Departamento de Engenharia de Teleinformática - DETi
Universidade Federal do Ceará - UFC

Learning to Control
Desenvolvendo Sistemas Inteligentes

Otacílio "Minho" Neto
{minhotmog@gmail.com}

October 25, 2018

Sumário

Apresentação

Olá!

Eu sou o **Minho**, graduando em Engenharia de Computação pela Universidade Federal do Ceará (DETI/UFC).

Na área de Inteligência Artificial, sou pesquisador principalmente em *Statistical Machine Learning* e *Teoria do Controle*.

Contatos:

- **E-mail:** minhotmog@gmail.com
- **GitHub:** github.com/TioMinho
- **Telegram:** @katchau
- **Twitter:** @MenezesMinho

Apresentação
Do que ele tá falando?

Essa palestra visa discutir o desenvolvimento de **sistemas autônomos** através de métodos estudados pela *Teoria do Controle* e por *Machine Learning*.

Todos os códigos das simulações apresentadas estão disponíveis em um repositório no Github .

Essa apresentação é inspirada no tutorial apresentado por **Benjamin Recht** na edição 2018 da *International Conference on Machine Learning* (IMCL).

Fundamentação do Problema

O problema pode ser expresso, conceitualmente, da seguinte forma:

Como utilizar os dados obtidos de um sistema para determinar suas futuras ações?

O problema pode ser expresso, matematicamente, da seguinte forma:

$$\begin{array}{ll} \textbf{minimize} & \mathbb{E}_e\left[\sum_{i=0}^{N} C_i(x_i, u_i)\right] \\ \textbf{s.t.} & x_{i+1} = f_i(x_i, u_i, e_i) \\ & u_i = \pi_i(\tau_i) \end{array}$$

As principais variáveis são:

- x_i é o *estado* do sistema, e o descreve completamente.
- u_i é a *entrada* do sistema, representando a ação a ser aplicada.
- e_i é um *ruído* no sistema, proveniente do ambiente ou de falhas internas.
- τ_i é uma *trajetória* de estados e entradas observadas até instante i.

O problema pode ser expresso, matematicamente, da seguinte forma:

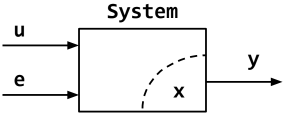
$$\begin{array}{ll} \textbf{minimize} & \mathbb{E}_e\left[\sum_{i=0}^{N} C_i(x_i, u_i)\right] \\ \textbf{s.t.} & x_{i+1} = f_i(x_i, u_i, e_i) \\ & u_i = \pi_i(\tau_i) \end{array}$$

As principais funções são:

- C_i é uma *função de custo* para determinada configuração do sistema. Se quisermos maximizá-la, chamamos de *função de recompensa*.
- f_i é a *função de transição de estado* que determina o próximo estado do sistema.
- π_i é a *policy* do sistema que determina qual próxima ação tomar.

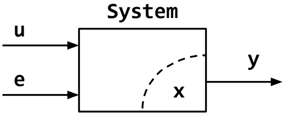
Teoria do Controle

Fundamentação do Problema



A **Teoria do Controle** estuda sistemas, controladores e tudo que os envolve:

- Um **Sistema** é uma entidade física, constituída de diversos componentes que interagem entre si, que reage a estímulos externos produzindo um comportamento dinâmico específico.
- Um **Controlador** é um dispositivo, conectado a um sistema, que pode ser desenvolvido para, com determinada especificações, levar o sistema a um estado específico e/ou mantê-lo em equilíbrio ao compensar distúrbios.



A **Teoria do Controle** estuda sistemas, controladores e tudo que os envolve:

- Um **Sistema** é uma entidade física, constituída de diversos componentes que interagem entre si, que reage a estímulos externos produzindo um comportamento dinâmico específico.
- Um **Controlador** é um dispositivo, conectado a um sistema, que pode ser desenvolvido para, com determinada especificações, levar o sistema a um estado específico e/ou mantê-lo em equilíbrio ao compensar distúrbios.

Uma maneira de analisar os sistemas consiste em descrever suas dinâmicas através de **modelos matemáticos**.

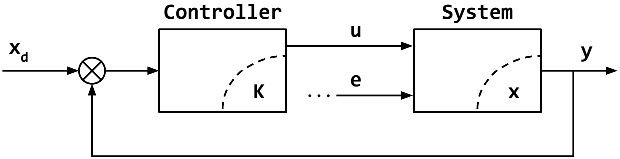
Um tipo de modelagem bastante popular consiste nos *Modelos Lineares de Espaço de Estados*, na forma:

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t) + e(t)$$

Para um sistema com n variáveis de estado e p variáveis de entrada:

- **A** é uma matriz $n \times n$ de transição de estados.
- **B** é uma matriz $n \times p$ de entradas.

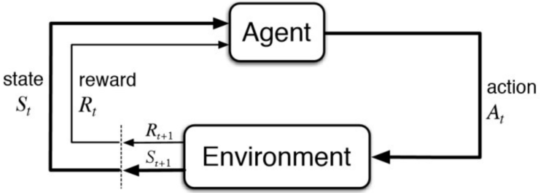
Uma maneira de controlar os sistemas consiste em calcular os sinais de entrada como uma correção a desvios da variáveis de estado a um *set-point*.



Essa configuração, conhecida como **Controlador de Feedback**, toma em conta o estado atual do sistema para tomar a próxima ação.

Reinforcement Learning

Fundamentação do Problema

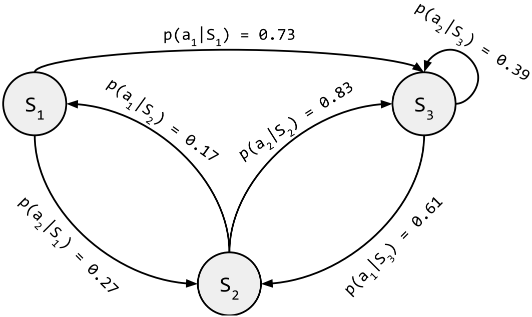


Reinforcement Learning (RL) é a área do Machine Learning que estuda a aprendizagem de agentes, inseridos em um ambiente, para atingir determinado objetivo.

Apesar da distinção de nomeclaturas, os objetivos dessa área são os mesmos da *Teoria do Controle*, porém:

- Os Ambientes e Ações costumam estar em domínios discretos.
- Os processos são considerados estocásticos e com alto grau de incerteza.
- Os dados de treino são dependentes, e o treino é feito online.

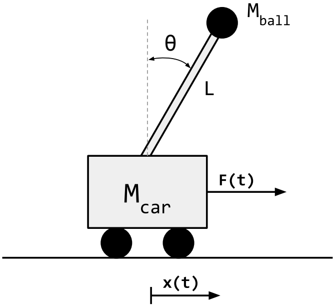
Em RL, as dinâmicas do ambiente são normalmente modeladas como um **Processo de Decisão de Markov** (MDP):



O problema é descobrir um conjunto ótimo de probabilidades para maximizar a recompensa (ou minimizar o custo).

Sistema de Benchmark

Fundamentação do Problema



O **Pêndulo Invertido**, ou **Cart-Pole**, é um sistema de *benchmark* clássico em Teoria do Controle, e possui aplicações interessantes.

Abordagem "Model-Based"

$$\begin{array}{ll} \textbf{minimize} & \mathbb{E}_e\left[\sum_{i=0}^{N} C_i(x_i, u_i)\right] \\ \textbf{s.t.} & \boxed{x_{i+1} = f_i(x_i, u_i, e_i)} \\ & u_i = \pi_i(\tau_i) \end{array}$$

Na primeira abordagem, assumimos que **temos conhecimento geral** sobre as dinâmicas do sistema (ou ambiente) que queremos controlar:

$$x_{i+1} = f(x_i, u_i, e_i) = \mathbf{A}x_i + \mathbf{B}u_i + e_i$$

Nesse caso, podemos gerar uma trajetória τ_i e treinar uma *policy* ótima $\pi_i(\tau_i)$ de maneira *offline*.

Como as ações são determinadas antes mesmo de o controlador (ou agente) atuar, dizemos que a solução é por **Planejamento**.

O Linear Quadratic Regulator (LQR)

Abordagem "Model-Based"

$$\begin{array}{ll} \textbf{minimize} & \mathbb{E}_e \left[\sum_{i=0}^{N} \left(x_i^T \mathbf{Q} x_i + u_i^T \mathbf{R} u_i \right) + x_N^T \mathbf{P_N} x_N \right] \\ \textbf{s.t.} & x_{i+1} = \mathbf{A} x_i + \mathbf{B} u_i + e_i \end{array}$$

Um dos mais estudados controladores de feedback consiste no **Linear Quadratic Regulator** (LQR).

Uma vez que sabemos as dinâmicas, podemos gerar uma trajetória e solucionar a melhor *policy* de várias maneiras:

- Batch Optimization
- Backpropagation
- **Dynamic Programming**

Vamos redefinir a função de custo com a seguinte forma, conhecida como **Action-Value Function** ou **Q-Function**:

$$\mathcal{Q}_s(x, u) = \min_{\mathbf{u}} \ \mathbb{E}_e \left[\sum_{i=s}^{N} \left(x_i^T \mathbf{Q} x_i + u_i^T \mathbf{R} u_i \right) + x_N^T \mathbf{P}_N x_N \right]$$

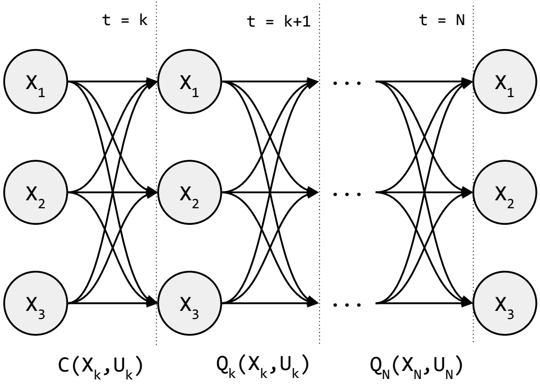
Pelo princípio do *Dynamic Programming*, podemos representar essa função recursivamente (top-down), com a chamada **Equação de Belmann**:

$$\mathcal{Q}_k(x, u) = \left(x_k^T \mathbf{Q} x_k + u_k^T \mathbf{R} u_k \right) + \min_{u'} \mathbb{E}_e \left[\mathcal{Q}_{k+1}(f_k(x_k, u_k, e_k), u') \right]$$

Onde definimos o custo terminal como sendo:

$$\mathcal{Q}_N(x, u) = x_N^T \mathbf{P}_N x_N$$

Podemos entender a Q-Function como o **custo-para-ir** de um instante *k* até o horizonte *N*.



Pelo princípio do *Dynamic Programming*, podemos representar essa função recursivamente (top-down), com a chamada **Equação de Belmann**:

$$\mathcal{Q}_k(x, u) = \left(x_k^T \mathbf{Q} x_k + u_k^T \mathbf{R} u_k\right) + \min_{u'} \mathbb{E}_e \left[\mathcal{Q}_{k+1}(f_k(x_k, u_k, e_k), u')\right]$$

Assumindo uma forma quadrática para $\mathcal{Q}(x, u)$, e plugando $f_k(x_k, u_k, e_k) = \mathbf{A}x_k + \mathbf{B}u_k + e_k$, podemos obter a solução derivando a Equação de Belmann e igualando a zero.

Solução do Linear Quadratic Regulator

$$\mathbf{P}_k = \mathbf{Q} + \mathbf{A}^T \mathbf{P}_{k+1} \mathbf{A} - \mathbf{A}^T \mathbf{P}_{k+1} \mathbf{B} \left(\mathbf{R} + \mathbf{B}^T \mathbf{P}_{k+1} \mathbf{B}\right)^{-1} \mathbf{B}^T \mathbf{P}_{k+1} \mathbf{A}$$

$$\mathbf{u}_k = -\left(\mathbf{B}^T \mathbf{P}_{k+1} \mathbf{B} + \mathbf{R}\right)^{-1} \mathbf{B}^T \mathbf{P}_{k+1} \mathbf{A} x_k = \mathbf{K}_k x_k$$

O fato de a Q-Function ter um formato quadrático, e conhecermos as dinâmicas do sistema, ainda permite que calculemos uma *policy* para horizontes infinitos:

$$\mathcal{Q}_s(x,u) = \min_{\mathbf{u}} \left(\lim_{N\to\infty} \mathbb{E}_e \left[\sum_{i=s}^{N} \left(x_i^T \mathbf{Q} x_i + u_i^T \mathbf{R} u_i \right) + x_N^T \mathbf{P}_N x_N \right] \right)$$

Pelo mesmo procedimento, achamos uma solução surpreendente:

Solução do LQR com Horizonte Infinito

$$\mathbf{P} = \mathbf{Q} + \mathbf{A}^T\mathbf{P}\mathbf{A} - \mathbf{A}^T\mathbf{P}\mathbf{B}(\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B})^{-1}\mathbf{B}^T\mathbf{P}\mathbf{A}$$

$$\mathbf{u}_k = -\left(\mathbf{B}^T\mathbf{P}\mathbf{B} + \mathbf{R}\right)^{-1}\mathbf{B}^T\mathbf{P}\mathbf{A}x_k = \mathbf{K}x_k$$

Modelagem do Sistema Dinâmico

Abordagem "Model-Based"

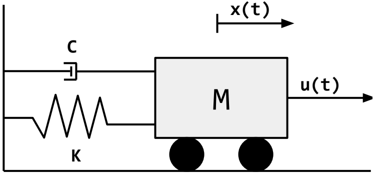
Para completarmos as soluções anteriores, porém, necessitamos definir o modelo através das matrizes **A** e **B**.

Existem duas principais maneiras:

- Modelar o sistema utilizando princípios derivados das Leis da Física.
- Identificar o modelo do sistema utilizando dados de simulações.

As *Leis da Física* nos permite analisar sistemas através de princípios de conservação de massa, conservação de energia, conservação de força, etc.

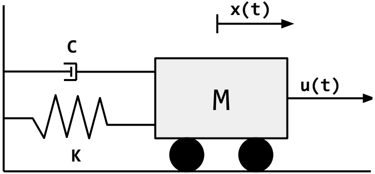
Considere, por exemplo, esse simples sistema mecânico:



Pelo *Princípio de Conservação das Forças*, teremos que:

$$M\ddot{x}(t) + C\dot{x}(t) + Kx(t) = u(t)$$

Considere, por exemplo, esse simples sistema mecânico:



Se considerarmos os estados como sendo $x_1(t) = x(t)$ e :$x_2(t) = \dot{x}_1(t)$, teremos a seguinte representação em Espaço de Estados:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ -K/M & -C/M \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 1/M \end{bmatrix}}_{\mathbf{B}} u$$

A segunda maneira consiste em simular (ou atuar) o sistema e coletar dados de sensores:

$$x_{t+1} \approx \varphi(x_t, u_t) + \mu_t$$

Podemos, então, utilizar de técnicas de *Aprendizagem Supervisionada* pra ajustar o modelo aos dados obtidos:

$$\hat{\varphi} = arg \min_{\varphi} \sum_{t=0}^{N-1} (x_{t+1} - \varphi(x_t, u_t))^2$$

Simulação!

Abordagem "Model-Free"

$$
\begin{aligned}
\textbf{minimize} \quad & \boxed{\mathbb{E}_e\left[\sum_{i=0}^{N} C_i(x_i, u_i)\right]} \\
\textbf{s.t.} \quad & x_{i+1} = f_i(x_i, u_i, e_i) \\
& \boxed{u_i = \pi_i(\tau_i)}
\end{aligned}
$$

Até então assumimos conhecimento sobre o *modelo do sistema dinâmico* (ou do *ambiente*, na linguagem do RL). Mas e se não pudermos (ou quisermos) ter esse conhecimento?

Existem duas principais abordagens *"Model-Free"*:

- Aprender a função de custo por **Approximate Dynamic Programming**, ou
- Aprender diretamente a *policy* por **Policy Gradient**.

Approximate Dynamic Programming

Abordagem "Model-Free"

$$\begin{aligned} \textbf{minimize} \quad & \mathbb{E}_e\left[\sum_{i=0}^{N} C_i(x_i, u_i)\right] \\ \textbf{s.t.} \quad & x_{i+1} = f_i(x_i, u_i, e_i) \\ & u_i = \pi_i(\tau_i) \end{aligned}$$

A intenção é aproximar a mesma solução encontrada para o LQR, mas sem saber a estrutura de $f_i(x_i, u_i, e_i)$, apenas a modelando como uma MDP.

- A fórmula recursiva de Belmann:

$$\mathcal{Q}_k(x, u) = C_k(x_k, u_k) + \min_{u'} \mathbb{E}_e\left[\mathcal{Q}_{k+1}(f_k(x_k, u_k, e_k), u')\right]$$

- *Policy* Ótima:

$$\pi_k(\tau_k) = arg \min_{u} \mathcal{Q}_k(x_k, u)$$

Uma solução bastante simples, e muito popular, consiste em, primeiramente, considerar um horizonte infinito com um *fator de desconto*:

$$\mathcal{Q}_k(x, u) = \min_u \quad \mathbb{E}_e \left[\sum_{i=k}^{\infty} \gamma^i C(x_i, u_i) \right]$$

A fórmula recursiva de Belmann se torna:

$$\mathcal{Q}(x_k, u_k) \approx C(x_k, u_k) + \gamma \min_{u'} \mathcal{Q}(x_{k+1}, u') + \mu_k$$

Podemos solucionar iterativamente a Q-Function pela seguinte regra:

Q-Learning or Stochastic Approximation

$$\mathcal{Q}_{new}(x, u) = (1 - \eta) Q_{old}(x, u) - \eta \left(C(x_k, u_k) + \gamma \min_{u'} Q_{old}(x_{k+1}, u') \right)$$

Q-Learning Algorithm

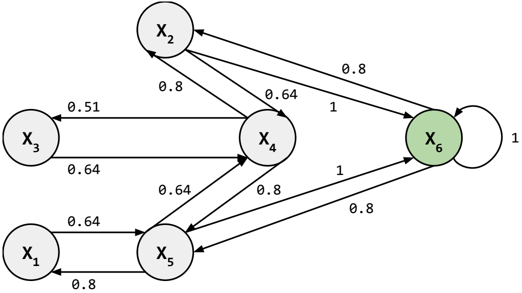
1. Inicialize a *Q-Table* com zeros;
2. Gere um número aleatório r entre 0 e 1;
3. Se $r < \epsilon$, então tome a ação $u_k = \pi(\tau_k) = arg\min_u \mathcal{Q}_k(x_k, u)$;
4. Senão, então tome uma ação aleatória;
5. Observe o próximo estado x_{k+1} e o custo $C(x_k, u_k)$;
6. Atualize o valor equivalente na *Q-Table* com:

$$\mathcal{Q}_{new}(x, u) = (1 - \eta) Q_{old}(x, u) - \eta \left(C(x_k, u_k) + \gamma \min_{u'} Q_{old}(x_{k+1}, u') \right)$$

7. Faça $k = k + 1$ e verifique a condição de término. Caso seja falsa, retorne ao passo 2;

Veja um exemplo de uma Q-Table:

	u_1	u_2	u_3	u_4	u_5	u_6
x_1	0	0	0	0	-0.8	0
x_2	0	0	0	-0.64	0	-1
x_3	0	0	0	-0.64	0	0
x_4	0	-0.80	0.51	0	-0.8	0
x_5	-0.64	0	0	-0.64	0	-1
x_6	0	-0.80	0	0	-0.8	-1



Simulação!

Policy Gradient

Abordagem "Model-Free"

$$\begin{aligned} \textbf{minimize} \quad & \mathbb{E}_e \left[\textstyle\sum_{i=0}^{N} C_i(x_i, u_i) \right] \\ \textbf{s.t.} \quad & x_{i+1} = f_i(x_i, u_i, e_i) \\ & u_i = \pi_i(\tau_i) \end{aligned}$$

Imagine que não sabemos as dinâmicos do sistema (ambiente), e nem sequer nos preocupamos em saber a função custo das ações do controlador (agente). Será que é possível solucionar o problema mesmo assim?

Os métodos de **Policy Gradient** baseiam-se em diretamente treinar uma *policy* para o problema, sem consultar funções de custo ou prever os estados do sistema.

Primeiramente, deveremos considerar uma *policy* como uma amostragem de uma distribuição parametrizada:

$$\pi_i(\tau_i) = p(u|x_i, \theta)$$

Podemos encarar o problema como uma otimização estocástica com a seguinte função de custo:

$$\min J(\theta) = \min_{\theta} \mathbb{E}_{p(u|x_i,\theta)} \left[\Phi(x_i, u_i)\right] \geq \min \Phi(x_i, u_i)$$

O gradiente pode ser calculado com um truque esperto de cálculo:

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{p(u|x_i,\theta)}\left[\Phi(x_i,u_i)\right] \\
\nabla_\theta J(\theta) &= \int \Phi(x_i,u_i)\nabla_\theta p(u|x_i,\theta)du \\
&= \int \Phi(x_i,u_i)\left(\frac{\nabla_\theta p(u|x_i,\theta)}{p(u|x_i,\theta)}\right)p(u|x_i,\theta)du \\
&= \int (\Phi(x_i,u_i)\nabla_\theta \log p(u|x_i,\theta))p(u|x_i,\theta)du \\
&= \mathbb{E}_{p(u|x_i,\theta)}\left[\Phi(x_i,u_i)\nabla_\theta \log p(u|x_i,\theta)\right]
\end{aligned}
$$

Simulação!

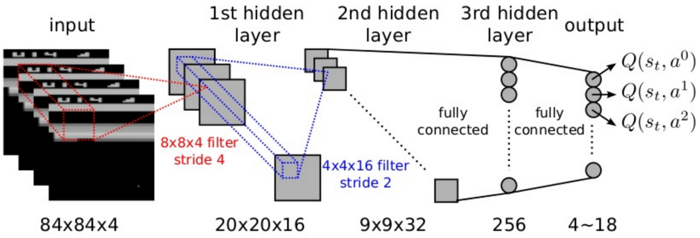
Podemos definir um algoritmo de *Policy Gradient* no formato do nosso querido *gradiente descendente*:

Monte-Carlo Policy Gradient

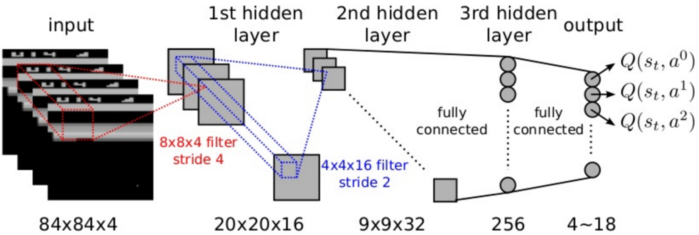
1. **Amostre** $u_i \sim p(u_i|x_i, \theta)$;
2. **Compute** $G(u_i, \theta_i) = \Phi(u_i, x_i)\nabla_\theta \log p(u|x_i, \theta)$;
3. **Atualize** $\theta = \theta - \alpha G(u_i, \theta_i)$;
4. Verifique convergência. Caso seja falso, volte ao Passo 1.

Obs.: São preferíveis outros métodos de otimização mais robustos que o gradiente descendente simples.

Considerações Finais



- Em **ADP**, uma Rede Neural Profunda (DNN) é utilizada no lugar da *Q-Table*, tornando-se então a parametrização da *Q-Function*.
- Em **PG**, uma Rede Neural Profunda (DNN) é utilizada no lugar da distribuição $\log p(u_i|x_i, \theta)$, parametrizando diretamente os estados para possíveis ações.



- Em **ADP**, uma Rede Neural Profunda (DNN) é utilizada no lugar da *Q-Table*, tornando-se então a parametrização da *Q-Function*.
- Em **PG**, uma Rede Neural Profunda (DNN) é utilizada no lugar da distribuição $\log p(u_i|x_i, \theta)$, parametrizando diretamente os estados para possíveis ações.

Comparação dos Métodos:

	Descrição	Estados	Ações
LQR por DP	Prevê estados e otimiza ações *offline*	Contínuo	Contínuo
ADP por Q-Learning	Aprende a função de custo *online*	Discreto	Discreto
Policy Gradient	Busca a melhor *policy* pelo espaço	Contínuo	Contínuo

Dúvidas?

Thank you!