distribution
over actions

REINFORCE: EXEMPLO

REINFORCE_cartpole.ipynb

CO

REINFORCE_cartpole.ipynb

☆

File Edit View Insert Runtime Tools Help Last saved at 11:06 AM

Table of contents

🔍

📄

📁

📋

🖨

Cartpole Environment

Instalação de bibliotecas e Imports

Global Variables

Criação de Ambiente

REINFORCE

Monte-Carlo Policy Gradient (REINFORCE) agent class

Create REINFORCE agent and train

Plot REINFORCE results

Resultado Esperado

DQN

Value Network keras model

Criação e compilação de agente keras-rl

Teste do agente treinado

Plot DQN Results

Resultado Esperado

+ Code + Text

Connect Editing

↑ ↓ 🔗 💬 ✎ 📄 🗑 ⋮

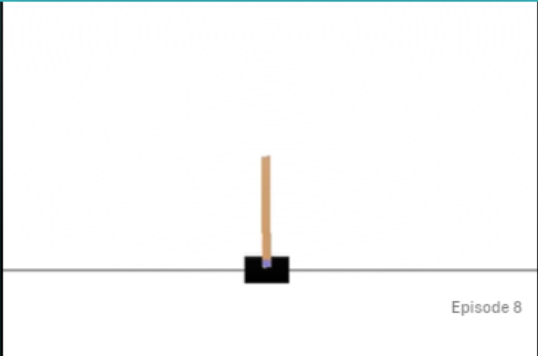
▼ Cartpole Environment

Vamos implementar o algoritmo REINFORCE no ambiente Cartpole-v0, no qual o objetivo é controlar a aceleração de um carro para equilibrar um pendulo invertido na posição vertical pelo maior tempo possível.

<https://gym.openai.com/envs/CartPole-v1/>

CartPole-v1

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.



RandomAgent on CartPole-v1

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson [Barto83].

LIMITAÇÕES DO ALGORITMO REINFORCE

- As atualizações dos Parâmetros θ são *unbiased*, mas apresentam **alta variância**:

$$\theta \leftarrow \theta + \alpha \gamma^t G_t \frac{\nabla_{\theta} \pi_{\theta}(A_t | S_t)}{\pi_{\theta}(A_t | S_t)}$$

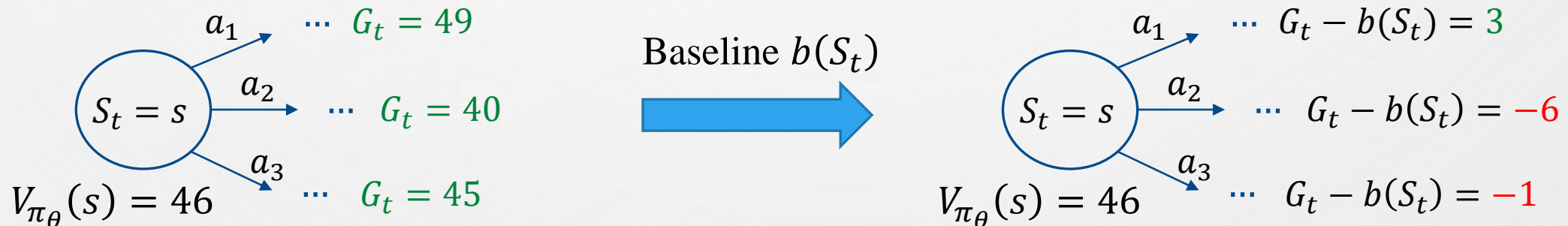
- Isso acontece porque os retornos G_t dependem de todas as recompensas obtidas até o final do episódio $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T$
- A alta variância do estimador de $\nabla_{\theta} J(\theta)$ leva a baixa eficiência amostral e longo tempo de convergência.
- Soluções:
 - Utilizar uma *Baseline* para reduzir variância \rightarrow REINFORCE with Baseline
 - Substituir Retorno G_t por estimativa treinada do retorno \rightarrow Actor-Critic

REINFORCE WITH BASELINE

- A partir do Policy Gradient Theorem, o gradiente da Performance é dado por:

$$\nabla_{\theta} J(\theta) \propto \mathbb{E}_{\pi_{\theta}} \left[G_t \frac{\nabla_{\theta} \pi_{\theta}(A_t | S_t)}{\pi_{\theta}(A_t | S_t)} \right]$$

- Se substituirmos G_t por $G_t - b(S_t)$, o valor esperado não é alterado (porque $b(S_t)$ independe de a).
- Uma escolha natural para $b(S_t)$ é uma aproximação da função valor $V_w(S_t)$
- Intuição: Facilita a identificação de ações ótimas em meio a diversas ações boas.



REINFORCE WITH BASELINE: MONTE-CARLO POLICY GRADIENT

Algoritmo: REINFORCE with Baseline (Monte-Carlo Policy Gradient)

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 θ da política parametrizada $\pi_{\theta}(a|s)$ e parâmetros w da Função Valor $V_w(s)$ aleatoriamente

Repetir para cada episódio:

Gerar episódio $\tau = \{S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T\} \sim \pi_{\theta}(a|s)$

Repetir para cada timestep $t = 0, 1, \dots, T - 1$ do episódio τ :

Calcular Retorno $G_t \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$

Calcular Erro: $\delta \leftarrow G_t - V_w(S_t)$

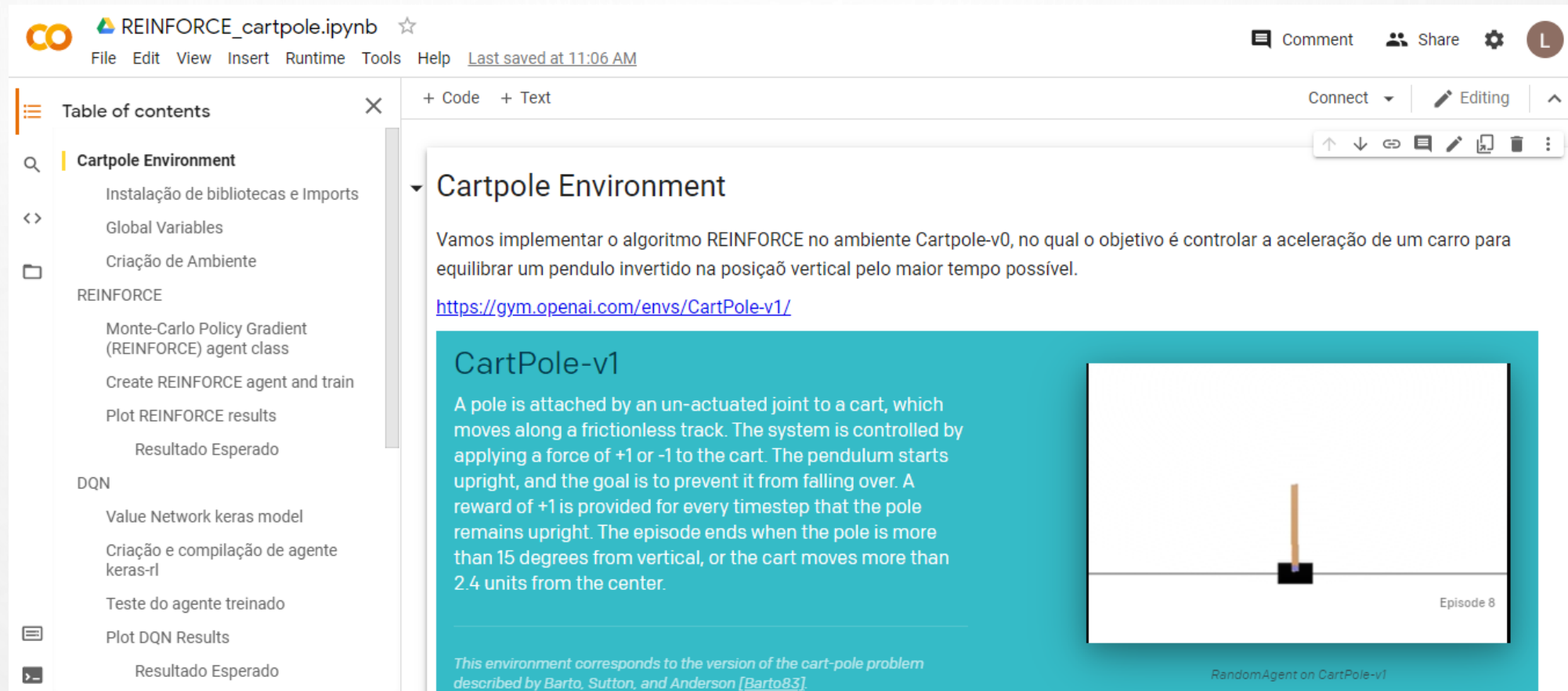
Atualizar Função Valor (Baseline): $w \leftarrow w + \alpha_w \delta \nabla_w V_w(S_t)$

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

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

REINFORCE WITH BASELINE: EXEMPLO

REINFORCE_cartpole.ipynb



REINFORCE_cartpole.ipynb

File Edit View Insert Runtime Tools Help Last saved at 11:06 AM

Comment Share

Table of contents

- Cartpole Environment
 - Instalação de bibliotecas e Imports
 - Global Variables
 - Criação de Ambiente
- REINFORCE
 - Monte-Carlo Policy Gradient (REINFORCE) agent class
 - Create REINFORCE agent and train
 - Plot REINFORCE results
 - Resultado Esperado
- DQN
 - Value Network keras model
 - Criação e compilação de agente keras-rl
 - Teste do agente treinado
 - Plot DQN Results
 - Resultado Esperado

+ Code + Text

Connect Editing

Cartpole Environment

Vamos implementar o algoritmo REINFORCE no ambiente Cartpole-v0, no qual o objetivo é controlar a aceleração de um carro para equilibrar um pendulo invertido na posição vertical pelo maior tempo possível.

<https://gym.openai.com/envs/CartPole-v1/>

CartPole-v1

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson [Barto83].

Episode 8

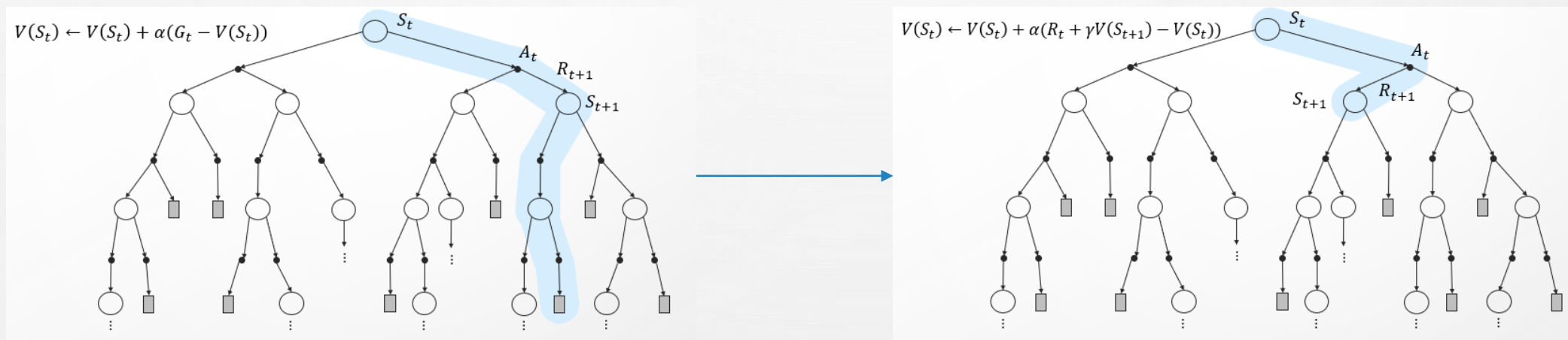
RandomAgent on CartPole-v1

ACTOR-CRITIC POLICY GRADIENT

Actor-Critic Policy Gradient

LIMITAÇÕES DO ALGORITMO REINFORCE COM BASELINE

- O Algoritmo REINFORCE com Baseline é um algoritmo de Monte-Carlo, ou seja, precisamos **esperar o término do episódio para atualizar $\pi_{\theta}(a|s)$ e $V_w(s)$** .
- Isso impede a implementação em problemas de **horizonte infinito** e torna o **treino menos frequente**.
- Na aula 3 vimos como o TD(0) corrige este problema estimando o retorno G_t a partir da Função Valor calculada no estado seguinte. A mesma estimativa pode ser implementada aqui:



ACTOR-CRITIC POLICY GRADIENT

- Seja a lei de atualização do algoritmo REINFORCE com Baseline:

$$\theta \leftarrow \theta + \alpha \gamma^t [\textcolor{red}{G}_t - V_w(S_t)] \frac{\nabla_{\theta} \pi_{\theta}(A_t | S_t)}{\pi_{\theta}(A_t | S_t)}$$

- Vamos utilizar a Função Valor parametrizada para aproximar o Retorno G_t **a cada timestep**:

$$\theta \leftarrow \theta + \alpha \gamma^t [\textcolor{red}{R}_t + \gamma V_w(S_{t+1}) - V_w(S_t)] \frac{\nabla_{\theta} \pi_{\theta}(A_t | S_t)}{\pi_{\theta}(A_t | S_t)}$$

- Essa lei de atualização constitui o algoritmo Actor-Critic:
 - Actor $\pi_{\theta}(a|s)$ executa ações de acordo com sua política e treina com base no retorno estimado pelo Critic.
 - Critic $V_w(s)$ avalia ações executadas pelo Actor e treina sua Função Valor.
- Permite o treino a cada timestep (sem precisar observar retorno real G_t)

ACTOR-CRITIC POLICY GRADIENT

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 θ, 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}

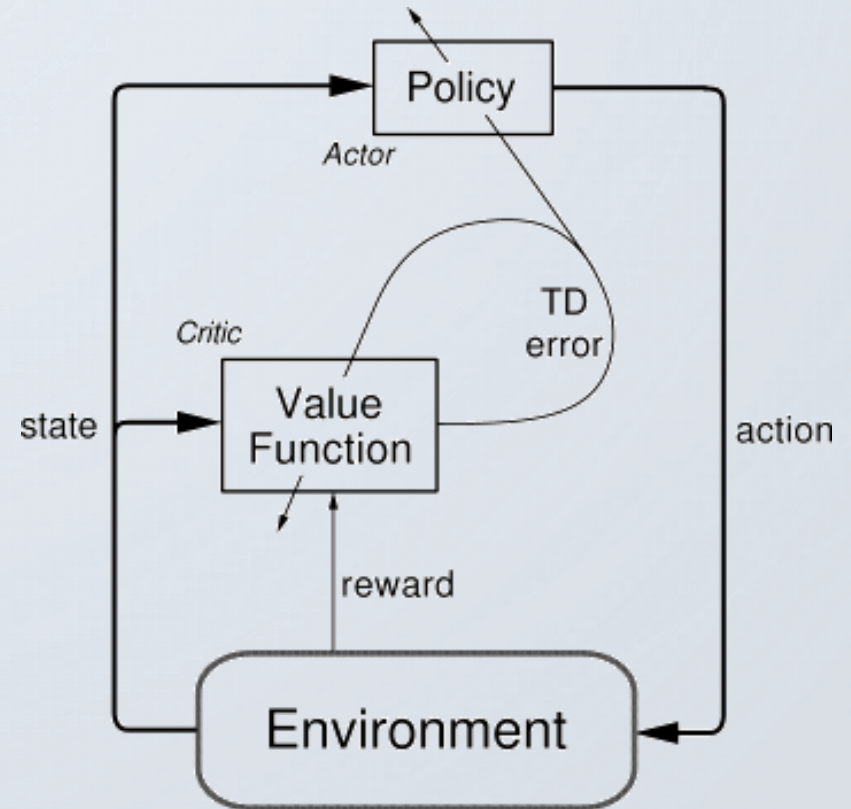
Calcular Erro: $\delta \leftarrow R_{t+1} + \gamma V_w(S_{t+1}) - V_w(S_t)$

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



ACTOR-CRITIC POLICY GRADIENT

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

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

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

Inicializar Parâmetros θ, \mathbf{w} aleatoriamente

Repetir (para cada episódio):

 Inicializar estado S_0

$t = 0$

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

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

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

 Amostrar ação $A_{t+1} \sim \pi_{\theta}(a'|S_{t+1})$

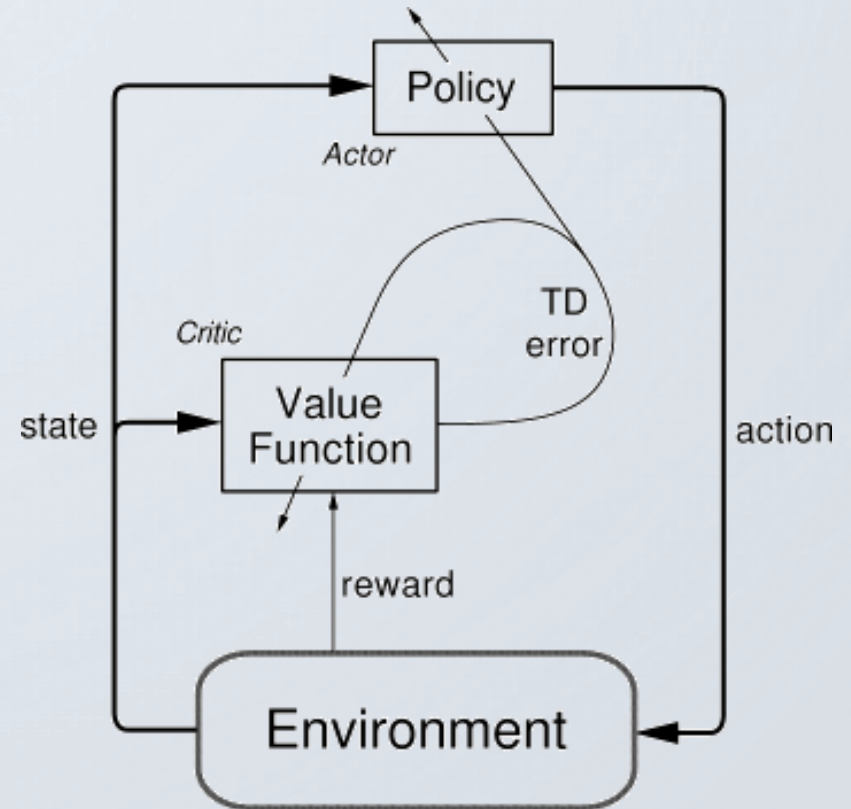
 Atualizar Actor: $\theta \leftarrow \theta + \alpha_{\theta} Q_{\mathbf{w}}(S_t, A_t) \nabla_{\theta} \log \pi_{\theta}(A_t|S_t)$

 Calcular Erro: $\delta \leftarrow R_{t+1} + \gamma Q_{\mathbf{w}}(S_{t+1}, A_{t+1}) - Q_{\mathbf{w}}(S_t, A_t)$

 Atualizar Critic: $\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}} \delta \nabla_{\mathbf{w}} Q_{\mathbf{w}}(S_t, A_t)$

$t \leftarrow t + 1$

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



ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC (A3C)

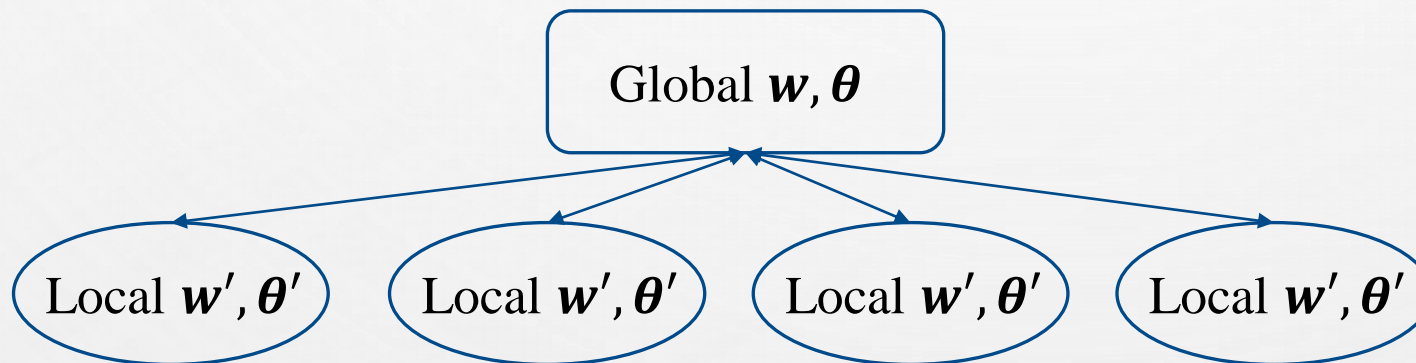
Asynchronous Advantage Actor-Critic
(A3C)

ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC (A3C)

O algoritmo Actor-Critic tradicional pode ser reformulado para computação paralela em contexto multi-agente. O Asynchronous Advantage Actor-Critic, ou A3C (Mnih, 2016) é uma extensão do Actor-Critic com foco em treinamento paralelo.

- Cada *thread* sincroniza parâmetros locais $\mathbf{w}', \boldsymbol{\theta}'$ com parâmetros globais $\mathbf{w}, \boldsymbol{\theta}$ de tempo em tempo.
- Em cada *thread* um crítico e um agente acumulam atualizações $d\mathbf{w}$ e $d\boldsymbol{\theta}$ a cada timestep local.
- No fim do episódio parâmetros globais são atualizados de forma assíncrona: $\mathbf{w} \leftarrow \mathbf{w} + d\mathbf{w}$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + d\boldsymbol{\theta}$$



ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC A3C

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$

// Assume thread-specific parameter vectors θ' and θ'_v

Initialize thread step counter $t \leftarrow 1$

repeat

Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.

Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$

$t_{start} = t$

Get state s_t

repeat

Perform a_t according to policy $\pi(a_t|s_t; \theta')$

Receive reward r_t and new state s_{t+1}

$t \leftarrow t + 1$

$T \leftarrow T + 1$

until terminal s_t **or** $t - t_{start} == t_{max}$

$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t // \text{ Bootstrap from last state} \end{cases}$

for $i \in \{t - 1, \dots, t_{start}\}$ **do**

$R \leftarrow r_i + \gamma R$

Accumulate gradients wrt θ' : $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$

Accumulate gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$

end for

Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$

until $T > T_{max}$

Simulação de episódio

Acúmulo de atualizações de gradientes

Atualização assíncrona de parâmetros globais

Mnih, V.; et. al. Asynchronous Methods for Deep Reinforcement Learning, 2016

ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC A3C



<https://www.youtube.com/watch?v=Ajic08-iPx8>

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Table 1. Mean and median human-normalized scores on 57 Atari games using the human starts evaluation metric. Supplementary Table SS3 shows the raw scores for all games.

Fonte: Mnih, V.; et. al. Asynchronous Methods for Deep Reinforcement Learning, 2016

DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

Deep Deterministic Policy Gradient
(DDPG)

DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

Nos algoritmos anteriores a política parametrizada $\pi_{\theta}(a|s)$ era uma **política estocástica**, ou seja, para cada estado a política retornava uma distribuição de probabilidades sobre o espaço de ações.

- O cálculo do gradiente $\nabla_{\theta}J(\theta)$ é mais complexo para políticas estocásticas pois é necessário amostrar sobre diferentes ações.
- Ao utilizar uma **política determinística** $\pi_{\theta}(s)$ podemos calcular $\nabla_{\theta}J(\theta)$ de forma fechada:

$$\mathbb{E}_{\pi_{\theta}} \left[\sum_{a \in \mathcal{A}} Q_{\pi_{\theta}}(S_t, a) \nabla_{\theta} \pi_{\theta}(a|S_t) \right] \Rightarrow \nabla_{\theta}J(\theta) = \mathbb{E}_{\pi_{\theta}} \left[\nabla_{\theta} \pi_{\theta}(S_t) \left(\nabla_a Q_w(S_t, a) \Big|_{a=A_t} \right) \right]$$

Política Estocástica
($\sum_a Q_{\pi_{\theta}}(S_t, a)$)
Política Determinística ($A_t = \pi_{\theta}(S_t)$)

DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

- Ao reformular o Policy Gradient para políticas determinísticas temos um **problema**:
Uma política estocástica naturalmente realiza exploração durante o treino.
No entanto, uma política determinística irá realizar sempre a mesma ação em cada estado.
- **Solução**: Introduzir um ruído aleatório \mathcal{N}_t sobre a ação amostrada antes de executá-la.
 - Este ruído desempenha um papel análogo ao comportamento ϵ -greedy no DQN
- O resultado é o DDPG, um algoritmo Actor-Critic análogo ao DQN
 - Model-Free
 - Off-Policy
 - Diferente do DQN, o DDPG funciona com ações contínuas.

DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .

Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer R

for episode = 1, M **do**

 Initialize a random process \mathcal{N} for action exploration

 Receive initial observation state s_1

for t = 1, T **do**

 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise

 Execute action a_t and observe reward r_t and observe new state s_{t+1}

 Store transition (s_t, a_t, r_t, s_{t+1}) in R

 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

 Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

end for

end for

Armazenar Transição em Buffer R

Amostrar minibatch de N Transições de Buffer R

Atualizar parâmetros de Actor e Critic

Atualizar target networks

David Silver, et al.. *Deterministic Policy Gradient Algorithms*. ICML, Jun 2014, Beijing, China.

DDPG EXEMPLO: PENDULUM-V0 EM KERAS-RL 2

DDPG_pendulum-v0.ipynb

The screenshot displays a Jupyter Notebook titled "DDPG_Pendulum-v0.ipynb" in a web-based environment. The interface includes a top menu bar with options like File, Edit, View, Insert, Runtime, Tools, and Help. A sidebar on the left shows a "Table of contents" with sections such as "Imports", "Create Environment", "Create Actor model", "Create Critic model", "Create and Compile DDPG Agent", "Train Agent", "Save and Test", and "Section". The main content area shows the notebook's title "DDPG: Pendulum-v0" and a description in Portuguese: "Vamos treinar um agente DDPG para resolver o problema de equilibrar um pêndulo invertido no ambiente 'Pendulum-v0' do gym. Neste problema tanto o espaço de estados quanto o espaço de ações são contínuos. A ação representa o torque aplicado na articulação do pêndulo e o estado é dado por sua posição e velocidade angular." It also provides links for the "gym Environment" and the "Código em:" (Code in:). Below the text, there is a preview of the "Pendulum-v0" environment documentation, which states: "The inverted pendulum swingup problem is a classic problem in the control literature. In this version of the problem, the pendulum starts in a random position, and the goal is to swing it up so it stays upright."

DDPG_Pendulum-v0.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

Comment Share Settings L

Table of contents X

- DDPG: Pendulum-v0
 - Imports
 - Create Environment
 - Create Actor model
 - Create Critic model
 - Create and Compile DDPG Agent
 - Train Agent
 - Save and Test
 - Section

+ Code + Text

Connect Editing ^

DDPG: Pendulum-v0

Vamos treinar um agente DDPG para resolver o problema de equilibrar um pêndulo invertido no ambiente 'Pendulum-v0' do gym.

Neste problema tanto o espaço de estados quanto o espaço de ações são contínuos. A ação representa o torque aplicado na articulação do pêndulo e o estado é dado por sua posição e velocidade angular.

gym Environment: <https://gym.openai.com/envs/Pendulum-v0/>

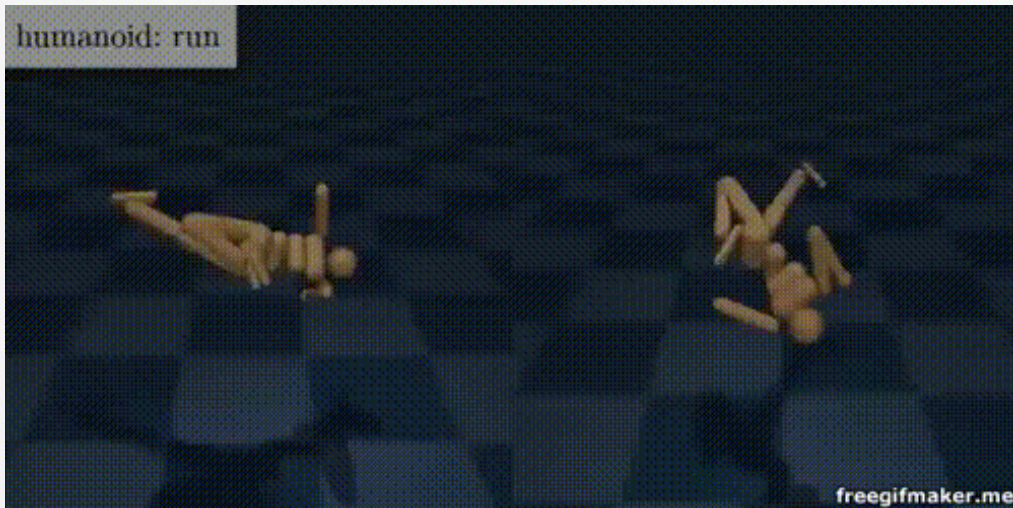
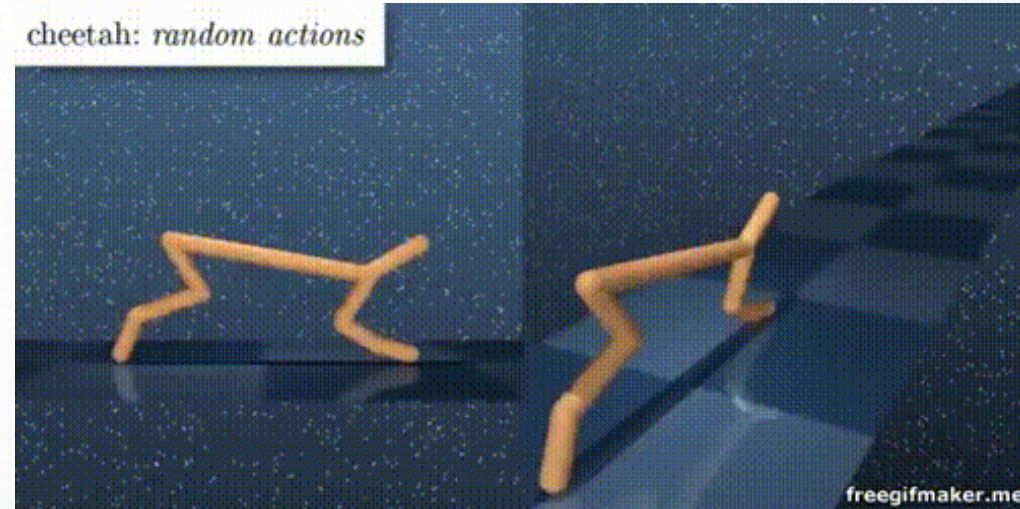
Código em: https://github.com/wau/keras-rl2/blob/master/examples/ddpg_pendulum.py

Environments Documentation

Pendulum-v0

The inverted pendulum swingup problem is a classic problem in the control literature. In this version of the problem, the pendulum starts in a random position, and the goal is to swing it up so it stays upright.

DISTRIBUTED DISTRIBUTIONAL DEEP DETERMINISTIC POLICY GRADIENT (D4PG)



<https://www.youtube.com/watch?v=rAai4QzcYbs>

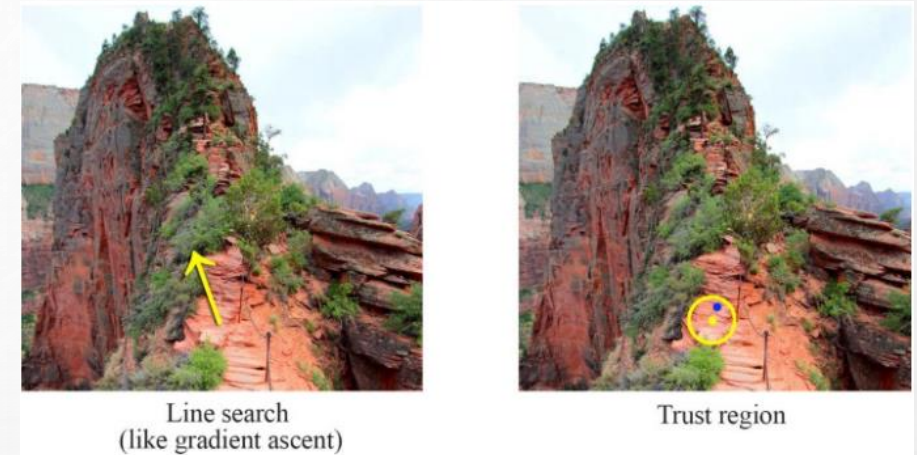
Tassa, Y.; et. al. *Deep Mind Control Suite*, 2018.

TRUST REGION POLICY OPTIMIZATION (TRPO)

Trust Region Policy Optimization
(TRPO)

DESAFIOS DE MÉTODOS DE POLICY GRADIENT

- Gradient Ascent $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$ faz uso de uma **aproximação de primeira ordem da derivada** da Performance e a **taxa de aprendizado α deve ser pequena o suficiente** para lidar com a curvatura da superfície de $J(\theta)$.
- Como determinar α ?
 - Muito alto: Política torna-se instável.
 - Muito baixo: Longo tempo de convergência e baixa eficiência amostral.
- Podemos limitar a diferença entre políticas sucessivas durante o treino por meio de **Trust Regions** (regiões de confiança) e melhorar eficiência amostral por meio de **Importance Sampling**.
- O algoritmo resultante é denominado Trust Region Policy Optimization (**TRPO**).



<https://jonathan-hui.medium.com/rl-trust-region-policy-optimization-trpo-explained-a6ee04e99999>

TRPO: IMPORTANCE SAMPLING

- Importance Sampling: Métodos tradicionais de Policy Gradient possuem baixa eficiência amostral pois precisam simular um episódio inteiro antes de atualizar a política $\pi_\theta(a|s)$ e, após a atualização, **as experiências obtidas pela política anterior não são utilizadas para treinar a nova.**
- Podemos corrigir este problema **estimando $\nabla_{\theta'} J(\theta')$ com base em experiências amostradas de π_θ :**

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} [\nabla_\theta \log \pi_\theta(\tau) \underline{r(\tau)}]$$

reward is calculated from the trajectory using the current policy

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \left(\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'} | \mathbf{s}_{t'})} \right) \left(\sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

refine the
current policy

Sample data
from another policy

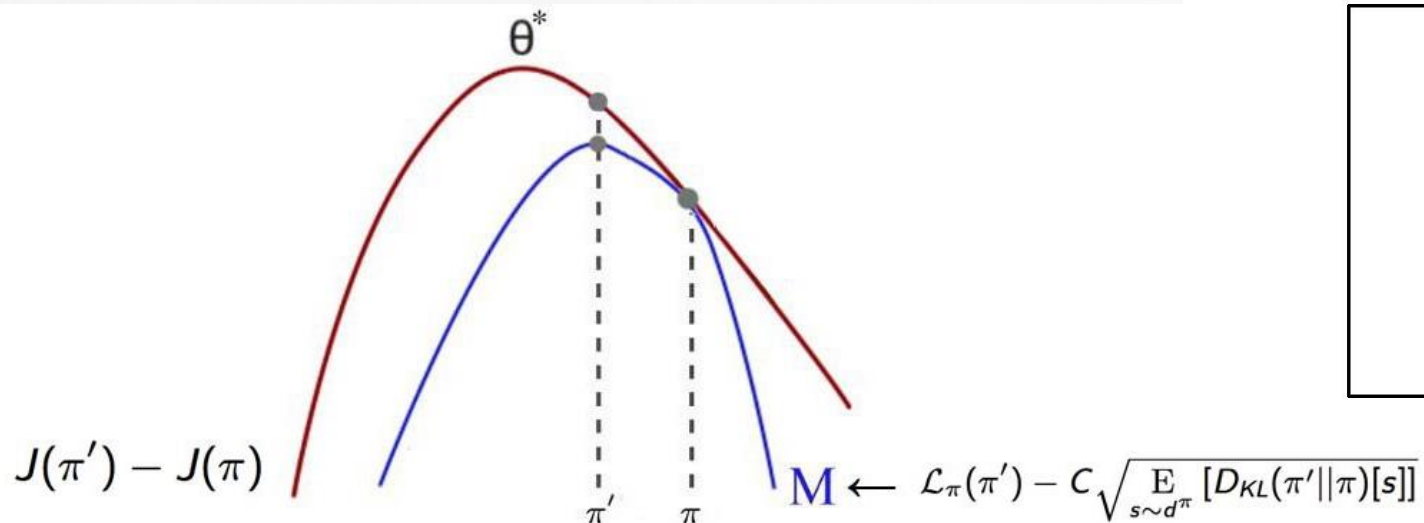
- Para controlar a variância desta estimativa precisamos garantir que as políticas antiga e nova sejam próximas o suficiente → **Limite para Divergência KL entre políticas (Trust Regions)**

TRPO: TRUST REGIONS

- A região de confiança é definida como uma região no espaço de parâmetros θ em que a Divergência KL entre a política nova π' e a política antiga π é menor do que δ :

$$\mathbb{E}_{s \sim d^\pi} [D_{KL}(\pi' || \pi)[s]] \leq \delta$$

- Intuição: Maximizar uma aproximação quadrática local M da diferença entre as Performances de π' e π sujeito à restrição acima.



$$\max_{\pi'} \mathcal{L}_\pi(\pi') - C \sqrt{\mathbb{E}_{s \sim d^{\pi_k}} [D_{KL}(\pi' || \pi)[s]]}$$

or

$$\max_{\pi'} \mathcal{L}_\pi(\pi')$$

$$\text{s.t. } \mathbb{E}_{s \sim d^\pi} [D_{KL}(\pi' || \pi)[s]] \leq \delta$$

<https://jonathan-hui.medium.com/rl-trust-region-policy-optimization-trpo-explained-a6ee04e99999>

TRUST REGION POLICY OPTIMIZATION: TRPO

<https://spinningup.openai.com/en/latest/algorithms/trpo.html>

Algorithm 1 Trust Region Policy Optimization

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: Hyperparameters: KL-divergence limit δ , backtracking coefficient α , maximum number of backtracking steps K

3: **for** $k = 0, 1, 2, \dots$ **do**

- 4: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 5: Compute rewards-to-go \hat{R}_t .
- 6: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 7: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) |_{\theta_k} \hat{A}_t.$$

- 8: Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

where \hat{H}_k is the Hessian of the sample average KL-divergence.

- 9: Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

where $j \in \{0, 1, 2, \dots, K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.

- 10: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 11: **end for**

(1)

Simulação de agente π_{θ_k} e estimativa de gradiente $\hat{g}_k \approx \nabla_{\theta} J(\theta_k)$

(2)

Atualização de parâmetros θ da política π_{θ} de acordo com *KL-divergence constraint*

TRPO garante melhora monotônica de políticas ao longo do treino!

(1): Similar à estimativa de gradiente $\nabla_{\theta} J(\theta)$ do Actor-Critic.
 (2): Estudar algoritmos Natural Policy Gradient e Truncated Natural Policy Gradient (TNPG). TRPO = TNPG + line search

PROXIMAL POLICY OPTIMIZATION (PPO)

Proximal Policy Optimization (PPO)

PROXIMAL POLICY OPTIMIZATION (PPO)

- A matriz Hessiana da divergência KL nos algoritmos NPG, TNPG e TRPO apresenta elevado custo computacional.
- O algoritmo Proximal Policy Optimization (PPO), em vez de impor uma *hard constraint* δ sobre a divergência KL, a incorpora como uma soft constraints na função $J(\pi') - J(\pi)$ a ser otimizada.

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0 , initial KL penalty β_0 , target KL-divergence δ

for $k = 0, 1, 2, \dots$ **do**

Collect set of partial trajectories \mathcal{D}_k on policy $\pi_k = \pi(\theta_k)$

Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

Compute policy update

$$\theta_{k+1} = \arg \max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta || \theta_k)$$

by taking K steps of minibatch SGD (via Adam)

if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \geq 1.5\delta$ **then**

$$\beta_{k+1} = 2\beta_k$$

else if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \leq \delta/1.5$ **then**

$$\beta_{k+1} = \beta_k/2$$

end if

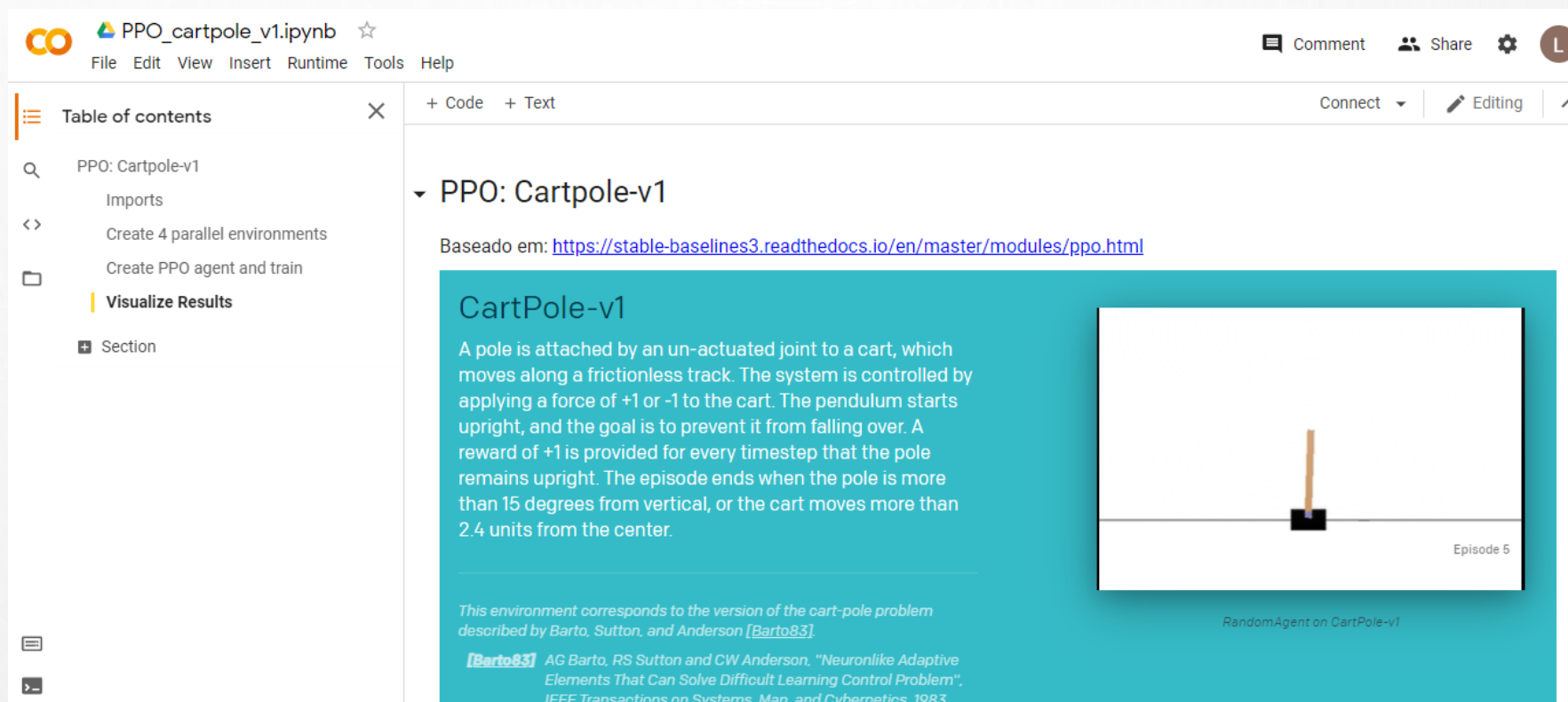
end for

Penalidade sobre divergência
entre políticas sucessivas

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t \right] - \beta \hat{\mathbb{E}}_t [\text{KL}[\pi_{\theta_{\text{old}}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)]]$$

PPO EXEMPLO: CARTPOLE-V1 EM STABLE BASELINES 3

PPO_cartpole_v1.ipynb



The screenshot displays a Jupyter Notebook interface for the file `PPO_cartpole_v1.ipynb`. The top bar includes the Colab logo, file name, and a star icon. The menu bar contains `File`, `Edit`, `View`, `Insert`, `Runtime`, `Tools`, and `Help`. On the right, there are links for `Comment`, `Share`, and a settings icon, along with a user profile icon labeled 'L'.

The left sidebar shows a `Table of contents` with the following items:

- PPO: Cartpole-v1
 - Imports
 - Create 4 parallel environments
 - Create PPO agent and train
 - Visualize Results**
 - + Section

The main content area is titled `PPO: Cartpole-v1` and includes a link to the Stable Baselines3 documentation: <https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html>.

CartPole-v1

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson [Barto83].

[Barto83] AG Barto, RS Sutton and CW Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem", *IEEE Transactions on Systems, Man, and Cybernetics*, 1983.

On the right side of the content area, there is a video player showing a simulation of the CartPole-v1 environment. The video is titled `RandomAgent on CartPole-v1` and shows the pole balancing on the cart at `Episode 5`.

LISTA DE ALGORITMOS DE POLICY GRADIENT MAIS UTILIZADOS

- REINFORCE
- Actor-Critic
- Off-Policy Policy Gradient
- A3C
- A2C
- DPG
- DDPG
- D4PG
- MADDPG
- TRPO
- PPO
- PPG
- ACER
- ACTKR
- SAC
- TD3
- SVPG
- IMPALA

Stable Baselines 3

Name	Box	Discrete	MultiDiscrete	MultiBinary	Multi Processing
A2C	✓	✓	✓	✓	✓
DDPG	✓	✗	✗	✗	✗
DQN	✗	✓	✗	✗	✗
HER	✓	✓	✗	✗	✗
PPO	✓	✓	✓	✓	✓
SAC	✓	✗	✗	✗	✗
TD3	✓	✗	✗	✗	✗

<https://stable-baselines3.readthedocs.io/en/master/guide/algos.html>

<https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#ppo>

TRABALHO T2: DDPG PORTFOLIO MANAGEMENT

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 π_θ e crítico Q_w .
- C) Inicialização e compilação de agente DDPGAgent.
- D) Treino e visualização de resultados.

T2.ipynb



REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction*, The MIT Press (2020). [Cp. 13]
- [2] <https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#actor-critic>
- [3] Williams, R., J. *Simple Statistical Gradient Following Algorithms for Connectionist Reinforcement Learning*, 1992
- [4] Mnih, V.; et. al. *Asynchronous Methods for Deep Reinforcement Learning*, 2016
- [5] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, et al.. *Deterministic Policy Gradient Algorithms*. ICML, Jun 2014, Beijing, China. fhal-00938992
- [6] Lillicrap, T; et. al. *Distributed Distributional Deterministic Policy Gradients*, ICLR, 2018.
- [7] Tassa, Y.; et. al. *Deep Mind Control Suite*, 2018.
- [8] Schulman, J.; et. al. *Trust Region Policy Optimization*, University of California, Berkeley, 2017.
- [9] Schulman, J.; et. al. *Proximal Policy Optimization Algorithms*, University of California, Berkeley, 2017.

Muito obrigado a todos!

Dúvidas