Resolução do ambiente OpenAI Cart Pole com base em Algoritmos Genéticos

Universidade Tecnológica Federal do Paraná –
Campus Medianeira
Rafael Augusto de Oliveira

Resumo

Neste relatório descrevo a resolução do ambiente Cart Pole do OpenAI Gym, reduzindo-o a um problema de otimização e utilizando Algoritmos Genéticos para maximizar a pontuação obtida. Também descrevo o ambiente e o agente conforme as classificações de Russell e Norvig em "Artificial Intelligence: A Modern Approach".

Keywords: Algoritmos Genéticos, problemas de otimização, OpenAI.

I. Introdução

OpenAI Gym [8] é conjunto de ambientes open-source para aplicação de algoritmos de aprendizado por reforço, que oferece uma interface genérica para que o agente possa realizar tipos diferentes de aprendizado sem mesmo conhecer o ambiente proposto. Neste caso, foi utilizado para a criação de um agente sem aprendizado de máquina (uma IA Fraca, portanto).

O método adotado foi reduzir o jogo a um problema de otimização, para poder abusar da técnica de Algoritmo Genético, procurando o ótimo global do ambiente – como Cart Pole é um problema infinito, foi utilizado a marca de 200 pontos.

O relatório está organizado da seguinte maneira: Na Seção II descrevo o ambiente, e sua classificação segundo Russell e Norvig [1]. Seção III é responsável por introduzir a técnica de Algoritmo Genético, explicando cada uma de suas funções. Na Seção IV

apresento a ideia proposta para resolver o jogo, bem como a classificação do agente segundo Russell e Norvig. As conclusões ficam para a Seção V.

II. O Ambiente

a. Descrição do ambiente

O ambiente escolhido foi o CartPole-v0, onde um mastro se encontra anexado a um carrinho que se move, sem atrito, em uma pista. Esse mastro se encontra em pé, e o objetivo é mover o carrinho, para a direita ou esquerda, de modo que o mesmo não caia.

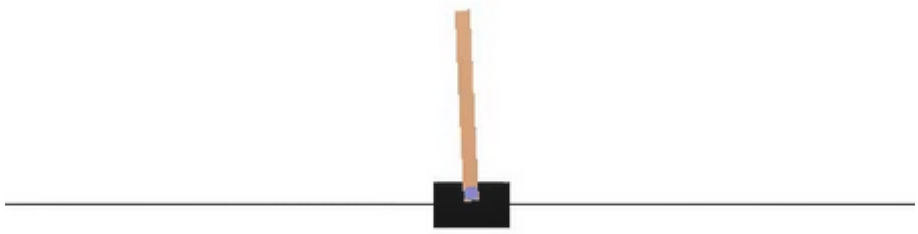

Fig. 1: Ilustração do ambiente CartPole-v0 [2]

O agente controla o carrinho aplicando uma força de ± 1, e uma recompensa de $+1$ é retornada a cada vez que o mesmo mantiver o mastro em pé. O jogo é considerado resolvido se a média das recompensas for maior ou igual a 195.0 após 100 tentativas consecutivas, ou termina se o mastro estiver a $\pm 12°$ da vertical; a posição do carrinho seja maior que ± 2.4; ou a duração do episódio seja maior que 200.

b. Classificação segundo Russell e Norvig[1]

Segundo a classificação de ambientes descrita por Russell e Norvig[1], o ambiente escolhido detem as seguintes características:

- Parcialmente Observável: Considerado dessa forma pois, mesmo que o agente tenha observação total sobre o carrinho e o mastro, o mesmo não possui ideia sobre a angulação máxima que o mastro pode ter, ou a posição máxima que o carrinho pode estar;
- Agente Simples: O Algoritmo Genético possuí uma população de possíveis soluções para o problema, mas apenas um indivíduo por vez é levado em conta, por isso o agente é considerado simples;
- Determinístico: O próximo estado do ambiente pode ser completamente determinado pelo estado atual e a ação a ser executada pelo agente;
- Episódico: O agente não precisa guardar informações sobre as ações passadas para performar a ação futura, ele simplesmente recebe uma percepção e realiza uma ação.
- Dinâmico: Por mais que o ambiente ao redor do carrinho não mude, o fato do agente não performar uma ação faz com que o mastro mova-se passando da angulação limite, fazendo com que ele perca o jogo;
- Contínuo: Mesmo que as ações do agente sejam discretas, baseado nas percepções como angulação do mastro e velocidade do carrinho, o ambiente é de estado contínuo e tempo contínuo.

III. Algoritmo Genético

a. Ideia geral

Subcampo da Computação Evolutiva, os Algoritmos Genéticos [3], [4], [5] são técnicas de pesquisa e otimização para a resolução de problemas complexos baseadas na evolução biológica, em que uma população de indivíduos – cada um sendo uma possível solução para o problema proposto – capazes de realizar reprodução sexuada (Crossover) e sofrer processos de mutação genética, são selecionados baseados nos seus genes para realizar reprodução, resultando em novas populações de indivíduos que são mais adaptados àquele ambiente.

Essa abstração trazida para o campo computacional gera uma técnica heurística, porém direcionada, para a procura do ótimo global da solução.

Tipicamente, o algoritmo possui uma representação genética – array geralmente binário-, e segue como descrito abaixo.

```
Algoritmo Genético Genérico
Fazendo T = 0 ser o contador de gerações;
Cria e inicializa uma população de n-bit cromossomos;
Enquanto Condicao_Parada não for Verdadeiro, Faca:
    Calcula a Fitness de F(x) para cada indivíduo x na população;
    Seleciona os indivíduos para a reprodução;
    Realiza Crossover;
    Gera Mutação nos descendentes;
    Troca a população atual pela nova;
    Incrementa contador de gerações, T = T + 1;
FimEnquanto;
```

Fig 2: Estrutura simples de um Algoritmo Genético

Segundo o modelo SGA (Simple Genetic Algorithm) proposto por Holland [6], [7], temos quatro funções que se repetem até que a condição de parada seja atingida, sendo elas: Função Fitness, Seleção, Crossover e Mutação.

b. Função Fitness

Responsável por avaliar quão bem o indivíduo se encaixa em determinado ambiente, atribuíndo-lhe uma pontuação (Fitness) que deve ser otimizada.

c. Seleção

Seleção escolhe quais os indivíduos dentro da população que serão submetidos ao Crossover para gerar descendentes, baseado na Fitness de cada um. Soluções mais adaptadas ao ambiente (isto é, com uma Fitness maior) tendem a produzir mais descendentes, bem como as menos adaptadas tendem a desaparecer.

O SGA utiliza o método de Roulette Wheel (Roleta), onde cada indivíduo é alocado em um setor (slot) da roleta baseado em sua *Fitness / Fitness-Média*, e então um numero aleatório nesse espectro da roleta é gerado, selecionando o indivíduo para o Crossover.

Outra possibilidade de Seleção é o Elitismo, onde o algoritmo garante que uma porcentagem dos indivíduos, sendo eles os melhores da população, sempre seja selecionada para prosseguir para a próxima geração.

d. Crossover

Após a Seleção, ocorre o proceço de Crossover (Recombinação Genética) com os indivíduos selecionados, aos pares. O SGA utiliza o método mais simples de Crossover, o Single-Point. Assumindo uma string binária de tamanho L, é então selecionado um valor aleatório entre 1 e $L - 1$, e a porção além do ponto selecionado é trocada entre os indivíduos, de forma que dois novos indivíduos sejam gerados.

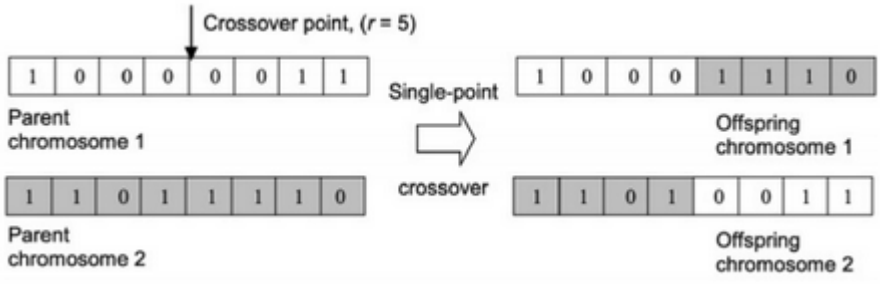

Fig 3: Ilustração do Single-Point Crossover em um par de strings [5]

O Crossover só ocorrerá se, após a Seleção, um numero aleatório entre 0 e 1 for maior que Pc (Crossover Rate – que varia entre 0 e 1).

Outras estratégias de Crossover também muito utilizadas são, Two-Point Crossover (ou de maneira generalizada k-Point), que funciona de forma similar ao Single-Point mas com a adição de mais pontos para a troca de segmentos, e o Uniform Crossover, que troca bit-por-bit no array à uma taxa fixa – geralmente de 0.5 .

e. Mutação

Os novos indivíduos gerados no Crossover são posteriormente sujeitos a Mutação, que altera o array em cada posição, mudando o bit entre 0 e 1, a uma probabilidade Pm (Mutation Probability), geralmente de 0.01 .

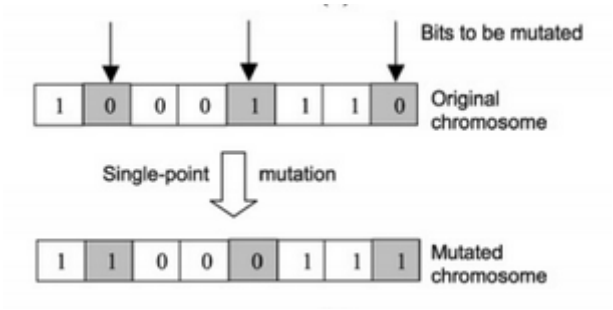

Fig 4: Ilustração de Mutação sendo realizada em 3 pontos [5]

Esse operador é o principal responsável por atribuir variação a espécie, garantindo que o algoritmo não fique preso em um ótimo local.

IV. O Agente

a. Trabalhando o jogo como um problema de otimização

Inicialmente o jogo foi transformado em um problema de otimização, sendo que a Função Fitness deveria atingir 200 pontos.

```
import gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
```

Fig 5: Exemplo de código-teste para o CartPole-v0 [6]

Baseando no código acima, *observation* recebe as primeiras observações do *env.reset()*, sendo esse um array com as quatro seguintes posições:

Observation

Type: Box(4)

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

Tab 1: Observações do ambiente [2]

Como as observações estão armazenadas em um objeto do tipo List, fez-se necessário organizar cada um dos elementos do array dos indivíduos como uma combinação linear de pesos para cada uma das observações [7], com um valor entre [-1, 1], para posteriormente realizar uma multiplicação de matrizes com o array de observações e,

por fim, o agente decidir se ele movimentará para a esquerda ou para a direita.

Actions

Type: Discrete(2)

Num	Action
0	Push cart to the left
1	Push cart to the right

Tab 2: Possíveis ações do agente [2]

```
parameters = np.random.rand(4) * 2 - 1
```

```
action = 0 if np.matmul(parameters,observation) < 0 else 1
```

Fig 6: Inicialização dos pesos em *parameters* e cálculo da *action* do agente [7]

Após a chamada de *env.step(action)*, o ambiente retorna quatro variáveis: As novas *observations* do jogo, *reward* com uma pontuação para o agente (0 ou 1), *done* uma variável *bool* que recebe *True* quando o jogo se encerra e *info* com informações para debug.

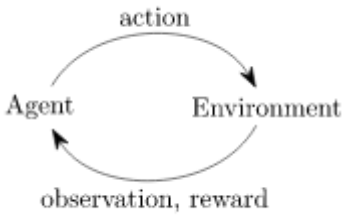

Fig 6: Ciclo do ambiente OpenAI [6]

Com base na pontuação de cada indivíduo, é possível calcular a Fitness Function do mesmo, para então realizar Crossover e Mutação,

gerando novas populações até que a máximização do problema, isto é, a pontuação de 200 seja atingida.

Para esta resolução foi utilizado uma população de 100 indivíduos, e as seguintes estratégias:

- Seleção: Seleção simples com probabilidade baseada na Fitness do indivíduo (*Fitness Individual / Fitness Total*) e Elitismo aplicado em 10% da população;
- Crossover: Crossover Uniforme com a probabilidade Pc calculado de fórma adaptativa, de forma a fazer com que o algoritmo tenha um bom balanço entre a capacidade de convergir para um ponto ótimo (local ou global), mas também tenha váriabilidade para buscar o ponto ótimo global [3];
- Mutação: Mutação Single-Point com uma probabilidade Pm fixa em 0.5, de forma a ser bem disruptivo com indivíduos menos adaptados [3].

Com essas configurações, foi possível atigir uma média de 45.86 gerações em 50 rodadas.

b. Classificação segundo Russell e Norvig [2]

Como resultado da mudança para um problema de otimização, temos um Agente Reativo Simples, onde ele unicamente recebe as percepções atuais do ambiente e executa ações de acordo, sendo essas inicializadas de maneira aleatória dentre os indivíduos da população e, posteriormente, otimizadas.

Seu PEAS, portanto:

- Performance: Manter o mastro a uma angulação máxima de $\pm 12°$;
- Environment: Uma pista horizontal, de extremos ± 2.4;
- Actuators: Movimentar o carrinho para a esquerda ou para a direita [0, 1];
- Sensors: Posição do carrinho, velocidade do carrinho, ângulo do mastro e velocidade na ponta do mastro.

V. Conclusão:

Reduzir jogos à problemas de otimização é uma alternativa bastante viável para ambientes não tão complexos, em que o ambiente é episódico e/ou as observações e ações não são tamanhas a ponto de ser necessário aprendizado de máquina. Algoritmos Genéticos, neste ponto, tem melhor performance que outros, como Hill Climbing, por possuírem "heurística direcionada", fazendo com que o algoritmo não fique preso em ótimos locais.

Neste relatório, apresentei a técnica e a lógica utilizadas para adaptar e resolver o problema, bem como uma completa descrição do agente e do ambiente que ele está inserido.

Algoritmos Genéticos me impressionam pelas suas características teóricas e pela possibilidade de generalização para os mais diversos problemas de diferentes campos do conhecimento, e pretendo estudar mais seus usos, principalmente aliados com aprendizado de máquina.

VI. Referências:

[1] Stuart J. Russell and Peter Norvig. "*Artificial Intelligence: A Modern Approach"* (3 ed.). Pearson Education, 2003.

[2] *"OpenAI Wiki: CartPole-v0"*. OpenAI. https://github.com/openai/gym/wiki/CartPole-v0.

[3] M. Srinivas and L. M. Patnaik, *"Adaptive probabilities of crossover and mutation in genetic algorithms"*. Systems, Man and Cybernetics, IEEE Transactions on, Vol. 24, No. 4, pp. 656-667, 1994.

[4] M. Srinivas and Lalit M. Patnaik. 1994. *"Genetic algorithms: A Survey"*. *Computer* 27, 6 (June 1994), 17-26. DOI: https://doi.org/10.1109/2.294849

[5] De Castro, Leandro. (2007). *"Fundamentals of Natural Computing: An Overview"*. Physics of Life Reviews. 4. 1-36. 10.1016/j.plrev.2006.10.002.

[6] *"OpenAI Gym Docs"*. OpenAI Gym. https://gym.openai.com/docs/.

[7] Frans, Kevin. *"Simple reinforcement learning methods to learn CartPole"*. July 01, 2016. http://kvfrans.com/simple-algoritms-for-solving-cartpole/.

[8] Brockman, Greg; Cheung, Vicki; Pettersson, Ludwig; Schneider, Jonas; Schulman, John; Tang, Jie; Zaremba, Wojciech. "OpenAI Gym". http://arxiv.org/abs/1606.01540. 2016.

[9] Andries P. Engelbrecht. 2007. *"Computational Intelligence: An Introduction"* (2nd ed.). Wiley Publishing.

[10] Alzantot, Moustafa. *"Episode 1 — Genetic Algorithm for Reinforcement Learning".* Becoming Human, June 07, 2017. https://becominghuman.ai/genetic-algorithm-for-reinforcement-learning-a38a5612c4dc.

[11] Marrali, Michele. Template. Studio Storti Srl. Under CC0 1.0 Universal (CC0 1.0) Public Domain Dedication.