Aprendizado por reforço usando algoritmos genéticos na busca de políticas ótimas

Gustavo Ciotto Pinton, RA 117136*
Lucas Rath Maia, RA 117751†

Abstract

Neste documento, é proposto um modelo de aprendizado por reforço baseado em redes neurais, cuja política é otimizada a partir de algoritmos genéticos. Busca-se com estas ferramentas sintetizar o controle de um módulo lunar simulado, cujo objetivo é pousar em uma região específica entre duas bandeiras. Dois métodos foram testados: um em que as características são parâmetros fornecidos pelo próprio simulador e outro, em que tais características são obtidas a partir de fotografias do estado atual da simulação, isto é, capturas da própria tela. Para este último, gera-se um conjunto de imagens aleatórias para treinamento.

1. Introdução

Algoritmos de aprendizado por reforço vêm ganhando força nos últimos anos pelo crescente interesse econômico que tais técnicas podem proporcionar. Um exemplo evidente dessa afirmação está no fato de que grandes empresas, como *Uber* e *Google*, estão apostando em carros capazes de dirigir e tomar decisões por si próprios sem a necessidade de um motorista humano. Apoiando-se nesse grande potencial, propõe-se neste documento o controle de um módulo lunar simulado a partir de algoritmos genéticos e algoritmos de aprendizado por reforço. Conforme detalhado posteriormente, isto pode ser realizado por dois caminhos distintos: um em que o estado é obtido pelo próprio simulador e o outro em que ele é a saída de uma rede neural convolucional, cuja entrada é uma imagem representando a situação do simulador.

Nas próximas subseções, os principais métodos e técnicas utilizados neste projeto são introduzidos ao leitor.

1.1. Reinforcement Learning

Aprendizado por reforço é um tipo importante de aprendizado de máquina em que um agente aprende a se comportar em um ambiente realizando ações e vendo os resultados.

*Is with the Institute of Computing, University of Campinas (Unicamp). **Contact**: gustavociotto@gmail.com

†Is with the Institute of Computing, University of Campinas (Unicamp). **Contact**: lucasrm25@gmail.com

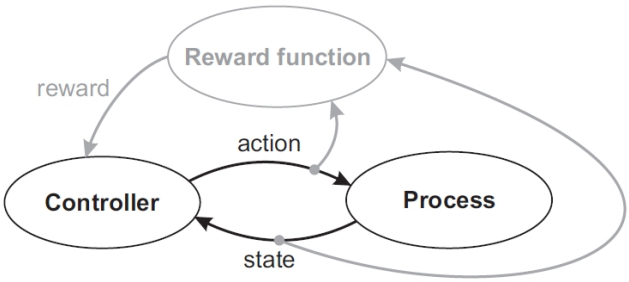

Figura 1: Esquema básico dos componentes de um algoritmo de aprendizado por reforço. Retirado de [1].

Em algoritmos de aprendizado por reforço, um controlador interage com um ambiente a partir de três sinais: estado que descreve o estado do ambiente (processo), ação que determina como o controlador modifica o ambiente e finalmente a recompensa, que permite avaliar sua performance. A cada passo, o controlador recebe uma medida do estado e, em seguida, aplica uma ação, modificando o ambiente. A partir da recompensa é possível avaliar a efetividade desta modificação. A figura 1 resume estes três sinais. O comportamento do controlador é definido pela sua **política**, isto é, por uma função $\pi : A \to S$ que mapeia **estados** S em **ações** A. Há dois tipos de política diferentes, isto é, determinística ou estocástica. A determinística descreve caso em que, para um estado S, uma única ação A será atribuída, conforme equação 1.

$$A = \pi(S) \tag{1}$$

Para a políticas estocásticas, por outro lado, $\pi(A, S)$ resulta na probalidade de A ser escolhida dado um estado S, conforme equação 2.

$$\pi(A, S) = \mathbb{P}(A|S) \tag{2}$$

O comportamento do ambiente, por sua vez, é descrito por sua dinâmica, que determina como o estado muda em resultado às ações tomadas pelo controlador. Tal comportamento aliado à função de recompensa e aos conjuntos de estados e ações possíveis constitui um processo de decisão de Markov [1].

Nos algoritmos de aprendizado por reforço, o objetivo é encontrar uma política ótima que maximiza o **retorno esperado**, isto é, a recompensa acumulada esperada durante todos as iterações. A recompensa acumulada esperada no passo t, G_t é escrita como uma soma de todas as recompensas R_t sucessivas a ele multiplicadas a uma constante $\gamma \in [0, 1]$, conforme equação 3. γ nos permite priorizar recompensas de ações tomadas mais cedo ($\gamma^k \approx 0$ para k altos) ou mais tarde ($\gamma^k \approx 1$ para k altos). Para o nosso caso, em que decisões tomadas na proximidade da aterrisagem devem ser consideradas, escolhemos γ próximo de 1.

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{3}$$

Definindo uma tarefa como uma instância de um problema de aprendizado por reforço, podemos classificá-lo em duas categorias:

- *Episodic*: uma tarefa que tem começo e fim, isto é, que possui estados inicial e terminal. Uma execução gera um **episódio**, constituído por uma lista de estados, ações e recompensas.
- *Contínuo*: tarefa que executa sem fim e, portanto, não possui um estado terminal.

Para o nosso problema, como há estados finais (pouso no alvo ou colisão), cada tarefa é do tipo **episodic**.

Um algoritmo de aprendizado por reforço aprende por dois métodos distintos:

- *Método de Monte Carlo*: quando um episódio acaba, o controlador avalia as recompensas totais acumuladas e calcula a recompensa futura esperada máxima $V^\pi(E)$, definida pela fórmula abaixo.

  $$V^\pi(E) = \mathbb{E}(G_t | E_t = E) \tag{4}$$

  Em uma nova execução, $V^\pi(E)$ é modificado conforme equação 5 abaixo, em que α é a taxa de aprendizado e controla a velocidade de convergência de $V^\pi(E)$.

  $$V^\pi_{(i+1)}(E_t) = V^\pi_{(i)}(E_t) + \alpha \left[G_t - V^\pi_{(i)}(E_t) \right] \tag{5}$$

- *Temporal Difference Learning*: este método não espera até o fim de cada episódio para atualizar $V^\pi(E)$, ele o atualizará a cada iteração de acordo com a equação 6:

  $$V^\pi_{(i+1)}(E_t) = V^\pi_{(i)}(E_t) + \ldots \\ \alpha \left[R_t + \gamma V^\pi_{(i)}(E_{t+1}) - G_t - V^\pi_{(i)}(E_t) \right] \tag{6}$$

No nosso caso, escolhemos o método de aprendizado de Monte Carlo.

Finalmente, tendo definido os elementos principais de aprendizado por reforço, é necessário definir qual métrica queremos otimizar. Há três delas, conforme lista abaixo:

- *Value Based Reinforcement Learning*: deseja-se otimizar $V^\pi(S)$, isto é, deseja maximizar a recompensa futura esperada.
- *Policy Based Reinforcement Learning*: deseja-se otimizar a política, isto é, a função π, de forma a maximizar as recompensas. Para tal, utilizamos um algoritmo genético em que cada indivíduo simboliza uma possível política candidata. A subseção 1.3 possui mais detalhes sobre nossas escolhas de implementação do algortimo genético.
- *Model Based Reinforcement Learning*: nesta abrodagem, cria-se um modelo no ambiente. Tal requisição é uma grande desvantagem, visto que para cada problema é necessário um modelo distinto.

Em suma, para o nosso problema iremos utilizar a abordagem de **policy based reinforcement learning** em um ambiente **episodic** e **determinístico** a partir da técnica de **Monte Carlo**.

1.2. Lunar Lander V2

Para simular o comportamento de um módulo lunar e toda a física e equações de movimento envolvidas neste problema, utilizamos ferramentas disponíveis no *website* da **OpenAI** [2]. **OpenAI** é uma organização sem fins lucrativos de pesquisa em inteligência articial que atua em diversos projetos nesta área. Dentre estes projetos, destaca-se o **Gym**, que trata-se de um *toolkit* para desenvolver e comparar algoritmos de aprendizado reforçado a partir de diversos simuladores. A figura 2 mostra uma ilustração do ambiente *LunarLander-V2* que trata a simulação do módulo lunar que usamos neste trabalho. Neste problema a nave espacial começa em um estado aleatório e tem por objetivo pousar na lua, entre as bandeiras, de forma mais suavemente e usando menos combustível possível.

Para o simulador da figura 2, recuperamos o estado atual da nave através de dois métodos distintos. Para o primeiro, utilizamos as próprias funções disponibilizadas pela biblioteca do simulador. Neste caso, tal estado é constituído por um vetor de oito componentes: posições e velocidades nos eixos x e y, ângulo de inclinação, velocidade angular e duas variáveis *booleanas* indicando se seus apoios estavam em contato com a lua ou não. Para o segundo método, por sua vez, obtemos capturas de tela em momentos aleatórios do jogo para servir de conjunto de treinamento de uma rede neural convolucional, cuja saída é um vetor contendo as

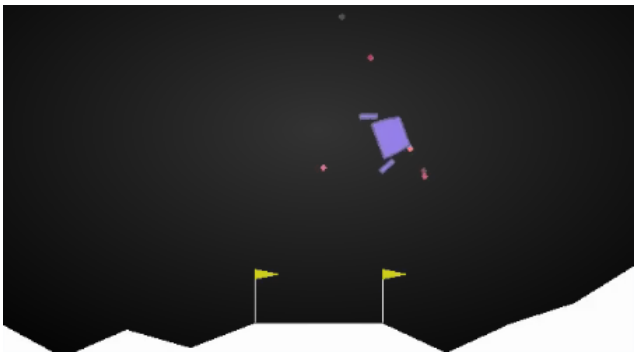

Figura 2: Simulador de módulo lunar.

posições x e y e ângulo de inclinação. A arquitetura de tal rede será melhor discutida e explicada na subseção 1.4.

As ações aplicáveis sobre o módulo lunar controlam quais dos três motores atuarão, sendo de duas naturezas distintas: discreta e contínua. Para a discreta, têm-se quatro ações distintas: *fazer nada*, *acionar motor da esquerda*, *acionar motor principal* e *acionar motor da direita*. Para a segunda, uma ação é composta por duas variáveis contínuas i e j, seguindo as regras abaixo:

- i controla o motor principal: se $i = -1$, o motor é desativado. Caso contrário, se $i \in [0, 1]$, acelera-se o módulo entre $[50\%, 100\%]$ de maneira linear.
- j controla os motores auxiliares: se $j \in [-1, -0.5]$, aciona-se motor da esquerda e, caso contrário, se $j \in [0.5, 1.0]$, ativa-se o motor da direita. Finalmente, se nenhuma das condições anteriores for atingida e se $j \in [-0.5, 0.5]$, desativa-se ambos os motores.

Para este problema, por questão de simplicidade, os autores escolheram as ações **discretas**.

Por último, mas não menos importante, encontra-se a função de avaliação do estado da nave. Tal função é chave principal do sucesso dos algoritmos genéticos, tendo em vista que ela consiste da métrica com a qual os algoritmos selecionarão os melhores indivíduos. Para o seu cálculo, soma-se os seguintes critérios:

- Distância com o ponto $(0, 0)$, isto é, o centro entre as duas bandeiras: multiplica-se a distância a tal ponto por -100.
- Velocidade: penaliza-se estados em a nave esteja muito rápida, priorizando sua estabilidade. Multiplicamos por -100 a norma do vetor velocidade.
- Inclinação: multiplica-se por -100 o ângulo de inclinação da nave.
- Contato com a superfície: cada contato vale +10.
- Gasto de combustível: a cada acionamento de qualquer um dos motores, um custo de -0.3 é considerado.

1.3. Policy search e estratégia evolutiva

Alguns trabalhos [3] propuseram o uso de estratégias evolutivas na busca de políticas ótimas para problemas de aprendizado por reforço. Em poucas palavras, um algoritmo evolutivo seleciona alguns indivíduos para servirem de pais para a próxima geração. Tal seleção é realizada a partir de uma função f, denominada **função objetiva**. Para este trabalho, escolhemos o algoritmo evolutivo **CMA-ES** (do inglês, *Covariance Matrix Adaptation Evolution Strategy*). O CMA-ES é um método estocástico para otimização de parâmetros reais (domínio contínuo) de funções não-lineares e não convexas [4]. São justamente estas últimas características que nos habilitam a utilizá-lo no nosso problema, tendo em vista que adotaremos uma política estocástica. Além disso, [3] argumenta que esta técnica é conhecida por dar bons resultados para espaços de solução de até alguns milhares de parâmetros.

Modelamos nosso problema de maneira que cada indivíduo representa uma possível política candidata. As politícas são representadas por uma função linear conforme equação 7, em que $\boldsymbol{W}$ é uma matriz de pesos e $\boldsymbol{b}$ é um vetor de *bias*. Tal escolha foi feita baseada em [3], onde o mesmo princípio foi utilizado para o controle do simulador do jogo de corrida *CarRacing-V0*.

$$A = \pi(S) = \boldsymbol{W}S + \boldsymbol{b} \tag{7}$$

A função objetiva ao final de cada geração é uma média das recompensas acumuladas esperadas obtidas nas $N = 5$ execuções que cada indíviduo realiza, conforme equações 3 e 8. Por fim, a população comporta 50 indivíduos no total.

$$f = \frac{1}{N}\sum_{i=1}^{N} G_t = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{8}$$

Conforme citado anteriormente, propomos dois métodos para a obtenção dos atributos a serem transmitidos aos indivíduos (soluções candidatas). Para o primeiro método, tais atributos são obtidos diretamente do simulador. Para o segundo, em que apenas *screenshots* do simulador estão disponíveis, utilizamos redes neurais convolutivas. Nessa abordagem, testamos outras duas ferramentas:

- *Autoencoder*: treina-se a rede para que a saída seja igual a entrada. A pecularidade deste tipo de rede é que há um afunilamento para uma camada escondida intermediária com poucos neurônios, seguida por uma posterior expansão. Após treinamento e validação, a saída desta camada escondida intermediária torna-se os atributos a serem utilizados pelos indivíduos no algortimo evolutivo.
- A segunda rede convolucional proposta recebe as imagens na entrada e retorna as coordenadas da nave, tais como posições x e y, inclicação e se os suportes tocaram o chão, conforme figura 3. Detalhes da arquitetura encontram-se na próxima subseção.

Nossos testes provaram que a **segunda** abordagem produziu melhores frutos.

1.4. Redes Convolucionais

O segundo método de obtenção das coordenadas do módulo lunar sugere a utilização de redes convolucionais, de acordo com esquema da figura 3.

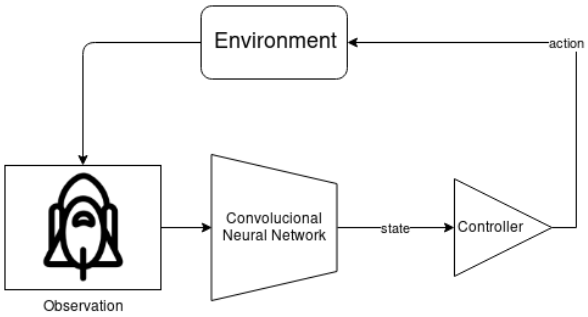

Figura 3: Atributos retirados de *screenshots* do simulador.

Conforme visto durante as aulas, uma rede convolucional recebe esse nome justamente devido à presença de várias camadas constituídas por filtros aplicados aos resultados da camada anterior por meio da operação de **convolução**. A grande vantagem destas redes sobre as convencionais é que não é necessário o pré-processamento de atributos, permitindo, deste modo, que as entradas da rede sejam as próprias imagens. Isso decorre do fato de que os filtros são capazes, após a etapa de treinamento, de retirar os atributos pertinentes para classificação, podendo-se dizer, portanto, que a própria rede aprende por si só os atributos que serão transportados para as camadas finais. Para a nossa aplicação, em que deseja-se obter as posições da nave a partir das imagens, essa característica é fundamental. Para o treinamento, geramos aleatoriamente *batches* de 60 imagens com a nave em diversas situações distintas. O objetivo é cobrirmos o maior número de situações possíveis, a fim de que a rede possa aprender suficientemente bem todas elas.

Primeiramente, definimos a estrutura da rede convolucional a ser usada. De fato, este problema de detecção de imagens já é bastante conhecido na literatura e portanto já existem redes neurais com estruturas otimizadas para este fim. Deste modo, escolhemos por utilizar como base para a extração dos features a rede *TinyYoloV2 - TinyDarknet19*, cuja estrutura pode ser vista na figura 4. Conforme figura, camadas convolucionais para recuperação dos *feature* são intercaladas com camadas de *max pooling* para redução da dimensionalidade.

Por meio da técnica chamada **transfer learning**, acrescentamos ao final da rede camadas densas a fim de obter o número de saídas desejadas para o conjunto de treinamento e por fim capturar a posição e angulo do Lunar Lander.

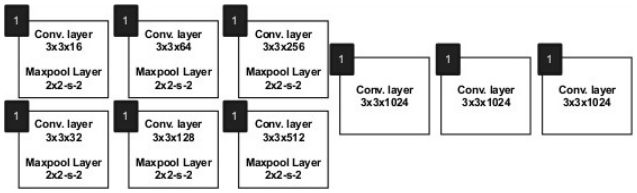

Figura 4: Arquitetura da rede neural convolucional Tiny-Yolo-V2

Tabela 1: Valores para os melhores indivíduos de algumas gerações.

Geração	Fitness	Geração	Fitness
1º	-105.5	195º	228.56
5º	-27.14	200º	248.86

Além disso, para acelerar o processo de treinamento, inicializamos a camada Yolo-V2 com pesos já pré-treinados de um dataset chamado *COCO*.

2. Resultados

As subseções abaixo discutem os resultados encontrados.

2.1. Experimento 1

A figura 6 possui três curvas: uma para a recompensa máxima obtida, outra para a mínima e a terceira para média. Conforme explicado nas seções anteriores, no total, utilizamos populações de 50 indivíduos em 200 gerações. Observa-se uma grande evolução no controle da nave, sendo que o melhor indivíduo da primeira geração recebeu recompensa bem negativa (≈ -200), enquanto que o último, uma recompensa próxima de 240.

A tabela 1, finalmente, representa os valores do melhores indivíduos de alguns gerações.

2.2. Experimento 2

O treinamento da rede da seção 1.4 utilizou 100.000 imagens comprimidas de 400x600x3 para 200x200x3 para acelerar o treinamento e aproximadamente 11000 iterações. O MSE (*mean squared error*) entre a saída da rede e os dados obtidos diretamente através do simulador variou conforme figura 5. A presença de vários picos nesta imagem revela as nossas escolhas para o *learning rate* α: quando a métrica MSE encontrava-se constante, isto é, estagnada em um nível C constante, o seu valor era aumentado para $\alpha \approx 10^{-4}$, com o intuito de sair de eventuais mínimos locais. Além disso, durante os treinamentos, α era alterado pelo otimizador ADAM [5], disponível na biblioteca do *TensorFlow*. A figura 7 mostra o resultado do treinamento da rede: o quadrado amarelo é uma representação visual das posições x, y e ângulo obtidos pela rede.

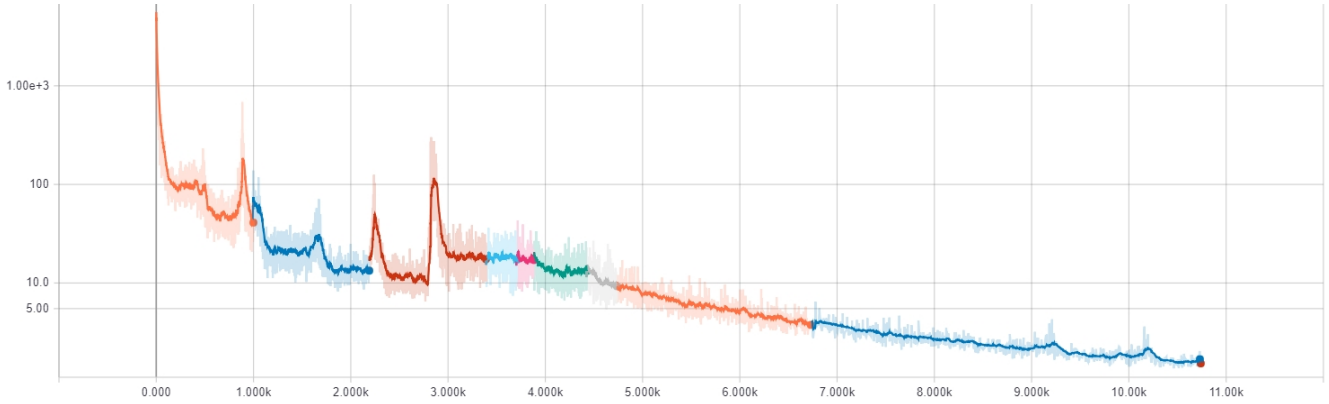

Figura 5: Função de custo do erro quadratico médio da rede convolucional descrita na seção 1.4.

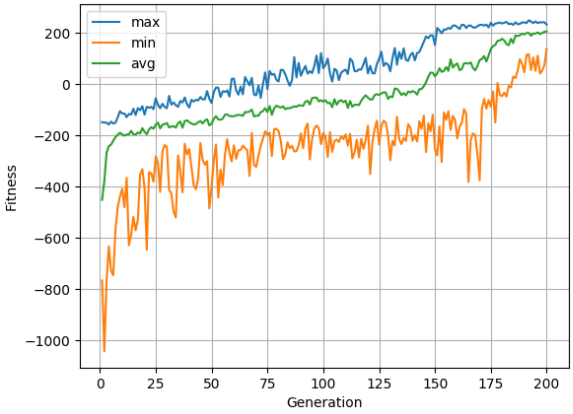

Figura 6: Curva de aprendizado do algoritmo evolutivo para o método 1.

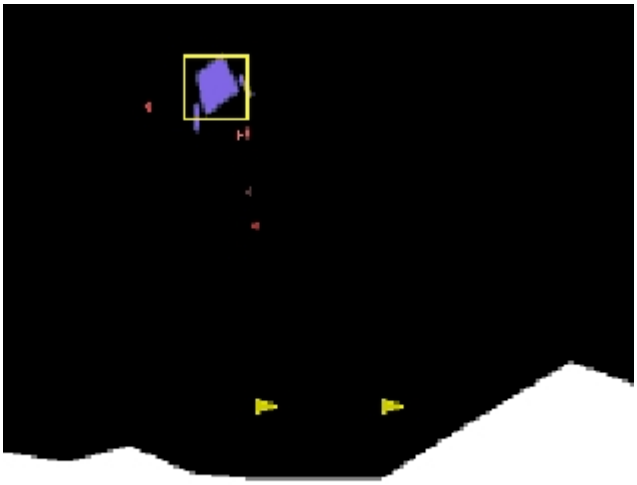

Figura 7: Comparação da saída da rede convolucional com a posição real da nave.

Por limites computacionais, não pudemos executar o algoritmo genético utilizando a rede convolucional proposta para obtenção das coordenadas da nave. No lugar, utilizamos as melhores políticas encontradas na execução do experimento 1.

3. Conclusões

Dois métodos foram propostos para a obtenção de características do simulador. O primeiro utiliza funções próprias do simulador, enquanto que o segundo retira as posições horizontal e vertical a partir de *snapshots* da simulação. Para a otimização da política empregada pelo aprendizado por reforço, escolhemos um algoritmo evolutimo, o CMAES. Os resultados para o primeiro caso foram muito satifatórios, em oposição ao segundo, em que, devido a limites computacionais, não fomos capazes de executar a procura por uma política ótima conforme planejado.

Referências

[1] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010. 1

[2] About OpenAI. https://openai.com/. [Online; acessado 24 de Junho de 2018]. 2

[3] D. Ha and J. Schmidhuber. World models. 2018. 3

[4] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016. 3

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 4