Reinforcement Learning Aplicado no Ambiente *Car Racing* do Gym OpenAI
Exame de Inteligencia Artificial Aplicada a Robótica Móvel (CT-213)

Diogo Bueno Rodrigues
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brasil
Email: diogo.rodrigues.8911@ga.ita.br

Luís Fernando Ribeiro Mendonça
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brasil
Email: luis.mendonca.8958@ga.ita.br

Luiz Felipe de Brito Ramos
Instituto Tecnológico de Aeronáutica
São José dos Campos, Brasil
Email: luiz.ramos.889@ga.ita.br

***Resumo*—Neste estudo, aplicamos e comparamos três algoritmos de aprendizado por reforço – Deep Q-Learning (DQN), Proximal Policy Optimization (PPO) e Deep Deterministic Policy Gradient (DDPG) – no ambiente Car Racing do OpenAI Gym. A implementação dos métodos foi ajustada para lidar com o espaço de ações contínuas do ambiente. Utilizamos a arquitetura convolucional para extrair características visuais dos estados observados e uma função de recompensa ajustada para promover comportamentos desejáveis. Os resultados demonstram que, apesar dos desafios inerentes ao aprendizado por reforço em espaços contínuos, todos os algoritmos conseguiram aprender políticas eficazes para a condução do carro. Observamos que o PPO apresentou desempenho superior em termos de estabilidade e convergência em comparação com DQN e DDPG.**

1. Introdução

Embora os algoritmos de aprendizado por reforço datem dos anos 1950, como resultado das equações de Bellman [1] - as quais, quando resolvidas, geram uma política ótima para o comportamento de um agente -, a tecnologia ainda se encontra em fase de desenvolvimento, muito em parte devido à dificuldade da sua implementação em ambientes reais, nos quais a dimensionalidade dos espaços, a dificuldade de estabelecer relações entre ação e recompensa, o ruído gerado pela arquitetura, e dentre outros problemas, reduzem a factibilidade da aplicação do método. Apesar disso, entretanto, com o aumento do poder computacional disponível para o treinamento de redes neurais mais complexas em anos recentes e a capacidade dessas técnicas em resolver questões com modelagens mais custosas - em contraparte à abordagem das técnicas de controle, que são utilizadas para modelos mais simples -, a pesquisa e os esforços produzidos sobre o assunto continuam em larga escala nos âmbitos acadêmico e industrial. Um desses esforços é levantado pela organização *OpenAI* com seu projeto *Gym*: um repositório na plataforma *GitHub* que conta com ambientes virtuais em que a extração de parâmetros para a engenharia de uma recompensa é facilitada, a fim de instigar o estudo sobre os algoritmos de aprendizado por reforço.

O contexto, nesse caso, é prolífico à exploração das diferenças entre implementações distintas do aprendizado por reforço, as quais, devido à dependência em técnicas de tentativa e erro, podem ser melhor analisadas em ambientes simulados. Dessa forma, a fim de contemplar o desempenho das redes em ambientes com espaços de ação contínuos, decidiu-se por utilizar o módulo *Car Racing* (versão v_2) do repositório *Gym*, que opera um carro percorrendo uma pista, com uma visão de cima para baixo. A *API*, nesse caso, permite o controle dos seguintes parâmetros: a direção do carro, representada pelo parâmetro $s \in [-1, 1]$; se o carro está acelerando ou não, representado pelo parâmetro $a \in \{0, 1\}$; e se o carro está freando ou não, representado pelo parâmetro $b \in \{0, 1\}$. Além disso, o estado produzido do ambiente produzido é identificado por uma matriz de *pixels* (espectro *RGB*) com dimensão 96×96, representando a região no entorno do carro.

Quanto aos algoritmos utilizados para o solução do problema, foram empregados os métodos *Deep Q-Learning* (*DQN*), *Proximity Police Optimization* (*PPO*) e *Deep Deterministic Policy Gradient* (*DDPG*).

2. Metodologia

2.1. Deep Q-Learning

Apesar de o problema apresentado pelo ambiente *Car Racing* ser continuo, discretizou-o em 13 possíveis ações para a implementação do método Deep-Q-Learning, as ações selecionadas seguem na tabela 1.

Para a obtenção do estado atual do carro utilizou-se *prints* da tela representados por um tensor de dimensões (96, 96, 3) em *RGB*, realizou-se, então, um preprocessamento da imagem, aplicando um filtro em cinza e realizando um *blur* gaussiano. Por fim, empilhou-se os três últimos *frames*, passando essas informações para a rede neural convolucional, essa sendo definida conforme a tabela 4, com otimizador Adam e *loss* como erro quadrático médio.

Para a recompensa utilizou-se a padrão do ambiente *Car Racing*, no qual recebe-se +3 caso percorra um bloco da pista e -0.1 a cada *frame* ocorrido, porém com um multi-

Tabela 1. AÇÕES DELIMITADAS NO DEEP-Q-LEARNING

Ações	Direção (-1 - 1)	Aceleração (0 - 1)	Freio (0 - 1)
1	0	0	0
2	0	1	0
3	0	0	0.2
4	1	1	0
5	-1	1	0
6	1	0	0.2
7	-1	0	0.2
8	0.5	0.5	0
9	-0.5	0.5	0
10	0.5	0	0.1
11	-0.5	0	0.1
12	1	0	0
13	-1	0	0

Tabela 2. REDE NEURAL CONVOLUCIONAL

Layer	Neurons	Kernel Size	Stride	Activation Function
Conv2D	6	7	3	ReLU
MaxPooling2D	-	2	-	-
Conv2D	12	4	1	ReLU
MaxPooling2D	-	2	-	-
Dense	216	-	-	ReLU
Dense	13	-	-	None

plicador de 1.5 caso a ação selecionada possua componente de aceleração diferente de zero.

2.2. Policy Optimization Algorithm (PPO)

O treinamento do algoritmo PPO foi realizado executando um episódio por vez, com as ações da rede neural sendo adicionadas de um ruído adicional para promover a exploração de novos estados. Ao término de cada execução, foram salvos o estado, o reward adquirido, a ação realizada e a ação que a rede executaria naquele estado caso a randomização para exploração fosse retirada. Esses dados foram adicionados a uma lista que foi utilizada na função de treino, onde foram calculados o *discounted reward* e a *advantage*. Essa lista só é reiniciada após a parada de uma série de treinos, o que pode ser feito a qualquer momento pois os pesos da rede neural sempre são salvos após a finalização de um treino.

O *discounted reward* (R_t) é calculado levando em conta a propagação da ação para os próximos estados:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

onde γ é o fator de desconto e r_{t+k} é o reward no tempo $t + k$. Foi utilizado γ = 0.96.

A *advantage* (A_t) é calculada como a diferença entre o *discounted reward* e os valores previstos pela rede crítica do PPO ($V(s_t)$):

Tabela 3. REDE NEURAL CONVOLUCIONAL PARA A POLICY

Layer	Neurons	Kernel Size	Stride	Activation Function
Conv2D	6	7	3	ReLU
MaxPooling2D	-	2	-	-
Conv2D	12	4	1	ReLU
MaxPooling2D	-	2	-	-
Dense	216	-	-	ReLU
Dense	4	-	-	Sigmoid

Tabela 4. REDE NEURAL CONVOLUCIONAL PARA A CRITIC

Layer	Neurons	Kernel Size	Stride	Activation Function
Conv2D	6	7	3	ReLU
MaxPooling2D	-	2	-	-
Conv2D	12	4	1	ReLU
MaxPooling2D	-	2	-	-
Dense	216	-	-	ReLU
Dense	1	-	-	None

$$A_t = R_t - V(s_t)$$

Este valor indica quão melhor ou pior é aquela ação comparada com o que já foi aprendido pela rede.

O treinamento é então iniciado utilizando a função de perda (*loss function*), que envolve o cálculo da razão (*ratio*):

$$\textit{ratio} = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

onde π_θ é a política atual e $\pi_{\theta_{old}}$ é a política antiga.

A função de perda é projetada para maximizar as razões que apresentam uma boa *advantage* e reduzir aquelas que pioram o desempenho da rede, sem variar muito a rede anterior, por meio da utilização da seguinte equação:

$$L^{CLIP}(\theta) = E_t\left[\min\left(\textit{ratio} \cdot A_t, \textit{clip}(\textit{ratio}, 1-\epsilon, 1+\epsilon) \cdot A_t\right)\right]$$

onde ϵ é um hiperparâmetro que controla a extensão do *clipping*, no qual foi utilizado $\epsilon = 0.2$.

Para o treino desse método, o freio foi limitado entre 0 e 0.1 e a aceleração foi limitada entre 0.5 e 1, isso foi feito com o intuito de evitar que o carro pare, além disso cada vez que foi iniciado outro set de treinos, a aceleração mínima foi reduzida com o intuito de deixar o carro aprender a reduzir a velocidade em curvas. O *input* da rede neural é o *state* sem manipulações, e o *output* é um vetor com 4 valores, em que o segundo menos o primeiro se refere à ação 1 (direção da roda), enquanto o terceiro e o quarto se referem às ações 2 (aceleração) e 3 (freio) do carro. Todos esses valores estão entre 0 e 1 por meio da função de ativação *sigmoid*. A rede neural utilizada para a *policy* foi apresentado na tabela 3 e para o *critic* foi apresentado na tabela 4.

2.3. Deep Deterministic Policy Gradient (DDPG)

O núcleo do algoritmo é formado por duas redes neurais: a rede *actor*, que produz uma ação em reação ao estado

Tabela 5. PSEUDOCÓDIGO DO MÉTODO DDPG

Iniciar aleatoriamente as redes *critic* $Q(s,a|\theta^Q)$ e *actor* $\mu(s|\theta^\mu)$ com pesos θ^Q e θ^μ
Iniciar redes *target* Q' e μ' com pesos $\theta^{Q'} \leftarrow \theta^Q$ e $\theta^{\mu'} \leftarrow \theta^\mu$
Iniciar *buffer* R
para episódio $= 1, M$ fazer
  Iniciar processo aleatório $\mathcal{N}$ para exploração
  Receber estado s_1
  para $t = 1, T$ fazer
    Selecionar ação $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$
    Executar ação a_t, receber recompensa r_t e observar estado s_{t+1}
    Guardar transição (s_t, a_t, r_t, s_{t+1}) em R
    Obter amostra de N transições (s_i, a_i, r_i, s_{i+1}) de R
    Definir $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Atualizar *critic* minimizando a perda $L = 1/N \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Atualizar a política do *actor* usando o gradiente da política amostrada:
      $\nabla_{\theta^\mu} J = 1/N \sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^Q} \mu(s|\theta^\mu)|_{s_i}$
    Atualizar as redes *target*:
      $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
      $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  finalizar *loop*
finalizar *loop*

Tabela 6. REDE NEURAL CONVOLUCIONAL PARA A ACTOR

Layer	Neurons	Kernel Size	Stride	Activation Function
Conv2D	16	5	4	ReLU
Conv2D	32	3	3	ReLU
Conv2D	32	3	3	ReLU
Dense	64	-	-	ReLU
Dense	2	-	-	Tanh

atual, e a *critic*, que produz uma avaliação da resposta produzida pela outra rede (negativa se a reação foi ruim e positiva caso contrário). Além disso, a arquitetura conta com duas redes *target*, a fim de aumentar a estabilidade do treinamento, e com um *buffer* de aprendizado, de forma a diminuir a correlação entre os estados utilizados. O pseudocódigo respectivo está descrito na tabela 5.

No que rege a entrada da rede, foram realizadas as seguintes operações: os números da imagem foram removidos; a barra de velocidade foi aumentada; as cores da grama e dos asfalto foram padronizadas; e os valores da matriz normalizados entre 0 e 1. Essas mudanças foram realizadas com base em uma implementação similar [2].

Além disso, a saída da rede *actor* foi implementada de forma a ter somente duas saídas, ao invés de todo o espaço de ações. Esse alteração foi realizada de forma a prevenir que o agente pudesse realizar as ações de frear e acelerar ao mesmo tempo, a fim de agilizar o treinamento. As recompensas utilizadas foram as recompensas padrões do módulo. No mais, as arquiteturas da rede são representadas nas tabelas 6 e 7.

3. Resultados e Discussões

3.1. Deep Q-Learning

Durante o treinamento guardou-se a recompensa total e a recompensa cumulativa em cada iteração, esse dados seguem nas figuras 1 e 2.

A seguir, realizou-se um teste do algoritmo com 30 iterações, obtendo-se o retorno total e o retorno cumulativo,

Tabela 7. REDE NEURAL CONVOLUCIONAL PARA A CRITIC

Layer	Neurons	Kernel Size	Stride	Activation Function
Conv2D	16	5	4	ReLU
Conv2D	32	3	3	ReLU
Conv2D	32	3	3	ReLU
Concatenação entre a saída da anterior com a entrada das ações				
Dense	64	-	-	ReLU
Dense	32	-	-	ReLU
Dense	1	-	-	Linear

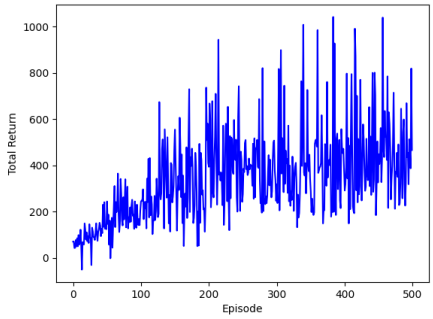

Figura 1.

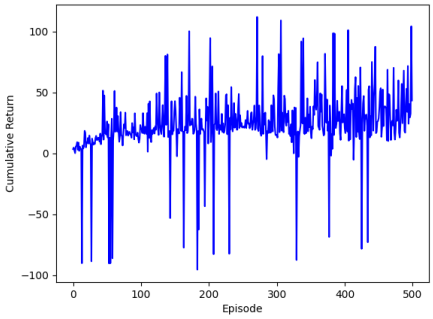

Figura 2.

Tabela 8. TESTE DO MÉTODO DEEP-Q-LEARNING

Retorno Total (média)	Retorno Cumulativo (média)
2.6e+02	5.3

porém, agora, utilizando apenas a recompensa definida pelo ambiente, os resultados obtidos seguem nas figuras 3 e 6 e na tabela 8.

Observa-se que o método *Deep-Q-Learning* apresentou uma convergência no treinamento conturbada, com pontos que saiam consideravelmente da média das iterações, o que refletiu em seu teste final, no qual apresentou-se apenas um resultado consideravelmente bom, com retorno total acima de 800, com o restante apresentando resultados médios, em torno de um retorno total de 300.

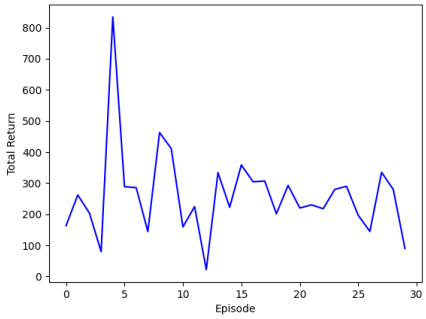

Figura 3.

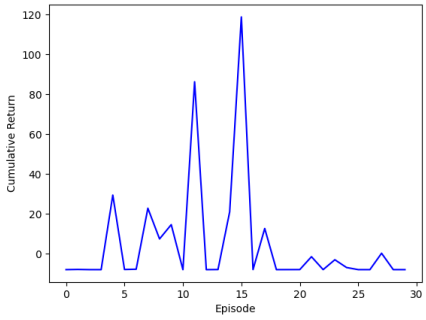

Figura 4.

3.2. Policy Optimization Algorithm

Figura 5.

Figura 6.

Na figura 5, é possível observar grandes quedas do *reward* máximo quando foi finalizado uma sessão para iniciar outra, por exemplo a partir do episódio 40 e a partir do episódio 60. Isso acontece devido ao fato de que, antes a *loss function* estava minimizando vários episódios ao mesmo tempo, levando a uma solução geral, enquanto que quando se iniciou outra sessão de treinamento, havia poucos episódios para serem minimizados, levando a decisões que auxiliem em estados específicos mas atrapalhem outros estados. Para resolver isso, a partir do episódio 80, antes de iniciar o treinamento, foi rodado alguns episódios para deixar na lista e dar um contexto geral para o algoritmo do que já foi aprendido.

Além disso, foi possível observar que antes do episódio 40, o *reward* máximo foi próximo de 830 durante o treino, porém a velocidade se manteve muito reduzida, devido ao fato da velocidade e o freio serem limitados, o que pode ser observado nos testes na figura 6, que obtiveram um *reward* um pouco abaixo devido a baixas velocidades. Para solução desse problema foi reduzido a velocidade mínima do carro a partir da iteração 40 e mais ainda a partir da iteração 60. No entanto o método de aleatoriedade utilizado para obtenção do ruído não é muito eficiente para ensinar o carro a freiar, pois o ruído utilizado freiava em um *frame* e acelerava no outro, não deixando claro para o algoritmo qual a consequência de freiar ou não. Para solucionar esse problema, a partir do episódio 80 foi adicionado ruídos constantes e prolongados por vários episódios, como freiar por alguns segundos, acelerar por alguns segundos ou virar para direita por alguns segundos, tudo isso com intenção de ensinar o algoritmo a manusear o carro em altas velocidades e aprender curvas para direita, as quais quando acentuadas, levava o carro a virar para a esquerda. Além disso também o ruído foi reduzido para 10 por cento do que foi utilizado no inicio.

É possível observar que a partir da iteração 80, na figura 5, o carro teve uma certa dificuldade em não ficar parado, marcado pelas baixas pontuações obtidas. Logo em seguida os resultados voltaram para o padrão, porém agora com a possibilidade de velocidades maiores devido a perda da limitação da desaceleração do carro.

O que aconteceu na verdade foi que mesmo após voltar para desempenho normal, o algoritmo voltou a ter dificuldades em não parar durante a pista, marcado pelo declinio dos rewards novamente, sem o início de uma nova sessão de treinamento, o que se deve ao fato de que ele não recebe um *input* claro sobre a velocidade do carro no momento nem a quanto tempo ele permaneceu naquele estado. Dessa forma, os melhores pesos para o carro foram atingidos na iteração 40, quando sua velocidade e o freio estavam limitados, tirando a possibilidade dele ficar parado.

Assim, o *PPO* atingiu um resultado satisfatório, deixando a desejar somente nas curvas para direita, as quais algumas vezes ele confundia com uma curva para esquerda, e também na utilização eficaz do freio e aceleração máximas. O primeiro foi devido ao fato de que curvas para direita só acontecem no meio ou final do percurso, e a segunda devido a desinformação quanto a velocidade do carro por parte do programa.

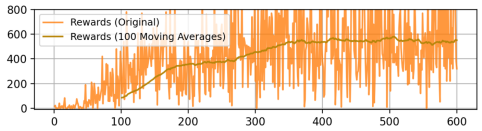

Figura 7. Recompensa por episódio e média móvel da recompensa por episódio ao longo do treinamento.

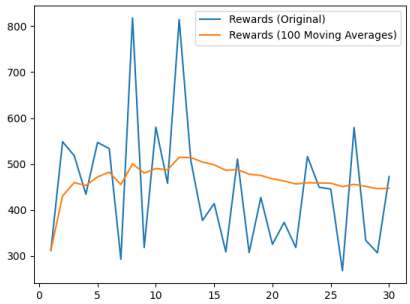

Figura 8. Recompensa por episódio e média móvel da recompensa por episódio ao longo do período de teste.

3.3. Deep Deterministic Policy Gradient (DDPG)

São mostrados na figura 7 o retorno e o retorno médio (dos últimos 100 episódios), durante um processo de treinamento de 600 episódios, o qual durou em torno de 6 horas.

Além disso, seguem também os resultados obtidos para o retorno e o retorno médio da melhor solução obtida pelo treinamento na figura 8, durante um período de teste de 30 episódios (não invocando o processo aleatório utilizado no treinamento para exploração).

Nota-se que as recompensas obtidas com o episódio, mesmo com os pesos obtidos pelo treinamento, variam bastante com a posição inicial do agente, com a média final do teste alcançando em torno de 500. Dessa forma, percebe-se que, mesmo com as implementações das redes *target*, cujo objetivo é aumentar a estabilidade do treinamento, a solução final obtida constituiu-se consideravelmente instável.

Conclusão

Conclui-se que o melhor método dos três aplicados para o ambiente do *Car Racing* foi o *Policy Optimization Algorithm*, tanto por apresentar um treinamento rápido e com consideravelmente menos iterações que os outros dois métodos apresentados, como pela sua melhor pontuação nos testes, apresentado resultados consistentemente acima de 500.

Ademais, observou-se que os métodos off-policy (Deep Q Learning e Deep Deterministic Policy Gradient) apresentaram uma dificuldade no aprendizado para completar a pista, no qual apenas apresentaram resultados semelhantes, no treinamento, ao método on-policy (Policy Optimization Algorithm) com mais de 400 iterações. Porém, quanto aplicados aos testes o método PPO apresentou uma superioridade aos métodos off-policy.

Referências

[1] Richard S. Sutton, *Temporal Credit Assignment in Reinforcement Learning* (PhD thesis). Amherst, MA: University of Massachusetts, 1984.

[2] wpiszlogin, *https://github.com/wpiszlogin/driver_critic/tree/main*. Acessado em 18 de Julho de 2024.