

# Pesquisa e Implementação de técnica de Aprendizado de Máquina para Jogos Digitais

Implementando a técnica Q Learning no ambiente Cart Pole fornecido pela OpenAI

UNISINOS/ 2018

Andrew Floriano

# Introdução

Meu objetivo neste trabalho consiste em apresentar a técnica de *Reinforcement Learning* aplicando diretamente no ambiente Cart Pole da OpenAI, também conhecido como pendulo invertido. O ambiente consiste em um carrinho com uma haste que pode se deslocar angularmente. A haste tende a cair sob efeito da gravidade, portanto é um sistema naturalmente instável.

O sistema pode ser estabilizado aplicando uma força horizontal ao carrinho, fazendo a haste ficar na vertical.

*Reinforcement learning* é um conceito geral que pode ser simplesmente descrito com um agente que realiza ações em um ambiente para maximizar sua recompensa cumulativa. A ideia subjacente é muito realista, onde, da mesma forma que os seres humanos na vida real, os agentes nos algoritmos de *Reinforcement Learning* são incentivados com **punições** por ações ruins e **recompensas** por ações boas.

A ferramenta que irei usar é o OpenAI Gym, a OpenAI fornece um *toolkit* baseado em Python para a pesquisa e desenvolvimento de algoritmos de aprendizado por reforço. A maior vantagem é que o OpenAI fornece uma interface unificada para trabalhar com esses ambientes e cuida da execução da simulação enquanto você se concentra nos algoritmos de aprendizado por reforço.

# Metodologia

## Modelando o Cart Pole no ambiente simulado

Em termos de aprendizado de máquina, o CartPole é basicamente um problema de classificação binária. Existem quatro características como entradas, que incluem a posição do carrinho, sua velocidade, o ângulo do polo com o carrinho e sua derivada (ou seja, quão rápido o polo/lado está "caindo").

A saída é binária, ou seja, 0 ou 1, correspondendo a "esquerda" ou "direita".

Um desafio é o fato de que todas as quatro características são valores contínuos (números de ponto flutuante), que, ingenuamente, implicam um espaço infinitamente grande.

```
24 def create_bins():
25     # obs[0] -> cart position --- -4.8 - 4.8
26     # obs[1] -> cart velocity --- -inf - inf
27     # obs[2] -> pole angle --- -41.8 - 41.8
28     # obs[3] -> pole velocity --- -inf - inf
29
30     bins = np.zeros((4,10)) #Array de 4x10
31     bins[0] = np.linspace(-4.8, 4.8, 10)
32     bins[1] = np.linspace(-5, 5, 10)
33     bins[2] = np.linspace(-.418, .418, 10)
34     bins[3] = np.linspace(-5, 5, 10)
```

Os maiores e principais parâmetros do algoritmo, que incluem ALPHA (**taxa de aprendizado**), Algoritmo Épsilon (**taxa de exploração**) e GAMA (**fator de desconto**).

```
12 MAXSTATES = 10**4 #Iremos ter no maximo 10.000 estados
13 GAMMA = 0.9
14 ALPHA = 0.01
15
```

O Alpha é usado para suavizar as atualizações e torná-las menos radicais ao nível do código para cada ação, o que, em primeiro lugar, evite erros possíveis erros.

GAMA é usada para penalizar o agente se demorar muito para atingir seu objetivo. No entanto, neste caso, o gama é definido como constante 1, já que é nosso objetivo "sobreviver" o maior tempo possível.

O Algoritmo Épsilon regula as ações tomadas do agente gradativamente. Consequentemente, em vez de escolher a melhor ação em um estado, é escolhida uma ação aleatória. Isso deve impedir que o algoritmo fique preso nos locais pequenos, por exemplo se uma escolha ruim fosse feita no começo, o agente jamais descobriria nenhum outro caminho, potencialmente melhor.

A lógica do algoritmo de Epsilon-Greedy usa um número random da Numpy que é menor que o a dele mesmo, a partir daí, o algoritmo precisa fazer uma ação randômica, que busca o maior valor possível da recompensa.

```
def play_one_game(bins, Q, eps=0.5):
    observation = env.reset()
    done = False
    cnt = 0 #numero de movimentos em um episodio
    state = get_state_as_string(assign_bins(observation, bins))
    total_reward = 0

    while not done:
        cnt += 1

        if np.random.uniform() < eps:
            act = env.action_space.sample() # Algoritmo que usa a estrategia de Epsilon-Greedy
        else:
            act = max_dict(Q[state])[0]

        observation, reward, done, _ = env.step(act)

        total_reward += reward
```

Inicializando a função de *Q-learning*

```
def initialize_Q(): #Inicializa o Q learning
    Q = {}

    all_states = get_all_states_as_string()
    for state in all_states:
        Q[state] = {}
        for action in range(env.action_space.n):
            Q[state][action] = 0 #estado do reward inicial e do que o programa espera é ZERO
    return Q
```

Para finalizar, parametrizo um número alto de jogos para o cobrir espaço suficiente até ele conseguir aperfeiçoar o movimento.

```
def play_many_games(bins, N=10000): ##setar um numero alto de jogos para ele jogar
    Q = initialize_Q()
```

A punição para o agente precisa ser severa, mas não é correto considerar a queda do pêndulo (falha) como uma punição, nesse caso, parametrizo para -300 a variável Reward, assim o agente tenta sempre buscar o máximo de valor possível, que nesse caso é 200.

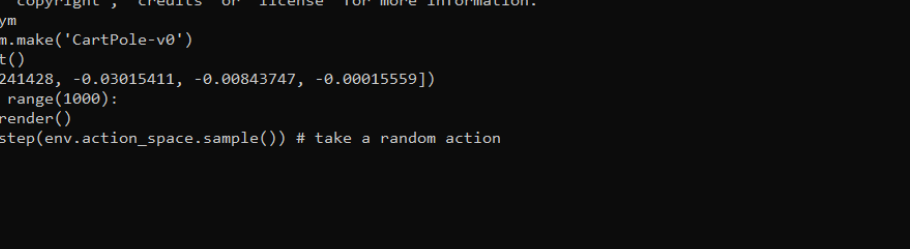
```
if done and cnt < 200:  
    reward = -300
```

Por fim, finalizo o algoritmo com uma função que me diz a performance no final de todas as ações.

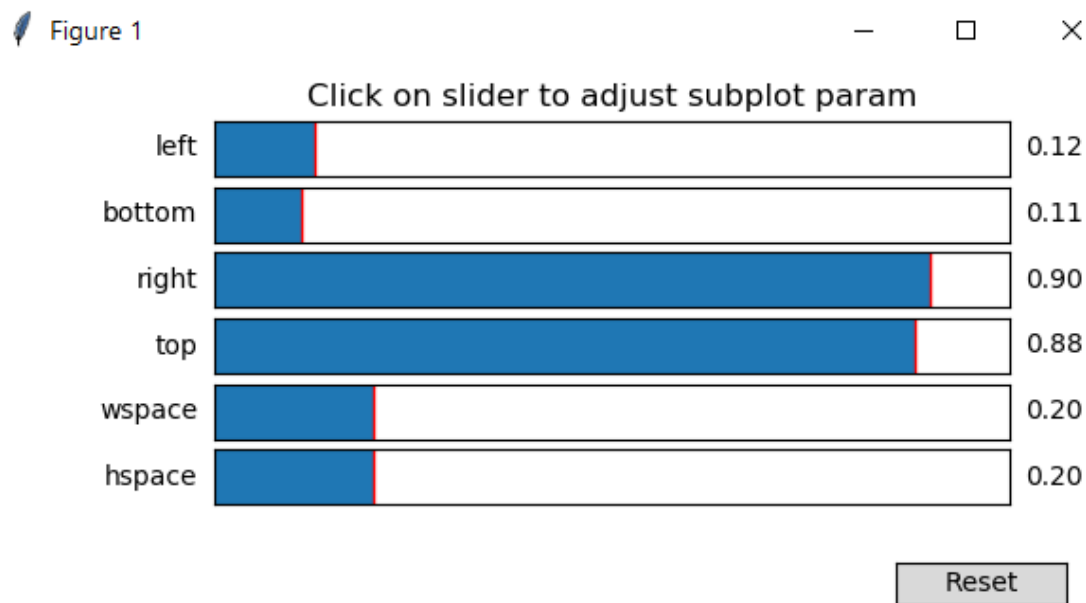
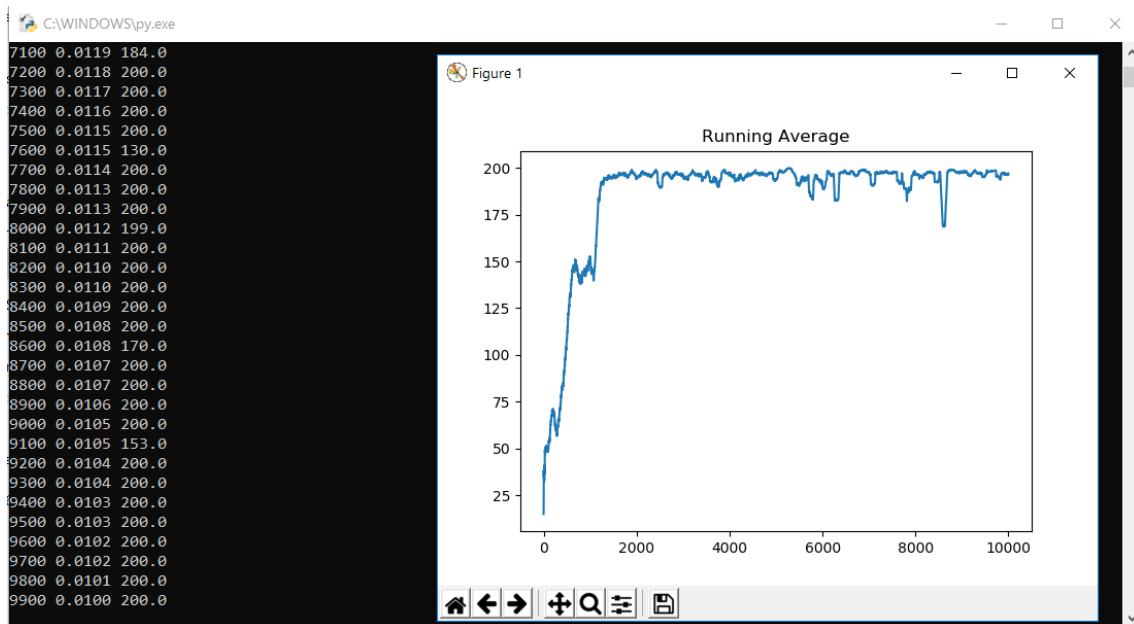
```
def plot_running_avg(totalrewards):  
    N = len(totalrewards)  
    running_avg = np.empty(N)  
    for t in range(N):  
        running_avg[t] = np.mean(totalrewards[max(0, t-100):(t+1)])  
    plt.plot(running_avg)  
    plt.title("Running Average")  
    plt.show()
```

## RESULTADO

Em pouco menos de 2000 episódios, o agente conseguiu se manter estável e conseguindo o máximo de recompensa possível. A redução de dimensionalidade foi a chave para a melhor performance do Q Learning.



```
Python 3.5 (64-bit)
Python 3.5.4 (v3.5.4:3f56838, Aug 8 2017, 02:17:05) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import gym
>>> env = gym.make('CartPole-v0')
>>> env.reset()
array([ 0.04241428, -0.03015411, -0.00843747, -0.00015559])
>>> for _ in range(1000):
...     env.render()
...     env.step(env.action_space.sample()) # take a random action
```



## CONSIDERACOES FINAIS

Com os resultados apresentados nesse trabalho, acredito que consegui atingir os objetivos iniciais de resolver o problema do pêndulo invertido através de agentes de aprendizagem por reforço. Implementei uma nova forma de interpretar as variáveis de percepção permitindo a um agente simples (como é o caso de Q-Learning) resolver com sucesso este problema.

Como de costume, esse algoritmo tem seus prós e contras. Se a faixa de pesos que soluciona com sucesso o problema é pequena, a busca aleatória do algoritmo pode levar um longo tempo até achar a melhor maneira de conseguir a máxima recompensa



## REFERENCIAS

<https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947>

<https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288>