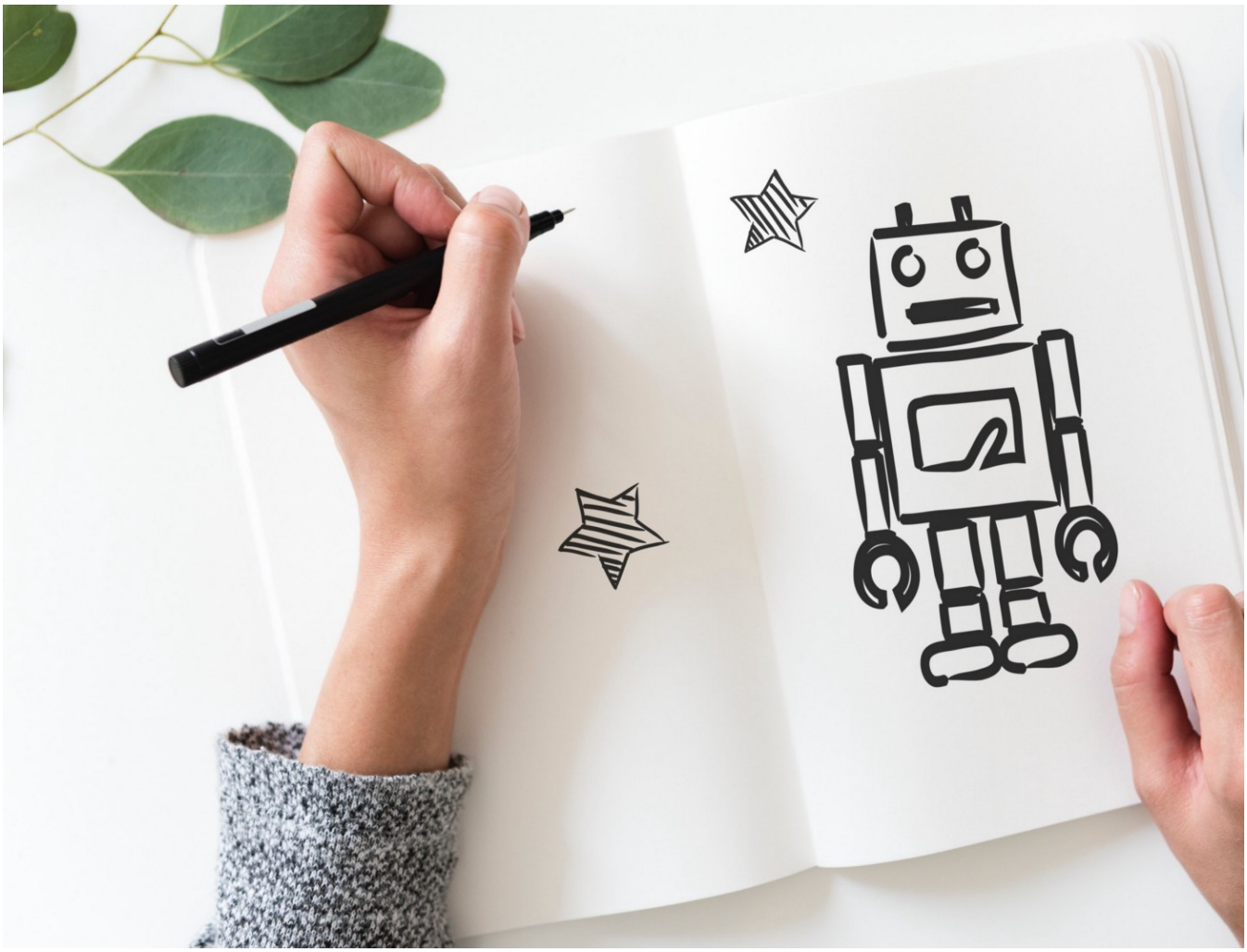
# OpenAI GYM | Resolvendo CartPole

Lucas Ricciardi Mar 9, 2019 · 5 min read



## Introdução

Citando a descrição do site, OpenAI GYM é uma ferramenta para desenvolvimento e comparação de algoritmos de **aprendizagem por reforço** que eu acabei descobrindo

durante a faculdade quando eu buscava entender mais sobre **inteligência artificial** e como ela poderia ser usada para dese



Essa plataforma atendeu perfeitamen
iniciar no mundo da inteligência artifi
como:

- Funcionamento de uma **rede neu**
- Processo de **filtragem de dados**;
- Estatísticas para **aferir a precisão da rede**;

Irei então compartilhar neste guia como foi o processo que utilizei para resolver o cenário e comentar sobre meus aprendizados durante o processo.

## Preparação do ambiente

Requisitos:

- Ubuntu
- Python

Para preparação do ambiente, iremos utilizar o **Pipenv**, uma ferramenta para gerenciamento de ambientes para a linguagem Python.

Para começar crie uma pasta para o projeto, navegue até ela e instale as depedências usando o Pipenv:

```
mkdir cart-pole

cd cart-pole

pipenv install gym scikit-learn

touch main.py
```

Aguarda o fim da instalação dos pacotes e verifique no diretório que foram criados os arquivos:

- main.py

- Pipfile
- Pipfile.lock



Agora, para verificarmos a sanidade d
executar nosso script sem problemas,
**main.py**, importe as dependências qu
`hello, world` para verificarmos que tu

Para executar o programa dentro de nosso ambiente virtual criado utilizando o Pipenv digite:

```
pipenv run python main.py
```



Se você ver um *hello, world* impresso n
aqui e você pode continuar com o tuto
passo para ser completado.

## Entendendo o problema

Copie e cole o trecho de código acima, depois execute ele usando o mesmo comando do tópico acima:

```
pipenv run python main.py
```



Você verá uma saída similar a apresen

```
{
 "name": "RANDOM DATA REPORT"
 "max_score": 114,
 "min_score": 8,
 "average_score": 22.073,
 "median_score": 19.0,
 "standard_deviation_score": 11.716811468996163,
 "variance_score": 137.283671
}
```

O programa acima roda uma simulação do ambiente **CartPole** e depois gera um relatório com dados estatísticos sobre a simulação como a pontuação máxima e mínima alcançadas, a média, a mediana, o desvio padrão e a variância das pontuações.

Cada simulação contém um número de episódios, definido pela variável `EPISODES`, e cada episódio contém até no máximo 200 frames. O objetivo é que nosso controlador consiga equilibrar o CartPole pelo máximo de frames possível. No relatório acima podemos ver que um controlador randômico conseguiu um score máximo de 114 frames, e um mínimo de 8 frames.

> *É possível visualizar a simulação alterando o paramêtro **view** para True no código mas utilizarei eles desligado para podermos comparar os dados.*

A simulação fornece dois parâmetros para que possamos implementar nossos controladores, eles são:

- **observation** — São informações do ambiente, é com base nele que iremos tomar nossa ação.
- **action** — São as ações que podemos realizar dentro de nosso ambiente. No caso desta simulação elas se resumem a empurrar para esquerda ou para a direita, ou seja, 0 ou 1.

## Resolvendo o problema

O leitor deve estar se perguntando por que eu não descrevi o parâmetro **observation** no tópico anterior. Na verdade ele é u
valores máximos e mínimos que esses



```
env.observation_space.high
# [4.8000002e+00 3.4028235e+3

env.observation_space.low
# [-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
```

Podemos ver o range que esses valores podem assumir. Contudo para a solução que irei apresentar não será necessário que conheçamos a fundo o que são e o que significam esses valores !

Pensem bem, nosso objetivo será encontrar uma função que receba um vetor de 4 elementos (observation) e nos retorne 0 ou 1 (action) de tal forma que essas ações maximizem o nosso score. O que estes números significam não importa, baste que encontremos a função certa ! Para isso vamos utilizar um tipo de rede neural conhecido como **MultiLayer Perceptron**.

Copie e code o código alterado e o execute. Dessa vez ele irá gerar um segundo report que é o resultado da nossa rede neural atuando na simulação. No meu caso o report foi:


```
{
 “name”: “RANDOM DATA REPORT”,
 “max_score”: 110,
 “min_score”: 8,
 “average_score”: 21.836,
 “median_score”: 18.0,
 “standard_deviation_score”: 11.80004677956829,
 “variance_score”: 139.24110399999998
}
{
 “name”: “AI DATA REPORT”,
 “max_score”: 200,
 “min_score”: 140,
 “average_score”: 197.273,
 “median_score”: 200.0,
 “standard_deviation_score”: 9.482851417163511,
 “variance_score”: 89.92447100000003
}
```


Uau ! Vejam a perfomance do controlador inteligente em comparação com o controlador randômico. A média da rede neural, em **1000 simulações** rodadas, foi de **197.273** ! O score máximo permitido pela simulação é de **200** ! Realmente impressionante, mas como tudo aconteceu ?

## Explicando o sistema

Uma rede neural é um **aproximador universal**, ou seja, qualqer função contínua pode ser aproximada utilizando uma rede neural. Para isto basta que tenhamos amostrar do **domínio** e do **contra-domínio** da função.

No inicío do programa, rodamos 1000 vezes uma simulação randômica. Cada vez que o nosso controlador randômico tomav



`np.random.randint` que segundo a doc

seguindo uma distribuição uniforme d

não importa o estado do ambiente, ele

ver que o máximo score que ele conseg

8 pontos. Como rodamos a simulação

bastante para conseguiur diferentes scores. *"Ah mas então se a gente desse azar e não conseguisse nenhuma pontuação alta ?"*, é um cenário que poderia vir a acontecer, por isso temos que rodar a simulação um número elevado de vezes para que todos os tipos de cenários tenham mais chances de acontecer.

Tendo o resultado de diversas simulaçõs randômicas, nós alimentamos nossa rede **somente com os resultados de jogos que obtiveram um bom resultado !** A variável `THRESHOLD` armazena uma pontuação mínima, só aceitamos treinar nossa rede com simulações que obtiveram um score igual ou superior ao nosso mínimo (que no meu caso, defini como 40 pontos).

Mais acima, eu disse que os valores do parâmetro **observation** não importavam e também disse que a rede neural era um aproximador universal. Agora vamos encaixar as peças, nosso controlador deve ser uma **função que receba um vetor de 4 elementos (observation) e nos retorne 0 ou 1 (action) de tal forma que essas ações maximizem o nosso score**. Para aproximarmos uma função usando uma rede neural, basta que tenhamos amostras do **domínio** e do **contra-domínio.** Conseguiram já encaixar as peças ?

Filtrando somente os jogos que obtiveram um score aceitável, temos amostras do dóminio (observations) e do contra-domínio (actions). Como nossa rede irá transformar um vetor de 4 elementos e um 0 ou 1 não importa, sabemos que ela irá aproximar esta função e que os resultados que ela irá obter serão bons !

Por isso quando queremos ensinar uma rede neural a identificar fotos de gatos, alimentamos elas com fotos de gatos e dizendo *"Hey, isto é um gato !"*. A rede irá aproximar uma função que leva um domínio em um contra-domínio, então basta encontrarmos uma relação entre os dois que a rede irá tratar de "implementar" esta relação para nós.

## Conclusão

E com isso chegamos ao fim, sugiro fortemente que o leitor interessado tente resolver outros ambientes disponíveis no site d
interessante e divertida de aprender u



Machine Learning Python Developer