Problema do pêndulo invertido utilizando Aprendizagem por Reforço

João Marcos Campagnolo, Renan de Matos Casaca

[1]Universidade Federal da Fronteira Sul (UFFS)

jota.campagnolo@gmail.com, recasaca@gmail.com

Resumo. *Este artigo apresenta conceitos básicos de Inteligência Artificial e Aprendizado por Reforço, além de descrever a implmentação de um algortimo que soluciona o problema do Cart-Pole disponibilizado pela biblioteca OpenIA Gym feita em python. Através dos resultados obtidos, discursamos sobre as escolhas inerentes ao problema, como valores de recompensa, número máximo de episódios e curva de aprendizagem.*

1. Inteligência Artificial

A inteligência artificial é um ramo da ciência da computação que se refere a capacidade das máquinas em aprender, seja através de software ou outros mecanismos. Nesse contexo, aprender significa que ela seja capaz de raciocinar, perceber e decidir de forma racional e inteligente.

Existem várias outras definições de inteligência artificial, como a encontrada em [Luger 2008], que se levanta questões para perguntas como: ”O que é inteligência artificial?”especificando como ramo da informática que se preocupa com a automação do comportamento inteligente.

A inteligência artificial utiliza métodos baseados em comportamentos para solucionar problemas complexos. E segundo [Coppin 2004], o objetivo do estudo da Inteligência Artificial busca algoritmos, heurísticas e metodologias baseadas no cérebro humano. E assim, vem produzindo sistemas computacionais cada vez mais úteis, utilizando métodos de inteligência artificial, sendo estes métodos de busca, aprendizado de máquina, redes neurais, entre outros.

2. Aprendizagem por Reforço

O Aprendizado de Máquina resumidamente consiste em algoritmos que ficam ”aprendendo”conforme os dados de entrada, ficam treinando, aperfeiçoando os resultados e buscando as melhores saídas. Segundo [Coppin 2004], na maioria dos problemas de aprendizado, a tarefa é aprender a classificar os dados de entrada, pois são esses dados que ficarão em treinamento. O algoritmo então, tenta aprender, a partir desses dados de treinamento, como classifica-los e como classificar novos dados que não foram ainda treinados.

Um algoritmo de Aprendizagem por Reforço, por sua vez, consiste em receber um reforço de acordo com a aprendizagem. Ou seja, o algoritmo recebe reforço positivo por operar os dados trienados corretamente ou então um reforço negativo por operar incorretamente. Ainda utilizando um exemplo de [Coppin 2004], para facilitar o entendimento, se um robô utilizando aprendizagem por reforço for pegar um objeto, quando pegar o objeto corretamente receberá um reforço positivo.

3. O problema do pêndulo invertido

Recebemos como proposta desenvolver um controlador para o problema do pêndulo invertido utilizando aprendizagem por reforço. O problema do pêndulo invertido consiste basicamente de um carro que está ligado a um pino através de um eixo, formando um espécie de pêndulo de cabeça para baixo.

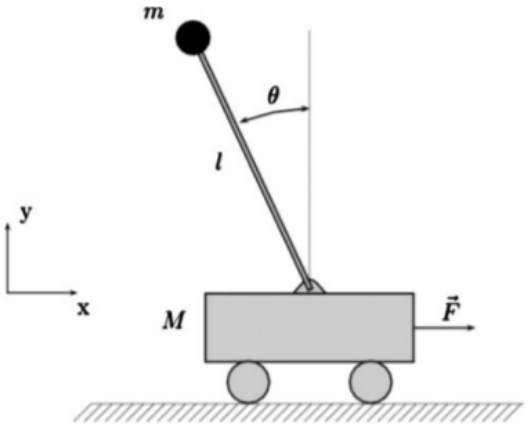

Figura 1. Problema do pêndulo invertido

O problema consiste em movimentar o carro, para a direita ou para a esquerda, de forma com que o pino fique equilibrado através do eixo. O pino, obviamente, também se move para a direita e para a esquerda, mas este não pode ser controlado.

4. Implementação

Para desenvolver o controlador utilizamos o simulador para pêndulo invertido do OpenIA Gym, que implementa o pêndulo através do ambiente ”CartPole-v0”. Também estamos utilizando outras bibliotecas em nossa implementação, como a **matplotlib** para desenhar um gráfico com a curva de aprendizagem do simulador.

4.1. Arquivo *cartpole.py*

O arquivo *cartpole.py* é o arquivo principal que contém a implementação do controlador. Nele importamos as bibliotecas necessárias para que o OpenIA Gym funcione de forma correta e fazemos algumas definições importantes.

Em nossa implementação utilizamos um dicionário para representar o conjunto de estados do problema. Esse dicionário é composto por uma chave, que identifica o estado, e as ações possíveis para o dado estado, sendo esquerda ou direita.

Como possuímos quatro variáveis de ambiente (posição do carro, velocidade do carro, ângulo do pêndulo, velocidade do pêndulo), e o domínio para cada uma delas é um intervalo linear, distribuímos esse intervalo em 10 novos intervalos, sendo $10^4 = 10.000$ a combinatória de todas as variáveis com todos os intervalos, consequentemente o número máximo de estados fica limitado em 10.000.

Além disso, criamos todos os estados no início da execução, e colocamos valor 0 para suas ações possíveis. Consequentemente precisamos definir uma forma de decidir por uma ação quando ambas as ações tem valor 0.

```
34  # Função de criação dos intervalDist <buckets>:
35  def createIntervalDists():
36      intervalDist = np.zeros((4,10)) # Divide o espaço linear de cada dôminio em 10 intervalos.
37      intervalDist[0] = np.linspace(-4.8, 4.8, 10) # Posição do Carrinho: [-4.8 a 4.8].
38      intervalDist[1] = np.linspace(-5, 5, 10) # Velocidade do Carrinho: [-inf a inf]. Fixado como [-5 a 5].
39      intervalDist[2] = np.linspace(-.418, .418, 10) # Angulo do Pêndulo: [-41.8 a 41.8].
40      intervalDist[3] = np.linspace(-5, 5, 10) # Velocidade do Pêndulo: [-inf a inf]. Fixado como [-5 a 5].
41      return intervalDist
```

Figura 2. Distribuição de intervalo para cada variável

Decidimos que neste cenário o algoritmo escolhe aleatóriamente uma ação: 0 (esquerda) ou 1 (direita). Posteriormente este mesmo estados terá seus valores de ação atualizados de acordo com a escolha feita.

O aprendizado se dá com a atualização do dicionário de estados, esse cálculo é feito ao término de cada episódio e utiliza a seguinte política:

*Q[s][a] += alfa * (reward + desc * Max(Q'[s'][a']) - Q[s][a]*

Precisamos definir também um limite de ações para o carro em cada episódio, pois após o algoritmo ter aprendido o problema, o carro poderia facilmente manter o pêndulo equilibrado e o algoritmo rodar infinitamente.

Por essa razão, foi definido que o número de ações possíveis para cada episódio é no máximo 200. Assim, a recompensa de um dado estado para aquele episódio não ultrapassará esse valor.

O desconto *desc* foi fixado em 0.9 e a recompensa é calculada a cada episódio, sendo uma recompensa positiva de 1 em caso de sucesso ou uma recompensa negativa de o dobro do máximo de ações possíveis em caso de fracasso.

Escolhemos uma recompensa negativa no valor de o dobro do valor máximo de ações possíveis para um episódio, pois precisamos dar uma punição grande para o estado que não mantiver o pêndulo equilibrado. Outro motivo para escolhermos essa recompensa foi que durante várias execuções observamos o comportamento do algoritmo, e constatamos que esse foi o valor que mais desempenhou sua função no problema proposto.

O algoritmo possui dois métodos para rodar o número de episódios definidos. O método *runEpisode* executa um episódio, acessando as variáveis de ambiente atuais (*observation*) e verificando qual estado corresponde a essa configuração atual, e assim executa a ação com maior valor, vide:

action = maxDict(Q[state])[0]

O método maxDict retorna a chave e o valor da ação, e nesse caso queremos a chave que representa a ação (0 ou 1).

Após a escolha, ele atualiza as variáveis de ambiente. Além disso, o método atualiza a recompensa total do episódio em questão. É nesse método também que é verificado se o limite de ações foi alcançado ou se o pêndulo caiu antes do máximo permitido, em decorrência disso atribui as respectivas recompensas.

O segundo método é o *runNEpisodes*, que roda uma quantidade N de episódios consecutivamente, armazenando o número de ações e recompensa final de cada episódio

que serão usadas para uma representação gráfica do aprendizado.

É nesse método também que criamos uma rotina para salvar o dicionário de estados, que pode ser utilizado para a criação do dicionário de uma nova execução.

4.2. Arquivo *qstates.npy*

O arquivo *qstates.npy* é usado para armazenar todo o espectro de estados possíveis definido para o problema. O dicionário de estados é salvo a cada 100 episódios executados.

Exemplo: '5555' : 0:-175.2, 1:-175.7

Neste caso o estado em questão é o de número 5555, tendo na ação 0 (esquerda) o valor -175,2 e para a ação 1 (direita) o valor -175,7.

5. Resultados

Durante a implementação do controlador criamos diversas variáveis que auxiliaram nos testes realizados, definimos por exemplo, algumas *flags* para controlar se queríamos ou não usar um dicionário gerado anteriormente ou se queríamos ou não mostrar todas as ações do carro para cada episódio.

Após a implementação, executamos o controlador com as definições listadas anteriormente e observamos que para 10.000 episódios o algoritmo convergiu rapidamente para o valor máximo de ações por episódio.

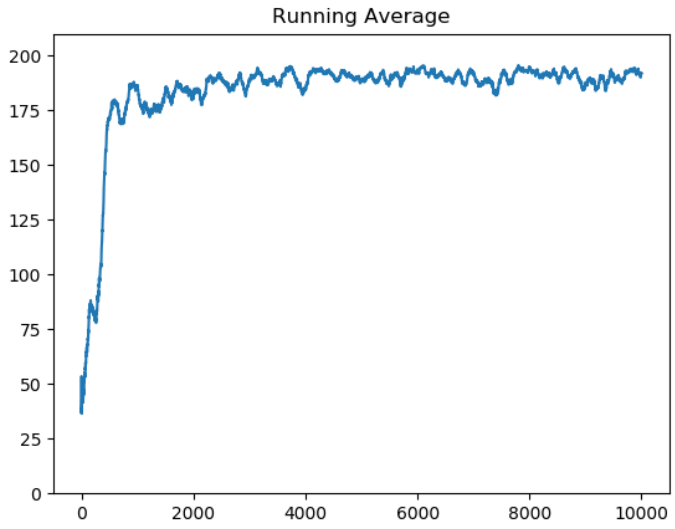

Figura 3. Média de recompensa sem conhecimento prévio

A Figura 3 mostra que a curva de aprendizagem sem conhecimento prévio, ou seja, quando o dicionário de estados é inicializado com zeros, é baixa. Ao longo da execução dos episódios a curva de aprendizagem (número de ações realizadas pelo carro) aumenta rapidamente até atingir valores próximos, ou iguais ao máximo de 200.

Sobre a convergência do problema, como observado na Figura 3, o algoritmo converge para um resultado satisfatório bem cedo. Entendemos que isso se deve ao fato do

problema ser relativamente simples: existem apenas duas ações possíveis (esquerda ou direita), e também ao fato de que usamos um grande número de episódios.

5.1. Utilizando conhecimento prévio

Em outra execução utilizamos o arquivo *qstates.npy* que contém o dicionário de estados gerado anteriormente para inicializar o dicionário de estados, ou seja, preservando o aprendizado. O resultado é mostrado na Figura 4.

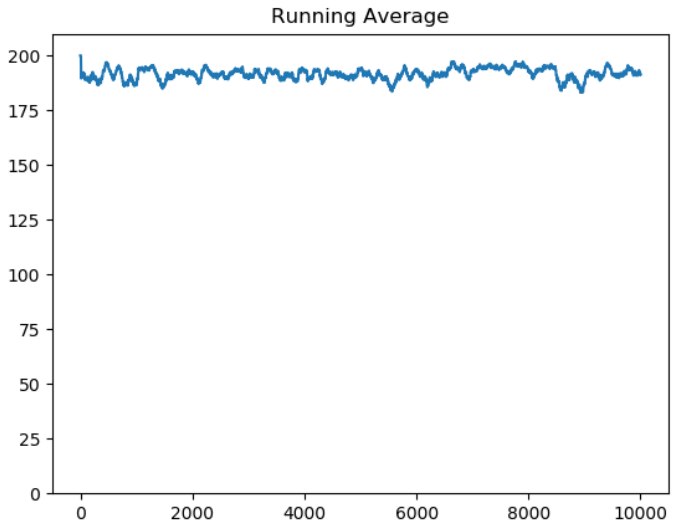

Figura 4. Média de recompensa com conhecimento prévio

Percebemos que na Figura 3 a convergência acontece apenas após a execução de vários episódios, isso é mostrado no crescimento da recompensa média. Já na Figura 4 a convergência acontece muito mais cedo, na realidade, desde a execução do primeiro episódio é possível observar um valor de recompensa médio bem elevado (próximo a 200), pois o algoritmo está utilizando o conhecimento prévido dos 10.000 episódios anteriores.

Referências

Coppin, B. (2004). *Inteligência Artificial*. LTC.

Luger, G. (2008). *Artificial Intelligence: structures and strategies for complex problem solving*. Pearson, 6th edition.