Deep Q Learning para ensinar inteligência artificial a jogar Asteroids

Vítor Kei Taira Tamada

Programa:
Bacharelado em Ciência da Computação

Orientador:
Prof. Dr. Denis Deratani Mauá

São Paulo, novembro de 2018

Deep Q Learning para ensinar inteligência artificial a jogar Asteroids

Esta é a versão original da monografia elaborada pelo aluno Vítor Kei Taira Tamada

Sumário

Capítulo 1

Introdução

Capítulo 2

Fundamentos

Para se criar e treinar uma inteligência artificial, diversos arcabouçous são necessários. Por um lado, existe a parte teórica e matemática na qual a inteligência se baseia para aprender. Por outro, do lado computacional, existem as bibliotecas que auxiliam no desenvolvimento, efetuando as contas necessárias e, neste trabalho em particular, emulando o jogo que serve de ambiente para o aprendizado. Este capítulo tem o intuito de familiarizar o leitor com a teoria e as ferramentas utilizadas no treinamento da inteligência artificial deste trabalho.

2.1 *Asteroids*

Asteroids é um jogo de fliperama do gênero *shooter* (jogo eletrônico de tiro) lançado em novembro de 1979 pela então desenvolvedora de jogos eletrônicos Atari Inc, atualmente conhecida apenas como Atari. O jogo foi inspirado por *Spacewar!*, *Computer Space*, *Space Invaders*, e *Cosmos*, sendo este último um jogo não finalizado.

O jogador controla uma nave espacial que se encontra em um campo de asteróides e precisa atirar para destruir todos enquanto evita colisões com os mesmos. A dificuldade aumenta conforme os asteróides se tornam mais numerosos. Diversas versões deste jogo foram criadas ao longo dos anos. Além das diferenças gráficas, as mudanças incluem naves espaciais inimigas que atiram contra o jogador, e tamanhos e formatos diferentes que os asteróides podem ter. *Asteroids* é considerado um dos primeiros grandes sucessos da era de ouro dos jogos de fliperama, época em que os jogos eletrônicos se estabeleceram como uma força dominante na cultura popular.

Para este trabalho, *Asteroids* foi emulado utilizando a plataforma Gym-Retro da companhia de pesquisas de inteligência artificial OpenAI. Todas as informações sobre o jogo apresentadas a seguir referem-se a tal versão. Para o aprendizado da inteligência artificial, não há naves inimigas e os asteróides podem assumir três tamanhos e três formatos distintos. Os tamanhos são grande (inicial), médio e pequeno, enquanto os formatos são consideravelmente parecidos. Há cinco comandos disponíveis: mover-se para frente, girar a nave no sentido horário, girar a nave no sentido anti-horário, atirar para frente, e entrar no hiper espaço. Mover-se para frente e girar no sentido horário ou anti-horário são as principais formas de movimento disponíveis ao jogador, e atirar serve para destruir os asteróides. Mover-se no hiper espaço consiste em fazer a nave desaparecer e, depois de alguns instantes, reaparecer em um local aleatório da tela. Há o risco de reaparecer em cima de um asteróide e, com isso, ter a nave destruída e perder uma vida. A nave possui aceleração e desaceleração - ou seja, inércia. Mesmo que o jogador deixe de pressionar o botão de mover-se para frente, ele continuará em movimento por um curto período de tempo antes de parar por completo. Isso gera um grau a mais de complexidade, pois faz com que manobras de esquiva e

curvas sejam mais difíceis de serem devidamente executadas.

2.2 Gym-Retro

Gym-Retro é uma plataforma para pesquisa de aprendizado por reforços e generalização em jogos desenvolvida e mantida pela empresa de pesquisas em inteligência artificial OpenAI. O lançamento mais recente inclui jogos do Sega Genesis, Sega Master System, Nintendo Entertainment System (NES), Super Nintendo Entertainment System (SNES) e Nintendo Game Boy, como ambientes para o desenvolvimento de IA, além de suporte preliminar para Sega Game Gear, Nintendo Game Boy Color, Nintendo Game Boy Advance e NEC TurboGrafx. Em qualquer um desses consoles, a ROM (*Read Only Memory*) do jogo é necessária. Apesar de não ter sido utilizada neste trabalho, a plataforma disponibiliza uma ferramente que permite criar *save states* (salvar um estado a partir do qual é possível continuar o jogo), encontrar locais da memória, criar cenários para o agente resolver, gravar e passar arquivos de vídeo, dentre outras funcionalidades.

Gym-Retro baseia-se na ferramenta Gym, desenvolvida e mantida pela OpenAI, que também tem como objetivo pesquisas em aprendizado por reforço, mas não apenas para jogos. Esta ferramenta foi utilizada por ter suporte para desenvolvimento de aprendizado para o jogo *Asteroids* e ser de fácil uso. A plataforma permite a entrada de oito ações diferentes: *UP*, *DOWN*, *LEFT*, *RIGHT*, *BUTTON*, *SELECT*, *RESET*, *null*, sendo que a ação realizada por cada botão varia de acordo com o jogo. Como descrito anteriormente, *Asteroids* utiliza apenas cinco deles: *UP* (mover-se para frente), *DOWN* (mover-se no híper espaço), *RIGHT* (girar no sentido horário), *LEFT* (mover-se no sentido anti-horário), e *BUTTON* (atirar). Os demais botões (*SELECT* e *RESET*) possuem funções relacioandas ao sistema e não ao jogo, e *null* corresponde a não realizar nenhuma ação.

2.3 TensorFlow

TensorFlow é um arcabouço de código aberto para computações numéricas de alta performance, desenvolvido e mantido pela Google. Seu núcleo de computação numérica flexível permite o uso da biblioteca em diversos campos da ciência. Oferece, em particular, grande suporte a aprendizado de máquina e aprendizagem profunda, ou, como é mais conhecido, *deep learning*. Esta ferramenta foi utilizada por oferecer uma API em Python estável, ter grande suporte, comunidade ativa, e ser de código aberto. Apesar de não ter sido utilizado, esta biblioteca também possui uma ferramenta de visualização de dados chamada TensorBoard.

2.4 Inteligência artificial (IA)

Inteligência artificial (IA) é um dos campos mais recentes de ciência e engenharia, tendo trabalhos no assunto sendo iniciados pouco depois da Segunda Guerra Mundial. Atualmente, ela é composta por diversos campos menores de estudo, podendo ser mais genérico, como aprendizado e percepção, até mais específico, como a capacidade de jogar um jogo, provar teoremas matemáticas, ou dirigir um carro em uma via movimentada. No livro Artificial Intelligence: A Morden Approach, de Stuart Jonathan Russel e Peter Norvig, oito definições são apresentadas em uma tabela de duas linhas por duas colunas. A linha de cima define processo de pensamento (*thought process*) e raciocínio (*reasoning*), enquanto a linha de baixo define comportamento (*behaviour*). Além disso, a coluna da esquerda mede o grau de fidelidade da inteligência quando comparado com performance humana, enquanto a da direita mede a racionalidade da performance - ou seja, se toma a ação "correta"dado o que o sistema sabe.

Pensando como um humano	**Pensando racionalmente**
O empolgante novo esforço de fazer computadores pensarem, serem máquinas com mentes, no sentido completo e literal da expressão (Haugeland, 1986)	*O estudo das faculdades mentais através de modelos computacionais* (Charniak & McDermott, 1985)
[A automação de] atividades que são associadas ao pensamento humano, como resolução de problemas, tomada de decisão, aprendizado, ... (Hellman, 1978)	*O estudo das computações que tornam possível a percepção, razão, e ação* (Winston, 1992)
Agindo como um humano	**Agindo racionalmente**
A arte de criar máquinas capazes de realizar funções que requerem inteligência quando feitas por pessoas (Kurzweil, 1990)	*Inteligência computacional é o estudo do design de agentes inteligentes* (Poole *et at*, 1998)
O estudo de como fazer os computadores fazerem coisas que, no momento, pessoas fazem melhor (Rich and Knight, 1991)	*IA... está relacionada a comportamento inteligente em objetos* (Nilsson, 1998)

Em linhas gerais, as definições da coluna da esquerda dizem respeito a uma inteligência artificial que se pareça com um humano, enquanto as da direita sobre uma inteligência artificial que toma ações visando estar correta e a atingir o melhor resultado possível. Este trabalho terá um foco maior na categoria "**Agindo racionalmente**", pois as ações tomadas pelo agente terão como objetivo o retorno da maior recompensa possível.

2.4.1 Aprendizagem por reforço

Aprendizagem por reforço (*reinforcement learning*) é uma técnica de aprendizado de inteligência artificial e uma das bases da utilizada neste trabalho. Para domínios mais simples, como Jogo da velha, é possível determinar qual a ação com maior recompensa esperada para cada estado - ou seja, a ação com maior probabilidade de vitória. Conforme o domínio se torna mais complexo, fazer esse mapeamento se torna inviável por conta da quantidade de estados que precisam ser armazenado, como é o caso do jogo *Asteroids*. Além disso, é comum haver situações em que não é possível determinar qual ação retornará a maior recompensa. Nesses casos, é mais viável criar e treinar um agente que aprenda a se comportar no ambiente em que está inserido do que informar se cada uma de suas ações em cada um dos estados possíveis é boa ou ruim.

Essa abordagem é conhecida como **Processo de Decisão de Markov** (*Markov Decision Process* (*MDP*)) para ambientes desconhecidos.

Processo de Decisão de Markov

Em um MDP padrão, a probabilidade de se chegar em um estado S' dado que o agente se encontra no estado S depende apenas da ação A tomada nesse estado s, o que caracteriza a **propriedade Markoviana**, existe um modelo probabilístico que caracteriza essa transição, dado por $P(S'|S, A)$; todos

os estados do ambiente e todas as ações que o agente pode tomar em cada estado são conhecidas; e a recompensa é imediatamente recebida após cada ação ser tomada.

As probabilidades de o agente tomar cada ação em um dado espaço são definidas por uma política π. A qualidade de uma política é medida por sua *utilidade esperada*, e a política ótima é denotada por π^*. Para calcular π^*, utiliza-se um algoritmo de iteração de valor (*value iteration*), que computa a utilidade esperada do estado atual: começando a partir de um estado arbitrário S tal que seu valor esperado é $V(S)$, aplica-se a equação de Bellman (*Bellman update* ou *Bellman equation*) até haver convergência de $V(S)$, que será denotado por $V^*(S)$. Esse $V^*(S)$ é usado para calcular a política ótima $\pi^*(s)$.

Seja i a iteração atual, S o estado atual, S' o estado futuro, A a ação tomada no estado atual, $R(S, A, S')$ a recompensa pela transição do estado S para o estado S' por tomar a ação A, e γ o valor de desconto (valor entre 0 e 1 tal que determina a importância de recompensas futuras para o agente), temos que:

Equação de Bellman:

$$V^{(i)}(S) = \max_A \sum_{S'} P(S'|S, A)[R(S, A, S') + \gamma V^{(i-1)}(S')] \tag{2.1}$$

Política gulosa para função valor ótima:

$$\pi^*(s) = \operatorname*{argmax}_A \sum_{S'} P(S'|S, A)[R(S, A, S') + \gamma V^*(S')] \tag{2.2}$$

Entretanto, essas fórmulas são aplicáveis somente quando as funções de reforço R e de probabilidade de transição P são conhecidas, que não é o caso do jogo *Asteroids*. Para lidar com esse problema, foi adotado o uso de **Q-learning**.

Q-learning

Quando não se conhece as probabilidades de transição, informação necessária para se obter a função valor pela equação de Bellman, é possível estimar $V(S)$ a partir de observações feitas sobre o ambiente. Logo, o problema deixa de ser tentar encontrar P e passa a ser como extrair a política do agente de uma função valor estimada.

Seja $Q^*(S, A)$ a função Q-valor que expressa a recompensa esperada por começar no estado S, tomar a ação A e continuar de maneira ótima. $Q^*(S, A)$ é uma parte da política gulosa para função valor ótima e é dada por:

$$\begin{aligned} Q^*(S, A) &= \sum_{S'} P(S'|S, A)[R(S, A, S') + \gamma V^*(S')] \\ &= \sum_{S'} P(S'|S, A)[R(S, A, S') + \gamma \max_{A'} Q^*(S', A')] \end{aligned} \tag{2.3}$$

Logo, substituindo 2.3 em 2.2, temos que a política gulosa ótima para a função Q-valor ótima é dada por:

$$\pi^*(S) = \operatorname*{argmax}_A Q^*(S, A) \tag{2.4}$$

O próximo passo será entender como atualizar a função Q-valor.

Supondo que o agente se encontra no estado S e toma a ação A, que causa uma transição no ambiente para o estado S' e gera uma recompensa $R(S, A, S')$, como computar $Q^{(i+1)}(S, A)$ baseado em $Q^{(i)}(S, A)$ e em $R(S, A, S')$, sendo i o momento atual? Para responder a essa pergunta, duas restrições precisam ser feitas: $Q^{(i+1)}(S, A)$ deve obedecer, pelo menos de forma aproximada, a equação de Bellman, e não deve

ser muito diferente de $Q^{(i)}(S, A)$, dado que são médias de recompensas. A seguinte equação responde essa questão.

Seja α a taxa de aprendizado (valor entre 0 e 1 que determina o quão importantes informações novas são em relação ao conhecimento que o agente possui),

$$\begin{aligned} Q^{(i+1)}(S, A) &= (1 - \alpha)Q^{(i)}(S, A) + \alpha[R(S, A, S') + \gamma \max_{A'} Q^{(i)}(S', A')] \\ &= Q^{(i)}(S, A) + \alpha[R(S, A, S') + \gamma \max_{A'} Q^{(i)}(S', A') - Q^{(i)}(S, A)] \end{aligned} \tag{2.5}$$

A convergência de $Q^{(i)}(S, A)$ em $Q^*(S, A)$ é garantida mesmo que o agente aja de forma subótima contanto que o ambiente seja um MDP, a taxa de aprendizado seja manipulada corretamente, e se a exploração não ignorar alguns estados e ações por completo - ou seja, raramente. Mesmo que as condições sejam satisfeitas, a convergência provavelmente será demasiadamente lenta. Entretanto, é interessante analisar os problemas levantados pela segunda e pela terceira condição que garantem a convergência e maneiras de solucioná-los.

Se a **taxa de aprendizado** for muito alta (próxima de 1), a atualização do aprendizado se torna instável. Por outro lado, se for muito baixa (próxima de 0), a convergência se torna lenta. Uma solução possível para essa questão é utilizar valores que mudam de acordo com o estado: utilizar valores mais baixos em estados que já foram visitados muitas vezes, pois o agente já terá uma boa noção da qualidade de cada ação possível, então há pouco que aprender; e utilizar valores mais altos em estados raramente visitados, pois o agente precisa aprender melhor sobre o estado.

Uma vez que a política é gulosa, o agente sempre tomará a ação que retorne a maior recompensa imediata. Isso é bom somente se todas as recompensas possíveis para aquele estados são conhecidas. Porém, se houver ações não exploradas, o agente pode perder uma recompensa maior do que ele já conhece apenas porque não está ciente de seu valor. Essa situação caracteriza o dilema *Exploration versus Exploitation*: é melhor tomar a ação que retorna a maior recompensa ou buscar uma melhor? Da mesma forma que na taxa de aprendizado, uma forma de contornar esse problema é mudar a probabilidade de decidir explorar o ambiente (*explore*) de acordo com a situação. Conforme o mundo é descoberto, se torna cada vez mais interessante agir de forma gulosa (*exploit*) do que explorar em estados muito visitado, e vice-versa em estados pouco visitados. Esse comportamento pode ser definido por uma função de exploração (*exploration function*).

Outro problema enfrentado por Q-learning é o de generalização. A política $\pi^*(S)$ determina a melhor ação a se tomar em cada estado. Logo, utiliza-se uma tabela para armazenar todas essas escolhas. Porém, como mencionado anteriormente, isso se torna inviável para espaços de estado muito grandes. Portanto, a solução é generalizar o aprendizado de um estado para o outro: se o agente sabe se comportar em um pequeno conjunto de estados, o ideal é ele saber o que fazer em um estado desconhecido contanto que seja parecido com um já aprendido. Em outras palavras, o agente aprende propriedades (*features*) dos estados ao invés dos estados propriamente ditos, e toma decisões a partir dessas informações. Essa forma de fazer escolhar é chamada de *approximate Q-learning*.

Approximate Q-learning

Para lidar com o enorme espaço de estados que alguns ambientes possuem, o agente armazena e aprende apenas algumas propriedades, que são funções de valor real, para tomar as decisões. Tais informações são armazenadas em um vetor e cada elemento desse vetor recebe um peso que determina a respectiva importância para que escolhas sejam feitas. Ou seja, a função Q-valor é representada por uma combinação linear das propriedades.

$$Q(S, A) = \omega_1 f_1(S, A) + \omega_2 f_2(S, A) + ... + \omega_n f_n(S, A) \tag{2.6}$$

Como o $V(S')$ é o valor esperado e $Q(S, A)$ é o valor previsto, a atualização pode ser interpretada como ajustar o valor do Q-valor pela diferença desses dois valores. Além disso, como o *approximate Q-learning* avalia características, apenas os pesos precisam ser atualizados:

$$\omega_k^{(i+1)}(S, A) = \omega_k^{(i)}(S, A) + \alpha[R(S, A, S') + \gamma V(S') - Q^{(i)}(S, A)] f_k(S, A), k = 1, 2, ..., n \tag{2.7}$$

Duas grandes vantagens de representar o Q-valor como uma combinação linear são evitar *overfitting* (a IA aprender tanto com o conjunto de treinamento que não consegue tomar decisões que diferem demais dele), e ser matematicamente conveniente, ter maneiras convenientes de calcular erro e funções que generalizem as decisões.

2.4.2 Aprendizagem de máquina - *Machine learning*

Aprendizagem profunda - *Deep learning*

Capítulo 3

Proposta

Capítulo 4

Desenvolvimento

Capítulo 5

Conclusões - Parte Subjetiva