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.

Diversas versões deste jogo foram criadas ao longo dos anos, então serão mencionadas somentes as características da iteração utilizada neste trabalho. O jogador controla uma nave espacial que se encontra em um campo de asteróides e precisa atirar para destrui-los enquanto evita colisões. A dificuldade aumenta conforme os asteróides se tornam mais numerosos. Os alvos podem assumir três tamanhos e três formatos diferentes. Enquanto os tamanhos são grande (inicial), médio e pequeno, os formatos são aproximadamente os mesmos em quesito de altura e largura. *Asteroids* é considerado um dos primeiros grandes sucessos da era de ouro dos jogos de fliperama, época em que os jogos eletrônicos começaram a se tornar comuns na cultura popular.

Para este trabalho, *Asteroids* foi emulado utilizando a plataforma Gym-Retro da companhia de pesquisas de inteligência artificial OpenAI. 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. Esse é um movimento de alto risco, pois é possível reaparecer em cima de um asteróide, resultando na perda de uma vida. Por outro lado, é útil para fugir rapidamente de situações complicadas. A nave também possui inércia, o que dificulta a movimentação dentro do jogo.

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. Essa ferramente permite emulação de diversos consoles de videogame, como Sega Genesis, Nintendo Entertainment System (NES) e, o utilizado neste trabalho, Atari2600. Para qualquer jogo que o usuário deseje emular, é necessário que ele tenha a ROM (*Read Only Memory*) do jogo.

O principal motivo de esta ferramenta ter sido escolhida é o suporte ao jogo *Asteroids* e pela facilidade de seu uso.

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 cienctíficos. Oferece, em particular, grande suporte a aprendizado de máquina e aprendizado profundo, ou, como é mais conhecida, *deep learning*. Esta ferramenta foi utilizada por oferecer uma API em Python estável, ter grande suporte, comunidade ativa, e ser de código aberto.

2.4 Inteligência artificial (IA)

Inteligência artificial (IA) é um dos campos mais recentes de ciência e de 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 matemáticos, 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 ao comportamento inteligente em objetos* (Nilsson, 1998)

Traduções livres feitas pelo autor

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 Aprendizado de máquina - *Machine learning*

Aprendizado de máquina, ou *machine learning* (ML) como é mais conhecido, é um campo de ciência da computação e um dos ramos de inteligência artificial que estuda a capacidade dos computadores de aprender a realizar uma tarefa sem ser explicitamente programado para isso. Após ter a tarefa definida, o computador recebe uma grande quantidade de dados, tenta reconhecer um padrão nessa entrada e, por fim, constrói um modelo para realizar predições. Por conta do enorme volume de dados necessários para o computador aprender, aprendizado de máquina é um campo que teve grandes avanços apenas nas últimas décadas, com o avanço da internet. Com cada vez mais pessoas tendo acesso a computadores mais rápidos e eficientes, bem como o surgimento de redes sociais, a quantidade de informação digital sendo gerada, armazenada, e disponibilizada cresceu. As três técnicas de aprendizado de máquina mais conhecidas são **aprendizado supervisionado**, **aprendizado não supervisionado**, e **aprendizado por reforço**. Aprendizado supervisionado e aprendizado por reforço serão os mais discutidos neste trabalho por serem as bases para as duas principais técnicas usadas: *Deep learning* e *Q-learning* respectivamente.

2.4.2 Aprendizado supervisionado

É mais fácil entender aprendizado supervisionado por meio de uma alegoria. O desenvolvedor da IA é um professor que fornece exercícios, cujas respostas ele possui, para o computador, seu aluno. O

computador resolve os exercícios e o professor diz se as respostas dadas por seu aluno estão corretas. Resolver o exercício é a tarefa delegada ao computador, os exercícios são os exemplos a partir dos quais o programa deve aprender, e a resposta que o professor tem é a saída esperada. Os exercícios com as respostas que o "professor"serão chamados de exemplos rotulados, enquanto a resposta do "aluno"será chamada de saída da IA.

Esta técnica costuma cair em dois tipos de problemas: classificação e regressão. Em problemas de classificação, deseja-se que o computador classifique corretamente uma entrada, dentre duas ou mais categorias pré-determinadas. Um exemplo simples deste tipo de problema é o de classificar se uma imagem é de cachorro ou não: após mostrar milhares de imagens para o computador, dizendo quais são de cachorro e quais não são, espera-se que ele classifique corretamente ao mostrar uma nova imagem. Em problemas de regressão, é esperado que os dados de entrada sigam uma função g, e o algoritmo deve encontrar uma função h que se aproxime o melhor possível de g. Um exemplo deste tipo de problema é fornecer a metragem uma residência, e ter o seu valor como saída da IA. Com entrada suficiente, o programa deve conseguir determinar o preço de uma residência apenas pelas medidas. Um problema pode ser tanto de classificação quanto de regressão a depender de como for montado. Por exemplo, ao invés de o programa dizer o preço do imóvel, ele poderia dizer se o preço é superior ou inferior a um certo valor.

Redes neurais (*neural networks*), Máquinas de Vetor de Suporte (*Support Vector Machine*, SVM) e Classificadores *Naive Bayes* são alguns dos algoritmos de aprendizado supervisionado mais comuns. O foco será em redes neurais neste trabalho.

Redes neurais - *Neural networks*

Redes neurais artificiais, mais conhecidas como redes neurais, são uma forma de processamento de informação inspirada no funcionamento do cérebro. Assim como o órgão que as inspirou, redes neurais possuem uma grande quantidade de elementos de processamento de informação conectados entre si chamados de neurônios, que trabalham em conjunto para resolver problemas. Dado que aprendem com exemplos, similar a pessoas, é considerado uma técnica de aprendizado supervisionado.

Com os avanços nesse campo nos últimos anos, diversos tipos diferentes de redes neurais foram desenvolvidos, como redes neurais convolucionais (*Convolutional neural networks*, CNN) e redes neurais de memória de curto-longo prazo[1] (*Long/Short Term Memory*, LSTM). Apesar de cada uma ter sua particularidade, redes neurais clássicas possuem duas características principais: os neurônios, e a estrutura dividida em camadas. Existem redes que não são redes neurais da maneira clássica, como Redes de Hopfield (*Hopfield Network*) e Máquinas de Boltzmann (*Boltzmann Machine*), que não serão abordadas neste trabalho.

Neurônios são funções que recebem como entrada a saída de cada neurônio da camada anterior, e devolvem um número, em geral entre 0 e 1 inclusos, cujo significado varia de acordo com o trabalho em questão. No caso deste, os neurônios da primeira camada têm como saída o número que representa cada pixel da imagem, após serem convertidos para uma escala de cinza e em um número entre 0 e 1. Os quadros (*frames*) de entrada também são redimensionados antes de serem analisados e processados pela IA. Os neurônios da última camada, por outro lado, têm como saída o valor utilidade esperado[2] a ser recebido caso aquela ação seja tomada. Esse valor é utilizado para se tomar a decisão em cada *frame*.

A estrutura das redes neurais clássicas é dividida em camadas que podem ser classificadas de três formas distintas: entrada, camadas ocultas, e saída. A entrada é o que a IA recebe inicialmente e precisa processar; as camadas ocultas (*hidden layers*) são o processamento; e a saída é uma série de números utilizados pela IA para tomar uma decisão, como probabilidades ou valores utilidade esperados. No caso

[1] Tradução livre feita pelo autor
[2] Mais detalhes sobre valor esperado na seção 2.4.3 - Aprendizado por reforço

deste trabalho, a entrada é o *frame* atual do jogo e a saída é o valor utilidade esperado de cada ação possível (mover-se, atirar e etc). Enquanto o número de neurônios na entrada e na saída são definidos pelo trabalho em questão, como número de pixels da tela e número de ações possíveis, o número de camadas ocultas e de neurônios em cada uma delas é arbitrário, sendo que essas quantidades são normalmente definidas por meio de experimentos.

Cada neurônio das camadas ocultas representa uma característica (*feature*) detectada pela IA ao longo do treinamento. Se essa característica estiver presente na camada de entrada, então o neurônio correspondente a essa característica será **ativado**. A ativação de um ou mais neurônios pode levar a ativação de outros neurônios na próxima camada e assim sucessivamente. Esse é um comportamento inspirado na forma como neurônios do cérebro enviam sinais de um para o outro. No caso das redes neurais, um neurônio é ativado quando a soma dos números de entrada passa de um certo valor e por uma função de ativação. Porém, nem todos os valores de entrada devem ser igualmente importantes, então cada um desses números recebe um peso que determina sua importância para a ativação da característica.

Em forma matemática, seja n o número de neurônios na camada anterior, $w_i, i = 1, ..., n$ os pesos das saídas de cada neurônio da camada anterior, e $a_i, i = 1, ..., n$ o valor de saída de cada neurônio da camada anterior e, por consequência, cada valor de entrada do neurônio atual, e b o valor que a soma deve ultrapassar para que o neurônio seja ativado, chamado de viés (*bias*).

$$w_1a_1 + w_2a_2 + ... + w_na_n - b \tag{2.1}$$

A soma 2.2 é então utilizada como entrada para a função de ativação escolhida pelo desenvolvedor da IA e, a depender da saída, o neurônio será ativado. Esse procedimento é feita em cada neurônio de cada camada da rede neural. Essas operações são mais fácil e rapidamente resolvidas utilizando matrizes.

Seja W a matriz tal que cada linha contém os pesos de cada neurônio da camada anterior para um determinado neurônio da camada atual, a^i o vetor tal que cada elemento é o valor de saída de cada neurônio da camada anterior, e b o viés para cada um dos neurônios da camada atual, é possível efetuar a soma 2.2 para todos os neurônios de uma camada da seguinte forma:

$$Wa^{(i)} + b, \qquad i = 1, ..., n \tag{2.2}$$

Apesar de as contas serem as mesmas quando feitas individualmente, esse formato é mais conveniente e eficiente de se programar por haver diversas bibliotecas que otimizam operações matriciais, além de ser mais fácil e rápido de ler e entender.

Dependendo do problema, essa soma pode assumir qualquer valor real. Contudo, é mais desejado que seja um valor entre 0 e 1 na maioria dos casos. Por conta disso, utiliza-se funções que convertem o resultado da soma para o intervalo de interesse chamadas de funções de ativação. Alguns exemplos de funções que eram ou são comumente utilizadas para isso são a sigmoide (também conhecida como curva logistica), ReLU (*Rectified Linear Unit*) e ELU (*Exponential Linear Unit*).

O próximo passo é entender como os valores dos neurônios e os respectivos pesos são utilizados para a inteligência conseguir soltar a resposta correta. Como mencionado anteriormente, conforme as características se mostram presentes na camada de entrada, os neurônios referentes a esses atributos são ativados, até que o neurônio com a resposta dada pela inteligência artificial seja ativado. No início, múltiplas saídas serão consideradas como boas respostas (alta recompensa esperada), pois a IA tem comportamento aleatório. Para que o computador saiba o quão ruim foi sua saída, é definida uma função de erro (também conhecida como função de custo). Uma forma comum de calcular esse erro é o erro quadrático médio, que consiste em subtrair o valor devolvido pela IA pelo valor esperado e elevar ao quadrado. Após essas operações serem realizadas em cada neurônio, os resultados são somados e obtém-

se um valor numérico para o quão errada a IA estava em um determinado exemplo. Quanto maior esse valor, mais incorreta a IA está. Após esse procedimento ser feito com milhares de exemplos, calcula-se a média dos erros obtidos e, com isso, avalia-se o desempenho da inteligência.

Percebe-se que esse procedimento para avaliar o desempenho do código é uma forma de encapsular a rede neural em uma função que tem como entrada os pesos e viés de cada neurônio e, como saída, o quão bom ou ruim eles são. Por ser uma função, é possível calcular "para onde ela deve se mover"de forma que minimize a saída utilizando um método chamado de gradiente descendente (*gradient descent*). Gradiente é uma generalização de derivada para multiplas variáveis. Ela representa a inclinação da tangente da função no ponto dado e aponta para a direção de maior crescimento, direção para onde a função deve "mover-se"para chegar no máximo local. Consequentemente, se tomar o valor negativo do gradiente da função, obtém-se a direção de maior decrescimento, direção do mínimo local. Em outras palavras, *gradient descent* é um algoritmo utilizado para encontrar o mínimo local de uma função, que é quase o que se deseja. O melhor caso para a função de erro seria encontrar o mínimo global, mas como isso nem sempre é possível, em parte por não haver garantias de que se encontrou o mínimo global, encontrar o melhor mínimo local é o suficiente.

De forma resumida, uma rede neural clássica aprende recebendo uma série de números como entrada e devolve uma saída; calcula-se o quão errada essa saída éem relação ao desejado para aquela determinada entrada e faz os ajustes necessários para minimizar o erro; após repetir esses passos milhares de vezes, espera-se que a IA tenha aprendido o suficiente a resolver o problema em mãos.

Aprendizado profundo - *Deep learning*

Como explicado anteriormente, redes neurais podem ser divididas em três tipos distintos de camadas: entrada, ocultas, e saída. Enquanto existe apenas uma camada de entrada e uma de saída, é possível haver uma ou mais camadas ocultas. Se houver muitas camadas ocultas, a rede neural passa a ser chamada de rede neural profunda (*deep neural network*). Atualmente, não existe uma definição exata de quantas camadas a rede neural precisa ter para começar a ser classificada como profunda e, mesmo que houvesse, esse número provavelmente mudaria com o passar do tempo.

Uma rede neural profunda que segue o modelo apresentado na seção anterior é chamado de *feedforward* e é o mais típico de *deep learning*. Ele recebe esse nome pois a informação flui da entrada para a saída sem haver conexões de *feedback* para que a previsão seja feita. Este tipo de rede neural forma a base para redes neurais convolucionais (*convolutional neural network*, CNN), técnica muito utilizada em reconhecimento de imagens. Como a ideia é treinar uma inteligência artificial que aprende vendo a tela do jogo, foi escolhido esse tipo de rede neural para este trabalho.

Uma das partes mais peculiares sobre *deep learning* é o fato de muitos dos avanços feitos nos últimos anos serem resultado de tentativa e erro e não de formalização e demonstração da teoria. Ainda não se entende perfeitamente o que faz uma rede neural profunda funcionar tão bem. As características na construção de um modelo consideradas como essenciais mudam rapidamente com o passar do tempo, com algumas inclusive passando a ser consideradas estorvos ao invés de atributos chave para acelerar e aprimorar o aprendizado.

Rede neural convolucional - *Convolutional neural network*

2.4.3 Aprendizado por reforço

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:

$$V^{(i)}(S) = \max_A \sum_{S'} P(S'|S, A)[R(S, A, S') + \gamma V^{(i-1)}(S')] \tag{2.3}$$

Política gulosa para função valor ótima:

$$\pi^*(s) = \operatorname*{argmax}_A \sum_{S'} P(S'|S, A)[R(S, A, S') + \gamma V^*(S')] \tag{2.4}$$

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:

$$\begin{aligned} Q^*(S,A) &= \sum_{S'} P(S'|S,A)[R(S,A,S') + \gamma V^*(S')] \\ &= \sum_{S'} P(S'|S,A)[R(S,A,S') + \gamma \max_{A'} Q^*(S',A')] \end{aligned} \tag{2.5}$$

Logo, substituindo 2.5 em 2.4, temos que a política gulosa ótima para a função Q-valor ótima é dada por:

$$\pi^*(S) = \operatorname*{argmax}_A Q^*(S,A) \tag{2.6}$$

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),

$$\begin{aligned} Q^{(i+1)}(S,A) &= (1-\alpha)Q^{(i)}(S,A) + \alpha[R(S,A,S') + \gamma \max_{A'} Q^{(i)}(S',A')] \\ &= Q^{(i)}(S,A) + \alpha[R(S,A,S') + \gamma \max_{A'} Q^{(i)}(S',A') - Q^{(i)}(S,A)] \end{aligned} \tag{2.7}$$

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.

$$Q(S, A) = \omega_1 f_1(S, A) + \omega_2 f_2(S, A) + ... + \omega_n f_n(S, A) \tag{2.8}$$

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:

$$\omega_k^{(i+1)}(S, A) = \omega_k^{(i)}(S, A) + \alpha[R(S, A, S') + \gamma V(S') - Q^{(i)}(S, A)]f_k(S, A), k = 1, 2, ..., n \tag{2.9}$$

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.

Capítulo 3

Proposta

Capítulo 4

Desenvolvimento

Capítulo 5

Conclusões - Parte Subjetiva