Aprendizado por Reforço - Algoritmo *Q-Learning*

Matheus Prado Prandini Faria[1], **Rita Maria da Silva Julia**[1], **Lídia Bononi Paiva Tomaz**[1]

[1]Universidade Federal de Uberlândia (UFU)
Faculdade de Computação (FACOM)
Programa de Pós-graduação em Ciência da Computação (PPGCO)

1. Aprendizado por Reforço

Aprendizado por Reforço (AR) refere-se a uma área de Aprendizado de Máquina (AM) na qual um agente aprende a tomar suas decisões baseando-se no resultado de suas ações por meio da interação com um determinado ambiente. Neste sentido, um agente irá deliberar com o ambiente percebendo o seu estado atual e selecionando ações segundo uma política. O efeito de cada ação gera um valor, denominado recompensa. Este valor é gerado a partir de uma função de recompensa que retorna um sinal de reforço ou penalidade dependendo do estado atingido como resultado desta ação. Ressalta-se que não é dado ao agente a informação de qual ação ele deve realizar, isto é, ele deve descobrir as ações que levam às maiores recompensas por meio de tentativa e erro. Assim, o agente é totalmente responsável por selecionar as ações que serão aplicadas sobre o estado atual, por esse motivo os agentes da AR são caracterizados como autônomos [Martins et al. 2007].

A questão que envolve a AR é basicamente: como um agente autônomo que atua sobre um determinado ambiente pode aprender a escolher suas ações para alcançar seus objetivos? Este é um problema muito comum em tarefas como o controle de um robô remoto e aprender a jogar jogos eletrônicos, como por exemplo o clássico jogo da cobrinha (*Snake*). A qualidade da sequência de ações é definida pela função de recompensa [Harmon and Harmon 1996]. Por exemplo, no jogo *Snake*, quando o agente consegue encontrar a maçã ele receberá um reforço (recompensa positiva, maior que zero), caso morra (situação em que bate em alguma das paredes ou em alguma parte de seu corpo) receberá uma penalidade (recompensa negativa, menor que zero), mas nos movimentos intermediários sua atuação será neutralizada (recompensa igual ou próxima a zero).

Um sistema típico de aprendizagem por reforço constitui-se, basicamente, de um agente interagindo em um ambiente via percepção e ação [Russell and Norvig 2003]. O agente percebe as situações dadas no ambiente (pelo menos parcialmente) e seleciona uma ação a ser executada em consequência de sua percepção. A ação executada muda, de alguma forma, o ambiente; e as mudanças são comunicadas ao agente por um sinal de reforço [Neto 2007]. A Figura 1 ilustra esta interação.

Conforme pode ser observado na Figura 1, um sistema de AR é constituído por dois elementos principais: *agente* e *ambiente*. Todavia, este tipo de sistema pode contar com quatro subelementos:

Uma política: define a forma de o agente aprender a se comportar em determinado momento. Em outras palavras, uma política π é um mapeamento de estados $s \in S$ percebidos em ações $a \in A$ a serem tomadas naquele momento. Assim, uma política pode ser definida em um valor $\pi(s,a)$. Se um agente AR muda a sua política, então as probabilidades de seleção de ações sofrem mudanças e, consequentemente, o

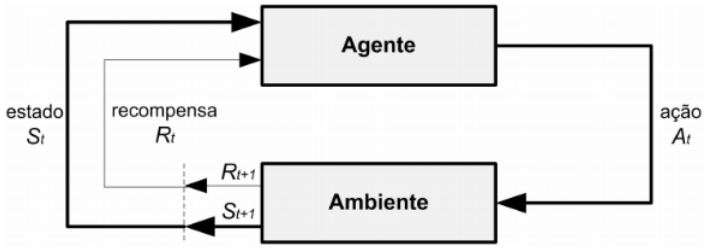

Figura 1. Ciclo de interação entre o agente e o ambiente na aprendizagem por reforço [Sutton and Barto 1998]

comportamento do sistema apresenta variações à medida que o agente vai acumulando experiência a partir das interações com o ambiente. Portanto, o processo de aprendizado no sistema AR pode ser expresso em termos da convergência até uma política ótima ($\pi^*(s,a)$) que conduza à solução do problema de forma ótima;

Função de recompensa: mapeia cada estado do ambiente (ou par estado-ação) em um valor. Tal valor define o reforço ou a penalidade que o agente irá receber como recompensa relacionado ao resultado da execução de uma ação que, consequentemente, altera o estado do ambiente. O objetivo da AR é maximizar as recompensas recebidas ao longo do tempo. A função de recompensa pode servir como base para ajustar a política utilizada. Por exemplo, se a escolha de uma ação resulta em uma penalidade, a política pode ser alterada para que outra ação possa ser tomada em tal situação no futuro de modo a melhorar o reforço do agente;

Função valor: especifica a recompensa que o agente espera acumular sobre o futuro, partindo de um certo estado. Enquanto a função de recompensa avalia um estado em sentido imediato, a função valor avalia o estado a longo prazo. Isso permite que um certo estado receba uma baixa recompensa, mas um alto valor, visto que ele pode ser seguido de outros estados com alta recompensa. O inverso também pode ocorrer. O objetivo é procurar ações que tragam melhores valores do que recompensas, visto que a longo prazo o efeito tende a ser mais positivo em relação às recompensas. Contudo, recompensas são mais fáceis de calcular, pois são dadas diretamente pelo ambiente, ao passo que valores devem ser estimados a partir da sequência de observações que o agente faz durante sua atuação no problema. Assim, em um algoritmo de AR, o componente de maior importância é aquele que estima valores de modo eficiente [Sutton and Barto 1998];

Modelo do ambiente : constitui uma forma de simular o comportamento do ambiente. Por exemplo, dado um par estado-ação, o modelo pode predizer o próximo estado e a recompensa a ser retornada. Modelos são utilizados para planejar, isto é, tem por meta auxiliar na decisão de ações considerando situações possíveis antes delas ocorrerem.

O agente deve continuamente selecionar ações a fim de deliberar no ambiente. Esta escolha de ações pode ser efetuada a partir de duas abordagens: os métodos *on-policy* e *off-policy*. Nos métodos *on-policy* as funções de valor são atualizadas usando resultados da execução de ações determinadas por alguma política, isto é, esta atualização é baseada apenas na experiência do agente (considerando ações que já foram executadas). Estas políticas são usualmente ”soft”e não determinísticas. O significado de ”soft”neste

caso garante que sempre haverá um elemento de exploração para a política, isto é, $\pi(a—s)$ ¿ 0 para todo $s \in S$ e para todo $a \in A$. A política não é rigorosa para selecionar sempre a ação que gera a maior recompensa. Os métodos *off-policy* também utilizam políticas "soft", todavia, se diferenciam dos métodos *on-policy* por estimar funções valor usando ações hipotéticas (ações que não foram testadas ainda). Portanto, os métodos *off-policy* podem separar exploração de controle, isto é, um agente treinado utilizando um método *off-policy* pode aprender táticas que não necessariamente existiram durante sua fase de aprendizado. As questões que envolvem os métodos *off-policy* são consideravelmente mais complicadas do que as que envolvem os métodos *on-policy* e por isso tem recebido a atenção dos pesquisadores de AR nas últimas décadas [Sutton et al. 2009].

A fim de tornar a dinâmica de aprendizagem mais efetiva, um sistema de AR deve ainda considerar um balanço entre duas características de obtenção de informações do ambiente que são denominadas exploração e explotação. Estas características, apesar de parecerem semelhantes, apresentam uma diferença fundamental. Ao atuar em um problema de AR, o agente deve priorizar as ações que já foram empregadas no passado de modo a maximizar a recompensa gerada. Para isso, o agente deve tentar ações que ainda não foram selecionadas. Neste contexto, o agente deve usufruir (explotar) sua experiência, mas também deve experimentar (explorar) novas ações. O dilema entre exploração e explotação é que nenhuma dessas características podem ser atingidas sem que o agente falhe. Por isso, o agente deve testar diversas ações e progressivamente escolher aquelas que geram melhores recompensas. Em uma tarefa estocástica, cada ação deve ser testada diversas vezes a fim de ser possível obter uma estimativa real da recompensa esperada [Sutton and Barto 1998].

Resumidamente, o modelo de um sistema AR consiste em [Sutton and Barto 1998]:

- um conjunto de variáveis de estado percebidas pelo agente. As combinações de valores dessas variáveis formam um conjunto de estados discretos do agente (S);
- um conjunto de ações discretas, que escolhidas pelo agente mudam o estado do ambiente (A(s), onde $s \in S$);
- um conjunto de valores das transições de estados (reforços tipicamente entre [0,1]).

Assim, para um agente ser bem-sucedido em uma tarefa de AR, ele necessita das seguintes habilidades: aprender sobre a interação entre estados, ações e recompensas subsequentes; determinar qual a melhor ação a ser escolhida a partir do seu aprendizado sobre a interação com o ambiente.

1.1. Fundamentos Matemáticos

Existem dois conceitos que devem ser conhecidos para facilitar a modelagem de um problema como um sistema de AR: Propriedade de Markov e Processo de Decisão de Markov (PDM). A seguir apresenta-se uma breve descrição destes conceitos, baseando-se em [Sutton and Barto 1998]. Por simplificação adotou-se um número finito de estados e valores de recompensas, uma vez que tal medida possibilita a representação de somatórios e probabilidades ao invés de integrais e densidades de probabilidade, mas toda argumentação pode ser estendida para incluir estados e recompensas contínuas.

1.2. Propriedade de Markov

Quando a probabilidade de transição de um estado s para um estado s' depende apenas do estado *s* e da ação *a* adotada em *s*, isso significa que o estado corrente fornece a informação suficiente para o sistema de aprendizado decidir que ação será tomada. Quando o sistema possui esta característica diz-se que ele satisfaz a *Propriedade de Markov* [Bellman 1957].

No caso mais geral, se a resposta em *t + 1* para uma ação efetuada em *t* depende de todo o histórico de ações até o momento atual, a dinâmica do ambiente é definida pela especificação completa da distribuição de probabilidades, como mostra a equação abaixo:

$$Pr\{s_{t+1} = s^{'}, r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\} \quad (1)$$

onde a probabilidade (Pr) do próximo estado s_{t+1} ser o estado s' e o reforço r_{t+1} é um função que depende de todos os estados, ações e reforços passados.

Logo, se a resposta do ambiente à ação a_t depende apenas dos estados e reforços em *t*, então a probabilidade da transição para o estado s' em t+1 pode ser obtida pela (2), a qual corresponde a uma simplificação da equação (1) do caso geral:

$$Pr^{a}_{s,s'} = Pr\{s_{t+1} = s^{'}, r_{t+1} = r | s_t, a_t, r_t\} \quad (2)$$

A probabilidade de transição satisfaz às seguintes condições:

1)

$$P^{a}_{s,s'} \geq 0, \forall s, s_{\prime} \in S, \forall a \in A(s) \quad (3)$$

2)

$$\sum_{s_{\prime} \in S} P^{a}_{s,s'} = 1, \forall s \in S, \forall a \in A(s) \quad (4)$$

A Propriedade de Markov é de fundamental importância na AR, uma vez que tanto as decisões como os valores são funções apenas do estado atual, abrindo a possibilidade de métodos de soluções incrementais, onde podem-se obter soluções a partir do estado atual e dos estados futuros, como é feito nos métodos das Diferenças Temporais.

1.2.1. Processo de Decisão de Markov (PDM)

O Processo de Decisão de Markov descreve, formalmente, um ambiente de aprendizado por reforço como um processo estocástico no qual o estado no próximo passo de tempo depende apenas do estado no passo de tempo atual e da decisão escolhida no presente, ou seja, o futuro independe do passado dado o presente (Propriedade de Markov) [Limnios and Oprişan 2001].

Segundo [Sutton e Barto 1998], um *Processo de Decisão de Markov* (PDM) - no inglês *Markov Decision Process* (MDP) - é definido como um conjunto de estados *S*, um conjunto de ações *A*(s), $\forall s \in S$, um conjunto de transições entre estados associadas com as ações, um conjunto de probabilidades P sobre o conjunto *S* que representa uma modelagem das transições entre os estados e um conjunto de recompensas R associadas a cada

estado ou a cada par estado-ação. Assim, dado um par de estado e ação, a probabilidade do estado *s* passar para um estado $s^{'}$ é:

$$Pr^{a}_{s,s'} = Pr\{s_{t+1} = s^{'} | s_t = s, a_t = a\} \quad (5)$$

onde Pr é a probabilidade de transição, isto é, Pr representa a probabilidade do estado s_{t+1} ser $s^{'}$, sempre que o estado s_t for igual a *s* e a ação a_t for igual a a. Desta forma, a dependência que o estado seguinte s_{t+1} seja o estado $s^{'}$ está relacionada a tomar a ação *a* no instante *t*.

Similarmente, dados um estado e ação atuais e um estado seguinte $s^{'}$, o valor esperado do retorno *R* é:

$$R^{a}_{s,s'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s^{'}\} \quad (6)$$

Os valores de probabilidade $P^{a}_{s,s'}$ e retorno esperado $R^{a}_{s,s'}$ determinam os aspectos mais importantes da dinâmica de um MDP finito:

1. um ambiente evolui probabilisticamente baseado em um conjunto finito e discreto de estados;
2. para cada estado do ambiente existe um conjunto finito de ações possíveis;
3. cada passo que o sistema de aprendizado executar uma ação, é verificado um custo positivo ou negativo para o ambiente em relação à ação executada; e,
4. estados são observados, ações são executadas e reforços são relacionados.

Assim, para quase todos os problemas de AR é suposto que o ambiente tenha a forma de um MDP, desde que seja satisfeita a Propriedade de Markov no ambiente. Nem todos os métodos de AR necessitam de uma modelagem MDP inteira do ambiente, mas é necessário ter-se pelo menos a visão do ambiente como um conjunto de estados e ações [Sutton and Barto 1998]. Dentre os métodos AR, destacam-se os métodos DT, Monte Carlo e Programação Dinâmica [Sutton and Barto 1998]. Na subseção a seguir, serão apresentados alguns exemplos de formulação de problemas de aprendizado por reforço utilizando o conceito de PDM. Em seguida, o método DT será detalhado por ser a base de compreensão do algoritmo *Q-Learning* que também será explicado posteriormente.

1.3. Exemplos PDMs

Nesta subseção serão apresentados dois exemplos sobre o uso de PDM para modelar problemas de aprendizado por reforço. O primeiro exemplo explorado é relacionado à tarefa de se atuar no jogo *Snake*. Em seguida, é descrito um outro problema com características distintas, denominado "Quem quer dinheiro?".

1.3.1. *Snake*

No jogo *Snake*, o agente controla a cobra e o seu objetivo é chegar até a posição onde se encontra a maçã. Alcançar tal objetivo representa vitória, no entanto, morrer (bater em alguma das paredes que limitam o tamanho da tela) e, consequentemente, não alcançando o objetivo, representa derrota. Tal jogo é ilustrado na Figura 2. A maçã é representada pelo *grid* vermelho. A cabeça da cobra, a qual guia o seu movimento, é representada pelo *grid* verde.

Dessa forma, uma possível formulação do jogo *Snake* utilizando o conceito de PDM é a seguinte:

- **Estados**: representados pelas imagens do jogo.
- **Ações**: movimentos para cima, baixo, direita e esquerda.
- **Recompensas**: recompensa positiva (1) em caso de vitória; recompensa negativa (-1) em caso de derrota; recompensa neutra (0) em caso de movimentos intermediários, os quais não levam nem à vitória nem à derrota.

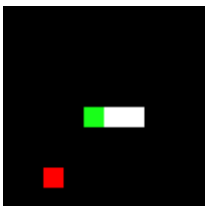

Figura 2. Jogo *Snake*

Nota-se que a função de transição não foi especificada pelo fato do ambiente do *Snake* ter uma característica determinística, ou seja, ao se realizar uma ação a em um estado s, o resultado sempre é conhecido e garantido. Não existe incerteza/aleatoriedade sobre o estado final do ambiente após a execução da ação. Por exemplo, na situação de jogo apresentada na Figura 2, caso seja efetuada a ação de movimento para baixo, é garantido que a cobra irá seguir o comando para baixo. Dada a ação para baixo, não existe uma probabilidade de efetuar, por exemplo, o movimento para baixo de 50% e não efetuar a ação de 50%.

1.3.2. ”Quem quer dinheiro?”

No problema do ”Quem quer dinheiro?”, ilustrado na Figura 3, o agente possui uma dúvida ”estarrecedora”sobre sua tomada de decisão. Independentemente do baú que escolher, é garantido que receberá dinheiro. A diferença entre os baús consiste na quantia em dinheiro que cada um possui. Desta forma, o baú da esquerda sempre contém o valor de 10 reais. Já o baú da direita, na maior parte das vezes contém o valor de um real, no entanto, há a possibilidade de conter 100 reais (maior valor dentre os citados).

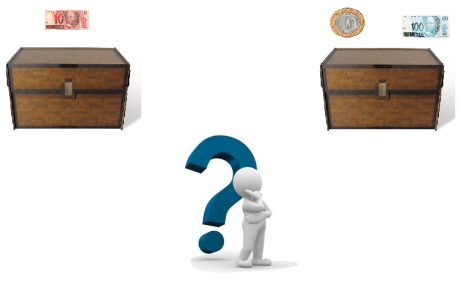

Figura 3. Problema do ”Quem quer dinheiro”

Suponha que a probabilidade do baú da direita conter um real seja de 90% e, consequentemente, de conter 100 reais seja de 10%. A partir de tal situação, o ambiente do ”Quem quer dinheiro?”pode ser representado de acordo com a Figura 4. É possível notar que uma mesma ação, no caso ir rumo ao baú da direita, pode resultar em diferentes estados finais (baú com um real e baú com 100 reais) com uma determinada probabilidade associada a cada um deles. A esta característica é dado o nome de estocasticidade, uma vez que uma mesma ação *a* em um estado *s* não irá sempre gerar como resultado um mesmo estado *s'*.

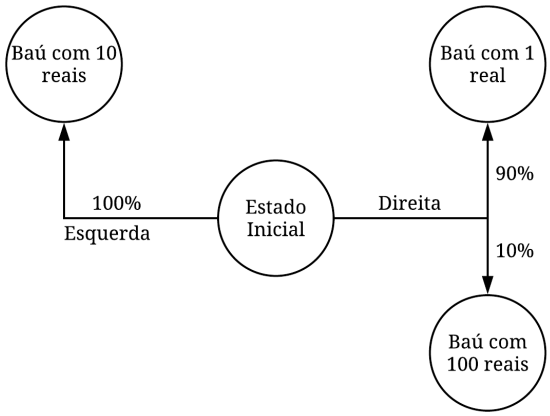

Figura 4. Ambiente estocástico do ”Quem quer dinheiro?”

Assim, uma possível formulação do ”Quem quer dinheiro?”apresentado utilizando o conceito de PDM é a seguinte:

- **Estados**: posição inicial do agente, baú contendo um real, baú contendo 10 reais e baú contendo 100 reais.
- **Ações**: movimento para a direita ou esquerda.
- **Função de Transição**: movimento para esquerda leva ao baú que contém 10 reais com probabilidade de 100%. Movimento para a direita leva ao baú que contém um real com uma probabilidade de 90% e ao baú que contém 100 reais de 10%.
- **Recompensas**: quantia em dinheiro encontrada no baú.

2. Método das Diferenças Temporais

Os métodos das Diferenças Temporais (DT) não exigem um modelo exato do sistema e permitem serem incrementais na busca de soluções para problemas de predição. Tais métodos são capazes de utilizar o conhecimento anterior em ambientes parcialmente conhecidos para predizer o comportamento futuro. Aprender a predizer é uma das formas mais básicas e predominantes em aprendizagem. Por meio de um certo conhecimento, alguém poderia aprender a predizer, por exemplo:

- Se uma determinada disposição de peças no tabuleiro de Xadrez conduzirá à vitória;
- Se uma determinada formação de nuvens acarretará em chuva;

- Se para uma determinada condição econômica de um país, isto implicará em um aumento ou diminuição na bolsa de valores.

Os métodos DT se destacam em relação a outros métodos de AR por ser totalmente *on-line* (iterativo) e incremental. Além disso, não necessita de um modelo do ambiente, de uma recompensa e uma distribuição de probabilidade para o próximo estado, conforme ocorre nos métodos baseados em Programação Dinâmica; e não tem a necessidade de aguardar até o final de um episódio para gerar o valor de reforço do agente, conforme ocorre nos métodos Monte Carlo [Sutton and Barto 1998]. O Algoritmo 1 apresenta o processo de aprendizagem dos métodos DT, adaptado de [Li 2017].

Algorithm 1 Algoritmo DT

Input: Política π
Output: Função Valor V

function APRENDIZAGEMDT
  Inicializar a Função Valor V (por exemplo, 0 para todos os estados)
  for `cada episódio` **do**:
    Inicializar o estado inicial s
    for `cada passo do episódio` **do**:
      $a \leftarrow$ ação dada pela política π para o estado s
      executar ação a, observar novo estado s' e recompensa r
      $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
      $s \leftarrow s'$
      if estado s é terminal **then break**

Os métodos DT são guiados pelo erro ou diferença entre predições sucessivas temporárias de estados sequenciais experimentados por um agente em um domínio. Desta forma, se um agente executa uma sequência de ações a_0,...,a_{i-1},a_i,a_{i+1},...,a_j como uma etapa preliminar de planejamento para tentar alcançar um determinado objetivo *f*, o sistema calcula o valor de predição associado a cada estado resultante da execução de cada uma destas ações. Os valores das sucessivas diferenças entre as predições serão usados para atualizar a função de aprendizado do sistema. Consequentemente, nos métodos DT, todas as ações executadas durante o treinamento do agente possuem impacto no processo de aprendizagem, o que caracteriza a propriedade dos métodos DT de considerar o impacto ponderado de cada ação a_i executada ao longo do tempo (ou seja, o histórico de todas as ações executadas influencia no aprendizado, sendo que o impacto de cada ação diminui em função de sua antiguidade na linha do tempo). Assim, a avaliação de uma política que defina o comportamento do agente sobre um ambiente, determinando que ação este deve executar em cada estado, é abordada como um problema de predição, isto é, estimar a função V^π sob a política π. Logo, nos métodos DT, a eficiência do aprendizado depende, diretamente, da precisão com que a função de avaliação estima o valor de predição dos estados envolvidos.

3. Q-Learning

Q-Learning é um famoso algoritmo de aprendizado por diferenças temporais [Watkins 1989]. Tal algoritmo é interessante por assumir que o agente não tem nenhum

tipo conhecimento do domínio, isto é, o agente não conhece tanto a função de transição quanto a recompensa para todos os estados no ambiente de modo a construir um modelo do mesmo. Desta forma, o agente necessita apenas de conhecer os estados existentes e as ações possíveis para cada um deles a fim de aprender a agir otimamente experimentando as consequências de suas ações a partir de sua interação com o ambiente. Tal fato garante a seguinte vantagem a este algoritmo: fácil adaptabilidade ao lidar com problemas com transições e recompensas estocásticas, sem exigir adaptações. O algoritmo 2 apresenta o processo de aprendizagem do *Q-Learning*.

Algorithm 2 Algoritmo *Q-Learning*

Output: Função Ação-Valor Q

1: **function** APRENDIZAGEM*QLearning*
2: Inicializar a Função Ação-Valor Q (por exemplo, 0 para todos os estados)
3: **for** `cada episódio` **do**:
4: Inicializar o estado inicial s
5: **for** `cada passo do episódio` **do**:
6: $a \leftarrow$ ação para o estado s derivada da Tabela Q (por exemplo, estratégia *epsilon-greedy*)
7: executar ação a, observar novo estado s' e recompensa r
8: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
9: $s \leftarrow s'$
10: **if** estado s é terminal **then break**

O objetivo do *Q-Learning* é aprender uma política a qual norteia a tomada de decisão de um agente. Isto é, tal política indica a um agente qual ação tomar sob determinadas circunstâncias. Para tanto, é utilizada uma tabela, denominada Tabela Q, a qual armazena os valores da função ação-valor Q para cada ação em cada estado. Este valor representa a qualidade a longo prazo de se executar uma determinada ação em um determinado estado. Tal tabela é representada por uma matriz de tamanho MxN (M corresponde ao número de estados possíveis e N corresponde ao número de ações válidas). A função Q é submetida a um processo iterativo de atualização de valores (definida na linha 8 do Algoritmo 2). Desta forma, o *Q-Learning* encontra uma política ótima no sentido de maximizar o valor da recompensa total a longo prazo a partir do estado atual. Há uma prova matemática em [Watkins and Dayan 1992] que garante a convergência do *Q-Learning* para uma política ótima sob condições razoáveis para a taxa de aprendizado (α) e sob a execução de um número infinito de episódios em ambientes com espaços de estados e ações finitos.

A Figura 5 ilustra o ciclo do processo de aprendizagem do *Q-Learning* em mais alto nível em relação ao especificado no Algoritmo 2. Assim, a fim de compreender melhor o funcionamento do treinamento do *Q-Learning*, cada passo presente no referido ciclo será explicado detalhadamente nas próximas subseções. Por fim, será apresentado um exemplo da aplicação de tal algoritmo.

3.1. Inicialização da Tabela Q

Conforme já mencionado, o tamanho da Tabela Q é proporcional ao número de estados possíveis (M) e ao número de ações válidas (N). Desta forma, tal tabela é composta por M

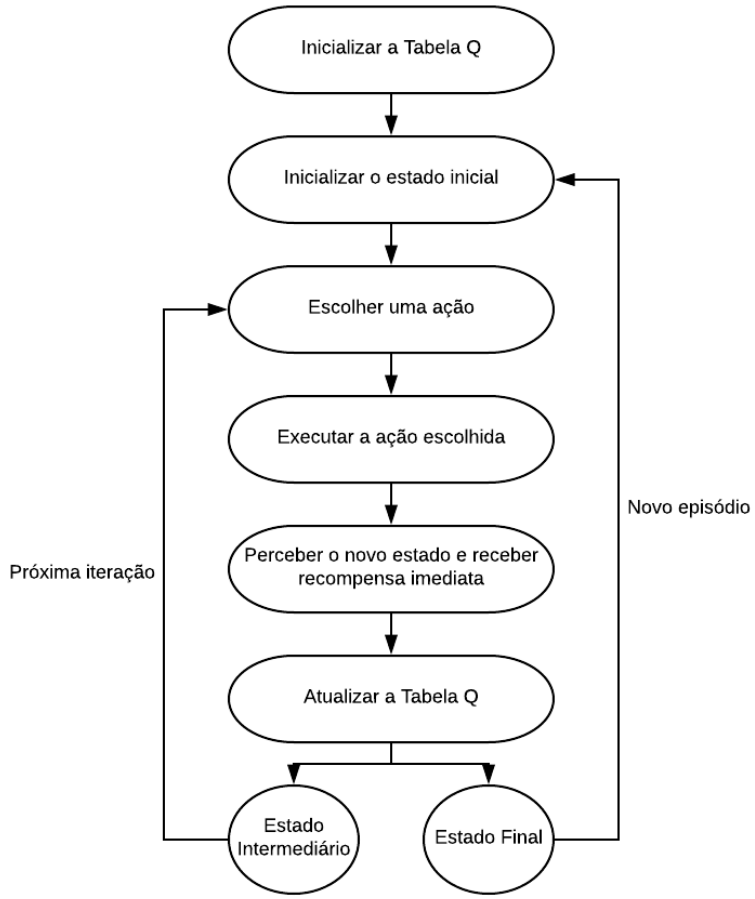

Figura 5. Processo de aprendizagem do *Q-Learning* ema alto nível

linhas e N colunas. A forma mais usual de inicialização dos valores da função Q é atribuir o valor zero (valor neutro) a todos os pares estado-ação possíveis. Assim, nenhum par de estado e ação recebe algum tipo de vantagem (em termos de preferência ou prioridade) sobre os demais inicialmente.

3.2. Inicialização do Estado Inicial

Este passo marca o início de um episódio. O estado inicial depende diretamente do domínio/problema abordado.

3.3. Escolha de uma ação (Dilema Exploração x Explotação)

Na aprendizagem por reforço, uma questão fundamental é a que envolve o balanceamento entre exploração e explotação, conforme introduzido na seção 1. Explotação, no atual contexto, significa usufruir das melhores ações conhecidas (efetuação de ações gulosas). No caso da escolha de ações considerando a Tabela Q, tal termo está relacionado ao fato de selecionar a ação *a* que está atualmente com maior valor no estado corrente *s* (Q(s,a)). Exploração, por sua vez, significa experimentar diferentes ações (efetuação de ações aleatórias) a fim de ampliar a gama de informações sobre a qualidade de ações distintas. Desta forma, o dilema entre exploração e explotação gera as seguintes dúvidas [Robin 2007]:

- Quando *gulosamente* aproveitar da estimação atual do valor da função Q e escolher a ação que a maximiza (explotação)?
- Quando *curiosamente* explorar outra ação que pode melhorar a estimação atual da função Q (exploração)?

Por exemplo, suponha que o algoritmo aprendeu que Q(*s*,*a*) para um determinado estado *s* e uma ação *a* é igual a 50. Será que é benéfica a tentativa de se executar uma ação *a'* no estado *s* se, até o atual momento, Q(*s*,*a'*) vale 10? Responder a esta questão não é algo trivial, pois é necessário avaliar diversos fatores [Prati 2011], tais como: as características inerentes ao ambiente (por exemplo, totalmente observável x parcialmente observável; determinístico x estocástico; entre outras); a quantidade de ações já efetuadas (ou número de episódios já concluídos); quantidade de ações restantes (ou número de episódios restantes).

Neste sentido, duas estratégias amplamente utilizadas para efetuar o balanceamento entre exploração e explotação são *epsilon-greedy* (ϵ-*greedy*) e *Softmax* [Sutton and Barto 1998]. Utilizando a estratégia ϵ-*greedy* [Watkins 1989], a cada passo de tempo, o agente seleciona uma ação aleatória com uma probabilidade fixa, ϵ, onde $0 \leq \epsilon \leq 1$. Com probabilidade $1 - \epsilon$ o agente seleciona uma das ações ótimas aprendidas com relação à função Q. Assim:

$$\pi(s) = \begin{cases} \text{ação aleatória dentre as válidas}, & \text{se } \xi < \epsilon \\ \text{argmax}_{a \in A(s)} Q(s,a), & \text{caso contrário}, \end{cases} \tag{7}$$

onde $0 \leq \xi \leq 1$ é um número aleatório uniforme sorteado em cada passo de tempo. Em contraste, a estratégia *Softmax* utiliza probabilidades de seleção de ações determinadas por meio da classificação de estimativas da função valor a partir da distribuição de *Boltzmann* [Tokic and Palm 2011]:

$$\pi(a|s) = Pr\{a_t = a | s_t = s\} = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_b \frac{e^{Q(s,b)}}{\tau}}, \tag{8}$$

onde τ é um parâmetro positivo denominado temperatura, conceito derivado da técnica *simulated annealing* [Russell and Norvig 2003]. Altas temperaturas fazem com que todas as ações sejam quase iguais (em termos de probabilidade de seleção), o que propicia a exploração. Já baixas temperaturas causam a seleção de ações gulosas, o que propicia a explotação.

As duas referidas estratégias possuem vantagens e desvantagens como descrito em [Sutton and Barto 1998]. No entanto, em termos de facilidade de implementação e entendimento, a ϵ-*greedy* geralmente leva preferência sobre a *Softmax*.

3.4. Execução da Ação

Este passo representa a execução da ação *a* - selecionada na etapa anterior a partir da estratégia ϵ-*greedy* - sobre o estado atual *s*.

3.5. Percepção do novo estado e recompensa imediata

Este passo representa a percepção do novo estado *s'* e da recompensa imediata *r* - resultados da execução da ação *a* efetuada no passo anterior.

3.6. Atualização da Tabela Q

A atualização da Tabela Q é a parte central do algoritmo *Q-Learning*, pois representa a forma como o agente aprende a qualificar (a longo prazo) o valor de uma certa ação em um determinado estado. Desta forma, é atualizado o valor Q(s,a) da ação *a* executada sobre o estado *s*, gerando como resultado o estado *s'* e recompensa imediata *r*. Para tanto, é utilizada a seguinte equação de ajuste:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_a Q(s',a) - Q(s,a)] \tag{9}$$

De modo a compreender o ajuste realizado, é importante entender os significados de cada termo da equação (9):

- **Q(s,a)**: valor que deseja-se atualizar. Representa a qualidade a longo prazo (em termos de maximização de recompensas) de se executar a ação *a* sobre o estado *s*.

- **Taxa de aprendizado** (α): esta taxa determina o impacto que a nova informação adquirida possui sobre a informação antiga. Isto é, representa o quanto será aprendido pelo agente com relação ao ajuste calculado. Uma taxa de aprendizado igual a zero representa que a nova informação não será considerada, ou seja, há somente explotação do conhecimento anterior. Já uma taxa igual a um representa que o agente passará a considerar apenas a nova informação, ou seja, a informação antiga será totalmente descartada.

- **Fator de desconto** (γ): este fator determina a relevância das ações futuras. Um fator de desconto igual a zero representa que o agente utilizará somente a recompensa imediata (o futuro não tem nenhuma importância em seu aprendizado), ou seja, ele não terá a habilidade de tomar boas decisões a longo prazo, somente decisões baseadas no imediatismo. Já um fator igual a um representa uma importância máxima do futuro, no entanto, pode causar divergência nos valores da função ação valor.

- **max**$_a$ **Q(s',a)**: maior valor da função Q para o novo estado *s'* (representa a ação gulosa em s').

Apesar do ajuste temporal ser relativo a toda parte indicada dentro dos colchetes na equação (9), a parte principal da equação que remete à ideia da diferença temporal é a seguinte:

$$[max_a Q(s',a) - Q(s,a)] \tag{10}$$

É possível notar que a intensidade do ajuste é norteado diretamente pela diferença entre o valor atual da melhor ação no estado subsequente (*s'*) e o valor da ação executada no estado corrente *s*. Tais valores são relativos à função Q. Desta forma, quanto maior a diferença entre os dois valores, maior será a intensidade do ajuste. Da mesma forma, quanto menor for a diferença, menor será a intensidade do ajuste. Quando tal diferença for mínima, de modo que torne o ajuste insignificante (ou seja, o valor do ajuste resultante é bem próximo ou igual a zero e não altera o valor Q(*s*,*a*)), é dito que o algoritmo *Q-Learning* convergiu para tal valor da função ação valor.

3.7. Atualização do estado corrente

Neste passo, conforme a linha 9 do Algoritmo 2, o estado corrente torna-se o estado resultante da ação a sobre o estado s da seguinte forma:

$$s \leftarrow s' \tag{11}$$

Além disso, verifica-se se o novo estado corrente corresponde a um estado final. Caso positivo, o episódio é encerrado e um novo episódio pode ser iniciado. Caso contrário, o episódio segue para a próxima iteração (passo de tempo).

3.8. Exemplo: *Gridworld*

O *gridworld* do Mário é apresentado na Figura 6. Neste problema, o objetivo do agente - representado pelo personagem Mário - é encontrar o melhor caminho até a princesa *Peach*. A tartaruga - personagem *Koopa Troopa* - representa um obstáculo para o agente. Nota-se que há dois caminhos possíveis para o agente atingir seu objetivo: um primeiro que envolve passar pela tartaruga (obstáculo), considerado o caminho mais difícil; um segundo que representa um caminho livre de obstáculos até a princesa, considerado o caminho mais fácil. Intuitivamente, o melhor caminho aqui é seguir por aquele mais fácil.

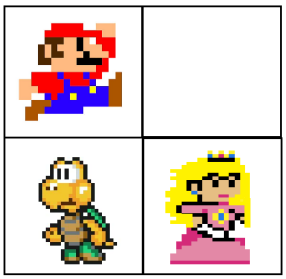

Figura 6. *Gridworld* do Mário

Para aplicar o algoritmo *Q-Learning* ao referido problema, é necessário definir, primeiramente, algumas características relevantes referentes ao conceito de PDM:

- **Estados (S)**: representados por cada quadrante do *gridworld*.
- **Ações (A)**: movimentos para cima, baixo, direita e esquerda.
- **Recompensas (R)**: quadrante com a princesa - recompensa positiva (1); quadrante com obstáculo - recompensa negativa (-1); quadrantes em branco - recompensa neutra (0).

Dessa forma, o gridworld pode ser visto como na Figura 7. O estado inicial é representado pelo primeiro quadrante (quadrante um), onde o agente está localizado. O estado final é representado pelo último quadrante (quadrante quatro), onde a princesa está localizada. Além disso, nota-se que é necessário respeitar as restrições de ações válidas

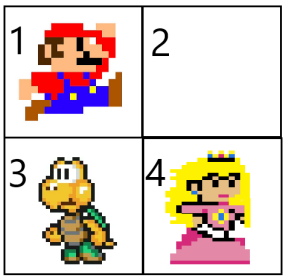

Figura 7. Estado inicial do *gridworld* do Mário

delimitadas por cada quadrante. Por exemplo, para o quadrante um, as ações válidas são os movimentos para baixo e para a direita.

Assim sendo, será mostrado em detalhes a execução de um episódio do algoritmo *Q-Learning* sobre o referido problema. Desta forma, antes de iniciá-lo, é preciso definir alguns parâmetros importantes para o algoritmo, tais como a taxa de aprendizado, o fator de desconto, entre outros. Para este exemplo, serão utilizados os seguintes parâmetros:

- **Taxa de Aprendizado** (α): 0.1;
- **Fator de Desconto** (γ): 0.9;
- **Valor** ϵ **(estratégia** ϵ***-greedy***): 0.9;

Além disso, inicializa-se a Tabela Q. Conforme subseção 3.1, a tabela inicializada com os valores da função Q é apresentada na Tabela 1. Destaca-se que o símbolo '-' (hífen) representa que a ação da coluna não pode ser realizada no estado da linha.

	Cima	**Baixo**	**Esquerda**	**Direita**
Quadrante 1	-	0	-	0
Quadrante 2	-	0	0	-
Quadrante 3	0	-	-	0
Quadrante 4	0	-	0	-

Tabela 1. Tabela Q inicializada

3.8.1. Execução de um episódio

O episódio é iniciado com o Mário localizado no primeiro quadrante, estado inicial, conforme ilustrado pela Figura 7. Em seguida, é utilizada a estratégia ϵ*-greedy* para selecionar uma ação com ϵ igual a 0.9. Suponha que o valor ξ sorteado seja igual a 0.5. Como $\xi < \epsilon$, é selecionada uma ação exploratória. As ações válidas para o quadrante 1 são os movimentos para baixo e para a direita. Suponha que a ação para baixo tenha sido selecionada aleatoriamente. A execução da ação *baixo* (a) no estado *quadrante 1* (s) gera como resultado o estado *quadrante 3* (s') e recompensa imediata igual a -1 (r), pois o estado gerado contém um obstáculo, conforme ilustrado na Figura 8.

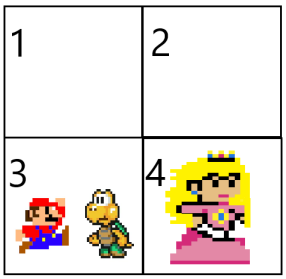

Figura 8. Resultado da execução da ação *baixo* no estado *quadrante 1*

Assim, é efetuada a atualização do valor da função Q para o estado atual *quadrante 1* e a ação executada *baixo*. Os seguintes valores serão utilizados para a atualização:

- Q(*quadrante 1*, *baixo*) = 0 (valor atual de Q(*quadrante 1*, *baixo*));
- $max_a Q(\textit{quadrante 3}, a)$ = 0 (melhor valor da função Q para o novo estado gerado *quadrante 3*, conforme Tabela Q);
- r = -1 (recompensa imediata ao atingir o novo estado gerado *quadrante 3*).

Conforme a equação (9), o cálculo do novo valor de Q(*quadrante 1*,*baixo*) é exibido a seguir:

$$Q(\textit{quadrante 1}, \textit{baixo}) = 0 + \alpha[r + \gamma max_a Q(\textit{quadrante 3}, a) - 0] \tag{12}$$

$$Q(\textit{quadrante 1}, \textit{baixo}) = 0 + 0.1 * [-1 + 0.9 * 0 - 0] \tag{13}$$

$$Q(\textit{quadrante 1}, \textit{baixo}) = -0.1 \tag{14}$$

A nova Tabela Q é apresentada na Tabela 1. Nota-se que o único valor alterado foi o Q(*quadrante 1*,*baixo*).

	Cima	**Baixo**	**Esquerda**	**Direita**
Quadrante 1	-	-0.1	-	0
Quadrante 2	-	0	0	-
Quadrante 3	0	-	-	0
Quadrante 4	0	-	0	-

Tabela 2. Tabela Q atualizada após o primeiro passo do episódio

Por fim, o estado corrente é atualizado para o *quadrante 3*. Como é um estado intermediário - não representa fim de episódio - então é realizado um próximo passo (iteração) do episódio.

Novamente, será escolhida uma ação para o estado atual. Suponha que o valor ξ sorteado neste segundo passo do episódio seja igual a 0.1. Assim, será realizada uma ação de exploração. As ações válidas para o estado *quadrante 3* são os movimentos para cima e para a direita. Suponha que o movimento para cima tenha sido escolhido aleatoriamente. A execução da ação *cima* (*a*) no estado *quadrante 3* (*s*) gera como resultado o estado *quadrante 1* (*s'*) e recompensa imediata igual a 0 (*r*) - pois *quadrante 1* é um estado neutro - conforme ilustrado na Figura 7.

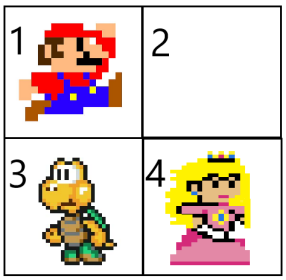

Figura 9. Resultado da execução da ação *cima* no estado *quadrante 3*

Assim, é efetuada a atualização do valor da função Q para o estado atual *quadrante 3* e a ação executada *cima*. Os seguintes valores serão utilizados para a atualização:

- Q(*quadrante 3*, *cima*) = 0 (valor atual de Q(*quadrante 3*, *cima*));
- $max_a Q(quadrante\ 1, a)$ = 0 (melhor valor da função Q para o novo estado gerado *quadrante 1*, conforme Tabela Q);
- r = 0 (recompensa imediata ao atingir o novo estado gerado *quadrante 1*).

Conforme a equação (9), o cálculo do novo valor de Q(*quadrante 3*,*cima*) é exibido a seguir:

$$Q(quadrante\ 3, cima) = 0 + \alpha[r + \gamma max_a Q(quadrante\ 1, a) - 0] \tag{15}$$

$$Q(quadrante\ 3, cima) = 0 + 0.1 * [0 + 0.9 * 0 - 0] \tag{16}$$

$$Q(quadrante\ 3, cima) = 0 \tag{17}$$

A nova Tabela Q é apresentada na Tabela 3. Nota-se que o valor Q(*quadrante 3*,*cima*) não foi alterado após o ajuste, pois já estava associado ao valor zero (0).

Por fim, o estado corrente é atualizado para o *quadrante 1*. Como é um estado intermediário - não representa fim de episódio - então é realizado um próximo passo (iteração) do episódio. Assim sendo, um terceiro passo do episódio é iniciado com o estado atual *quadrante 1*.

	Cima	Baixo	Esquerda	Direita
Quadrante 1	-	-0.1	-	0
Quadrante 2	-	0	0	-
Quadrante 3	0	-	-	0
Quadrante 4	0	-	0	-

Tabela 3. Tabela Q atualizada após o segundo passo do episódio

No terceiro passo do episódio, será escolhida uma ação para o estado atual *quadrante 1*. Suponha que o valor ξ sorteado neste passo seja igual a 0.95. Assim, como $\xi \geq \epsilon$, será realizada uma ação de explotação. Pelos valores da função ação valor mais recentes, apresentados na Tabela 3, o melhor valor associado ao estado *quadrante 1* é relativo à coluna da ação *direita*, o qual vale zero (0). Desta forma, a ação *direita* (*a*) é selecionada e executada sobre o estado *quadrante 1* (*s*), gerando como resultado o estado *quadrante 2* (*s'*) e recompensa imediata igual a 0 (*r*) - pois *quadrante 2* é um estado neutro - conforme ilustrado na Figura 10.

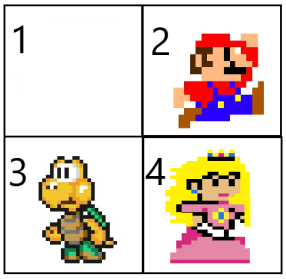

Figura 10. Resultado da execução da ação *direita* no estado *quadrante 1*

Assim, é efetuada a atualização do valor da função Q para o estado atual *quadrante 1* e a ação executada *direita*. Os seguintes valores serão utilizados para a atualização:

- Q(*quadrante 1*, *direita*) = 0 (valor atual de Q(*quadrante 1*, *direita*));
- $max_a Q(quadrante\ 2, a)$ = 0 (melhor valor da função Q para o novo estado gerado *quadrante 2*, conforme Tabela Q);
- r = 0 (recompensa imediata ao atingir o novo estado gerado *quadrante 2*).

Conforme a equação (9), o cálculo do novo valor de Q(*quadrante 1*,*direita*) é exibido a seguir:

$$Q(quadrante\ 1, direita) = 0 + \alpha[r + \gamma max_a Q(quadrante\ 2, a) - 0] \tag{18}$$

$$Q(quadrante\ 1, direita) = 0 + 0.1 * [0 + 0.9 * 0 - 0] \tag{19}$$

$$Q(quadrante\ 1, direita) = 0 \tag{20}$$

Assim sendo, a nova Tabela Q é apresentada na Tabela 4. Nota-se que o valor Q(*quadrante 1*,*direita*) não foi alterado após o ajuste, pois já estava associado ao valor zero (0). Percebe-se que, já no início, o algoritmo consegue identificar que, a partir do estado inicial (*quadrante 1*), o melhor caminho provavelmente será seguir pela direita ao invés de seguir para baixo, uma vez que já associou um valor menor para a ação *baixo* em relação a ação *direita*.

	Cima	**Baixo**	**Esquerda**	**Direita**
Quadrante 1	-	-0.1	-	0
Quadrante 2	-	0	0	-
Quadrante 3	0	-	-	0
Quadrante 4	0	-	0	-

Tabela 4. Tabela Q atualizada após o terceiro passo do episódio

Por fim, o estado corrente é atualizado para o *quadrante 2*. Como é um estado intermediário - não representa fim de episódio - então é realizado um próximo passo (iteração) do episódio. Assim, um quarto passo do episódio é iniciado com o estado atual *quadrante 2*.

No quarto passo do episódio, será escolhida uma ação para o estado atual *quadrante 2*. Suponha que o valor ξ sorteado neste passo seja igual a 0.7. Assim, será realizada uma ação de exploração. As ações válidas para o estado *quadrante 2* são os movimentos para baixo e para a esquerda. Suponha que o movimento para baixo tenha sido escolhido aleatoriamente. A execução da ação *baixo* (*a*) no estado *quadrante 2* (*s*) gera como resultado o estado *quadrante 4* (*s'*) e recompensa imediata igual a 1 (*r*) - pois *quadrante 4* é um estado que contém a princesa - conforme ilustrado na Figura 11.

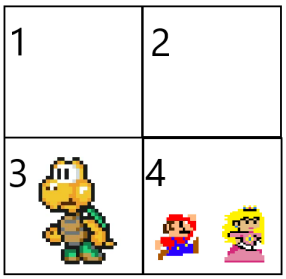

Figura 11. Resultado da execução da ação *baixo* no estado *quadrante 2*

Assim, é efetuada a atualização do valor da função Q para o estado atual *quadrante 2* e a ação executada *baixo*. Os seguintes valores serão utilizados para a atualização:

- Q(*quadrante 2*, *baixo*) = 0 (valor atual de Q(*quadrante 2*, *baixo*));
- $max_a Q(quadrante\ 4, a)$ = 0 (melhor valor da função Q para o novo estado gerado *quadrante 4*, conforme Tabela Q);
- r = 1 (recompensa imediata ao atingir o novo estado gerado *quadrante 4*).

Conforme a equação (9), o cálculo do novo valor de Q(*quadrante 2*,*baixo*) é exibido a seguir:

$$Q(\textit{quadrante 2}, \textit{baixo}) = 0 + \alpha[r + \gamma max_a Q(\textit{quadrante 4}, a) - 0] \tag{21}$$

$$Q(\textit{quadrante 2}, \textit{baixo}) = 0 + 0.1 * [1 + 0.9 * 0 - 0] \tag{22}$$

$$Q(\textit{quadrante 2}, \textit{baixo}) = 0.1 \tag{23}$$

Assim sendo, a nova Tabela Q é apresentada na Tabela 5. Nota-se que o novo valor Q(*quadrante 2*,*baixo*) é igual a 0.1.

	Cima	**Baixo**	**Esquerda**	**Direita**
Quadrante 1	-	-0.1	-	0
Quadrante 2	-	0.1	0	-
Quadrante 3	0	-	-	0
Quadrante 4	0	-	0	-

Tabela 5. Tabela Q atualizada após o terceiro passo do episódio

Por fim, o estado corrente é atualizado para o *quadrante 4*. Como é um estado final - representa fim de episódio - então o atual episódio é finalizado.

É interessante destacar que, em apenas um episódio, o *Q-Learning* já demonstra um certo avanço com relação ao aprendizado do aspecto geral do ambiente *Gridworld* do Mário ao verificar que a qualidade de uma ação a qual leva diretamente a um obstáculo é menor do que aquelas que levam a um estado neutro (que não contém obstáculo e nem o objetivo), além do fato de ações que levam diretamente ao estado final receberem um valor maior em relação as ações que não levam a tal estado.

3.8.2. *Q-Learning* após 10.000 episódios

Ao aplicar o *Q-Learning* ao problema do *Gridworld* do Mário considerando 10.000 episódios com os mesmo parâmetros especificado na subseção 3.8.1, os valores Q convergem conforme apresentado na Tabela 6.

	Cima	**Baixo**	**Esquerda**	**Direita**
Quadrante 1	-	-0.1	-	0.9
Quadrante 2	-	1	0.81	-
Quadrante 3	0.81	-	-	1
Quadrante 4	0	-	0	-

Tabela 6. Tabela Q após a execução de 10.000 episódios sobre o *Gridworld* do Mário

Ressalta-se que, ao fim dos 10.000 episódios, o *Q-Learning* conseguiu aprender a política ótima para o referido problema. Na fase de operação (também denominada fase de teste), na qual não há atualização dos valores Q, o algoritmo sempre seleciona a ação com maior valor Q para o estado corrente. Tal fase será executada a seguir.

O estado inicial é ilustrado na Figura 12.

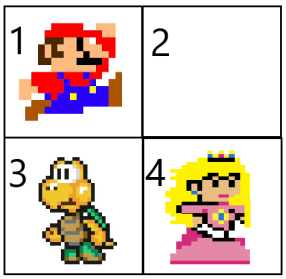

Figura 12. Estado inicial do *Gridworld*

Pela Tabela 6 a melhor ação para o estado *quadrante 1* (estado inicial) é o movimento para a direita. Desta forma, a execução de tal ação produz como resultado o estado *quadrante 2*, conforme Figura 13.

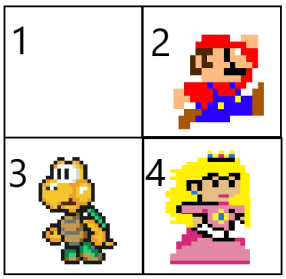

Figura 13. Estado 2 do *Gridworld*

Agora, pela Tabela 6 a melhor ação para o estado *quadrante 2* é o movimento para baixo. Desta forma, a execução de tal ação produz como resultado o estado *quadrante 4* (estado final), conforme Figura 14.

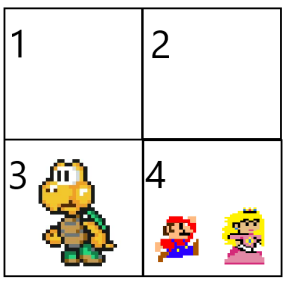

Figura 14. Estado final do *Gridworld*

Portanto, o melhor caminho encontrado pelo *Q-Learning* é o seguinte: *quadrante 1*; *quadrante 2*; *quadrante 4*.

3.9. Breve análise da complexidade do *Q-Learning*

Em relação a complexidade de tempo, a tarefa de escolher uma ação é barata [Prati 2011]. No entanto, o tempo de treinamento cresce conforme o aumento do número de estados.

Em relação a complexidade de espaço, uma vez que é necessário armazenar uma tabela contendo os valores Q para todos os pares estado-ação, tem-se que o espaço é O(número de estados x número de ações).

3.10. Problemas

O principal problema relacionado ao *Q-Learning* é referente a sua complexidade de espaço. Pelo fato do número de estados possíveis crescer exponencialmente com a quantidade de características representadas, quanto mais complexo for um ambiente (em termos de número de estados e número de ações), mais memória é necessária para a execução do algoritmo, o que torna o processo de aprendizagem mais oneroso. Além disso, o uso do *Q-Learning* em espaços contínuos torna-se significativamente mais complicado [Irpan 2015], sendo geralmente necessário discretizar os estados para contornar este gargalo. De modo a ilustrar tal cenário, será considerada a tarefa de bater faltas no jogo eletrônico *FIFA* - produzido pela *Electronic Arts Games* - conforme Figura 15.

(a)

(b)

Figura 15. Diferença mínima entre os *pixels* de duas imagens em um mesmo contexto (tarefa de bater faltas no jogo *FIFA*)

A Figura 15 apresenta duas imagens que representam a mesma situação geral, porém com uma diferença pouco perceptível de *pixels*, relativa a posição do braço do agente (na Figura 15b) o braço está mais próximo ao corpo do agente em relação à Figura 15a)). Neste sentido, se os estados forem representados pelas imagens do jogo (mais especificamente, pela matriz de *pixels* que as compõem), o *Q-Learning* consideraria as duas imagens como sendo diferentes e, consequentemente, criaria duas linhas na Tabela Q para representar, respectivamente, os estados relativos às imagens 15a) e 15b). Nota-se que isto gera um problema de redundância de informações, uma vez que ambas imagens representam o mesmo estado geral de jogo, mas com uma diferença mínima de distribuição dos *pixels* (praticamente insignificante ao olho humano).

Além disso, considerando que um agente tenha sido treinado com o *Q-Learning* para o referido cenário de bater faltas no *FIFA* utilizando os *pixels* (imagens) para representar os estados, a seguinte situação pode ocorrer na fase de teste: o agente se depara com uma imagem não existente em seus estados armazenados na Tabela Q, isto é, o agente atingiu um estado (distribuição de *pixels*) nunca visto antes. Neste cenário, como tal imagem não existe em sua Tabela Q, então o agente não consegue decidir a ação que realizará para tal imagem. Assim, particularmente para a representação de imagens, o *Q-Learning* não possui a capacidade de generalizar experiências prévias sobre estados (imagens) nunca vistos, mas com uma distribuição de *pixels* muita parecida com imagens já processadas pelo algoritmo.

Assim, com o intuito de solucionar tais problemas, a equipe *DeepMind* utilizou uma estratégia a qual combina o uso de redes neurais convolucionais com o algoritmo *Q-Learning* de modo a aproximar a função Q ótima num método denominado *Deep Q-Learning* ou *Deep Q-Network* (DQN) [Mnih et al. 2015], um dos precursores do ramo Aprendizado Por Reforço Profundo - do inglês *Deep Reinforcement Learning* (DRL). Desta forma, não é mais necessário uma tabela para armazenar os estados possíveis. Tal método atingiu um desempenho sobre-humano em diversos jogos de Atari 2600 utilizando as imagens brutas (*pixels*) do jogo como representação dos estados.

4. Bônus

Lista de recursos que abrangem a área de Aprendizado por Reforço e Aprendizado Profundo por Reforço - *Deep Reinforcement Learning* (DRL). Ressalta-se que todos eles são gratuitos.

- ***Draft* do Livro *Reinforcement Learning: An Introduction* de *Sutton and Barto* - 2017**: `http://incompleteideas.net/book/bookdraft2017nov5.pdf`

- **Curso de DRL do *School of AI* - 2018**: `https://www.theschool.ai/courses/move-37-course/`

- **Curso de DRL da *UC Berkeley* - 2018**: `http://rail.eecs.berkeley.edu/deeprlcourse/`

- **Videoaulas sobre *Advanced Deep Learning and Reinforcement Learning* da *DeepMind* - 2018**: `https://www.youtube.com/playlist?list=`

PLqYmG7hTraZDNJre23vqCGIVpfZ_K2RZs

- ***Spinning Up in Deep RL* da *OpenAI* - 2018**: https://spinningup.openai.com/en/latest/index.html

Referências

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

Harmon, M. E. and Harmon, S. S. (1996). Reinforcement learning: A tutorial.

Irpan, A. (2015). Q-learning in continuous state action spaces. https://www.alexirpan.com/public/research/281areport.pdf.

Li, Y. (2017). Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274.

Limnios, N. and Oprişan, G. (2001). *Semi-Markov Processes and Reliability*. Birkhäuser Boston.

Martins, W., Afonseca, U. R., Nalin, L. E. G., and Gomes, V. M. (2007). Tutoriais inteligentes baseados em aprendizado por reforço: Concepção, implementação e avaliação empírica. *Simpósio Brasileiro de Informática na Educação - SBIE - Mackenzie*.

Mnih, V., Kavukcuoglu, K., Silver, D., and Rusu, A. A. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Neto, H. C. (2007). Ls-draughts: um sistema de aprendizagem de jogos de damas baseado em algoritmos genéticos, redes neurais e diferenças temporais. Master's thesis, Universidade Federal de Uberlândia.

Prati, R. (2011). Aprendizado por reforço (*Reinforcement Learning*). http://professor.ufabc.edu.br/~ronaldo.prati/InteligenciaArtificial/reinforcement-learning.pdf.

Robin, J. (2007). Slides - aprendizado por reforço. www.cin.ufpe.br/~compint/aulas-IAS/ias/ias-021/rl.ppt.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 993–1000, New York, NY, USA. ACM.

Tokic, M. and Palm, G. (2011). Value-difference based exploration: Adaptive control between epsilon-greedy and softmax. In Bach, J. and Edelkamp, S., editors, *KI 2011: Advances in Artificial Intelligence*, pages 335–346, Berlin, Heidelberg. Springer Berlin Heidelberg.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford.

Watkins, C. J. C. H. and Dayan, P. (1992). Technical note q-learning. *Machine Learning*, 8:279–292.