Aprendizado por reforço

Até o momento, tivemos contato com várias técnicas de aprendizado de máquina ligadas, essencialmente, a dois paradigmas: (*i*) supervisionado e (*ii*) não-supervisionado. No primeiro caso, o modelo aprende a partir de exemplos de entrada e saída $\{\mathbf{x}_i, \mathbf{y}_i\}, i = 1, \ldots, N$, isto é, com base na informação disponível em amostras rotuladas. No segundo caso, o modelo busca representar a estrutura subjacente aos dados não-rotulados de entrada.

Por sua vez, o aprendizado por reforço (RL, do inglês *reinforcement learning*) envolve a aquisição de conhecimento a partir da *interação* de um modelo tomador de decisões, chamado de *agente*, com o *ambiente* (SUTTON & BARTO, 2018; GÉRON, 2019). Em suma, o *agente* observa o estado do *ambiente* e realiza uma sequência de *ações*, recebendo em resposta sinais de *recompensa* ou de *punição* segundo a qualidade das

ações tomadas com respeito ao(s) objetivo(s) que se pretende atingir. O desafio em RL consiste em aprender como escolher as ações que maximizem a recompensa esperada ou acumulada ao longo do tempo.

Desta maneira, assim como ocorre nos dois tipos de aprendizado supracitados, é preciso resolver um problema de otimização em RL. Porém, diferentemente do aprendizado supervisionado, que trabalha com exemplos, o aprendizado em RL se dá com base na experiência.

RL é um dos mais antigos temas em *machine learning*, com início nos anos 50 (BELLMAN, 1957), mas, à semelhança de redes neurais, também experimentou uma pequena revolução em anos mais recentes. Em 2013, pesquisadores da *startup* britânica *DeepMind* apresentaram um modelo capaz de jogar qualquer jogo de Atari do zero, muitas vezes demonstrando habilidades sobre-humanas, usando apenas os

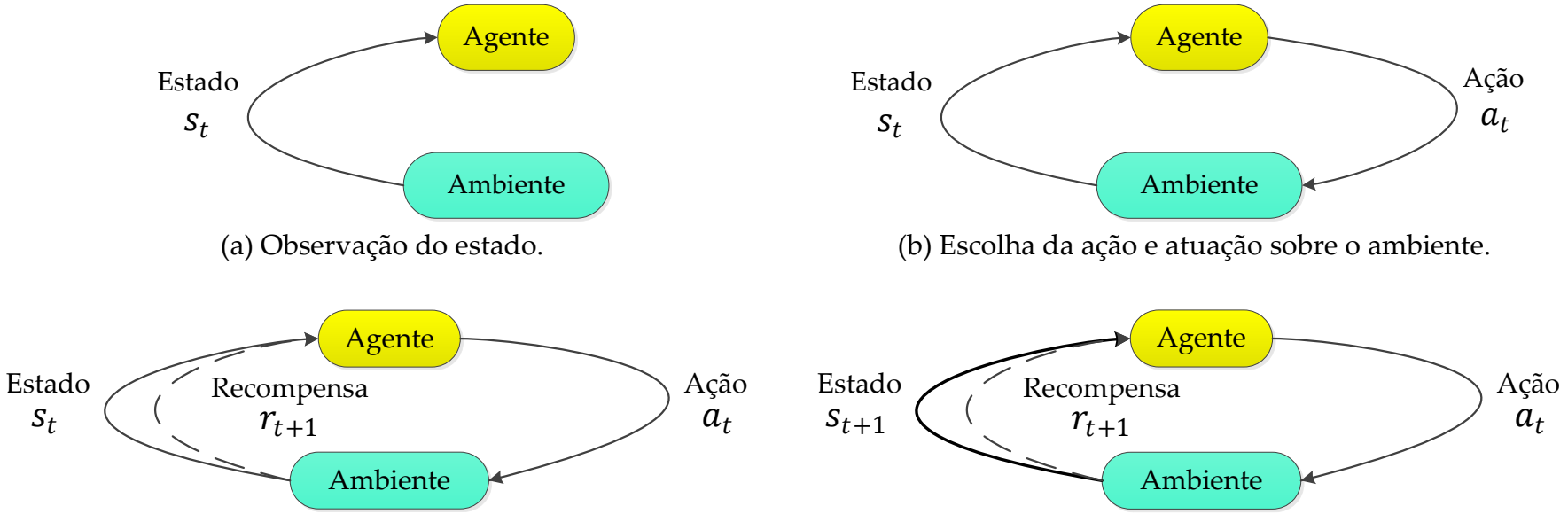
pixels brutos do cenário como entrada e sem qualquer conhecimento *a priori* de estratégias e das próprias regras do jogo (MNIH ET AL., 2013; MNIH ET AL., 2015).

Outro marco importante foi o desenvolvimento do *AlphaGo*, o qual, nos anos de 2016 e 2017, superou de forma acachapante jogadores mundialmente renomados do jogo *Go* (4-1 contra Lee Sedol, 3-0 contra Ke Jie).

1. Conceitos iniciais e motivação

A Figura 1 apresenta a interação típica entre o agente e o ambiente no contexto de aprendizado por reforço. Inicialmente, (a) o agente toma conhecimento do estado atual s_t do ambiente e, com base nesta informação, (b) ele define uma ação a ser realizada sobre o ambiente, o qual (c) reage a esta atuação, produzindo uma recompensa e (d) transita para um novo estado. Tanto a recompensa quanto o novo estado podem ter uma natureza estocástica, ou seja, podem estar associados a

distribuições de probabilidades. Este processo (a)-(d) se repete, de modo que o agente é forçado a tomar uma sequência de ações. Ao final, desejamos que a soma acumulada das recompensas recebidas seja máxima.



(a) Observação do estado.

(b) Escolha da ação e atuação sobre o ambiente.

(c) Realimentação do ambiente.

(d) Transição para novo estado.

Figura 1 – Resumo da interação entre agente e ambiente no contexto de RL.

O cenário que acabamos de descrever é bastante amplo, podendo englobar várias possibilidades:

- Ambiente: total ou parcialmente observável.
- Agente: único ou múltiplos agentes.
- Ambiente / Ações: determinístico(as) ou estocástico(as).
- Ambiente: estático ou dinâmico.
- Ambiente / Ações: discreto(as) ou contínuo(as).

Com respeito ao agente tomador de decisão, dois conceitos importantes que permeiam o aprendizado por reforço são: *política* (*policy*) e *funções de qualidade* (*value functions*) (SUTTON & BARTO, 2018).

A *política* de um agente descreve o modo pelo qual ele escolhe a ação a partir da observação do ambiente. Em essência, a política pode ser vista como uma função que mapeia o estado atual do ambiente, ou um conjunto de observações e de ações passadas, na ação a ser tomada.

Uma perspectiva interessante que tem sido explorada em RL consiste em representar a política de um agente através de uma rede neural (profunda) e, então, fazer explicitamente uma busca pela melhor política através do ajuste dos parâmetros da rede. Esta abordagem está associada à ideia de gradiente de política (*policy gradient*), e será discutida mais adiante.

Por sua vez, as *funções de qualidade* permitem que o agente avalie o estado do ambiente e as ações, ou, mais usualmente, o par estado-ação, e servem para guiar a política do agente.

O agente também pode adotar um *modelo de mundo*, *i.e.*, uma representação de como o ambiente funciona. Às vezes, o modelo de mundo pode estar implícito na política e nas funções de qualidade (*e.g.*, através de restrições a determinadas ações dependendo do estado em que o ambiente se encontra).

Um ponto essencial em RL é que **a melhor estratégia para o agente é sempre tomar a ação que maximiza a recompensa futura acumulada**. Em outras palavras, dentre todas as ações possíveis em um determinado estado, o agente deve escolher aquela ação que ao ser tomada leve o sistema a um novo estado tal que, dali em diante, seguindo a mesma política, produza a máxima recompensa acumulada.

1.1. Taxonomia de RL

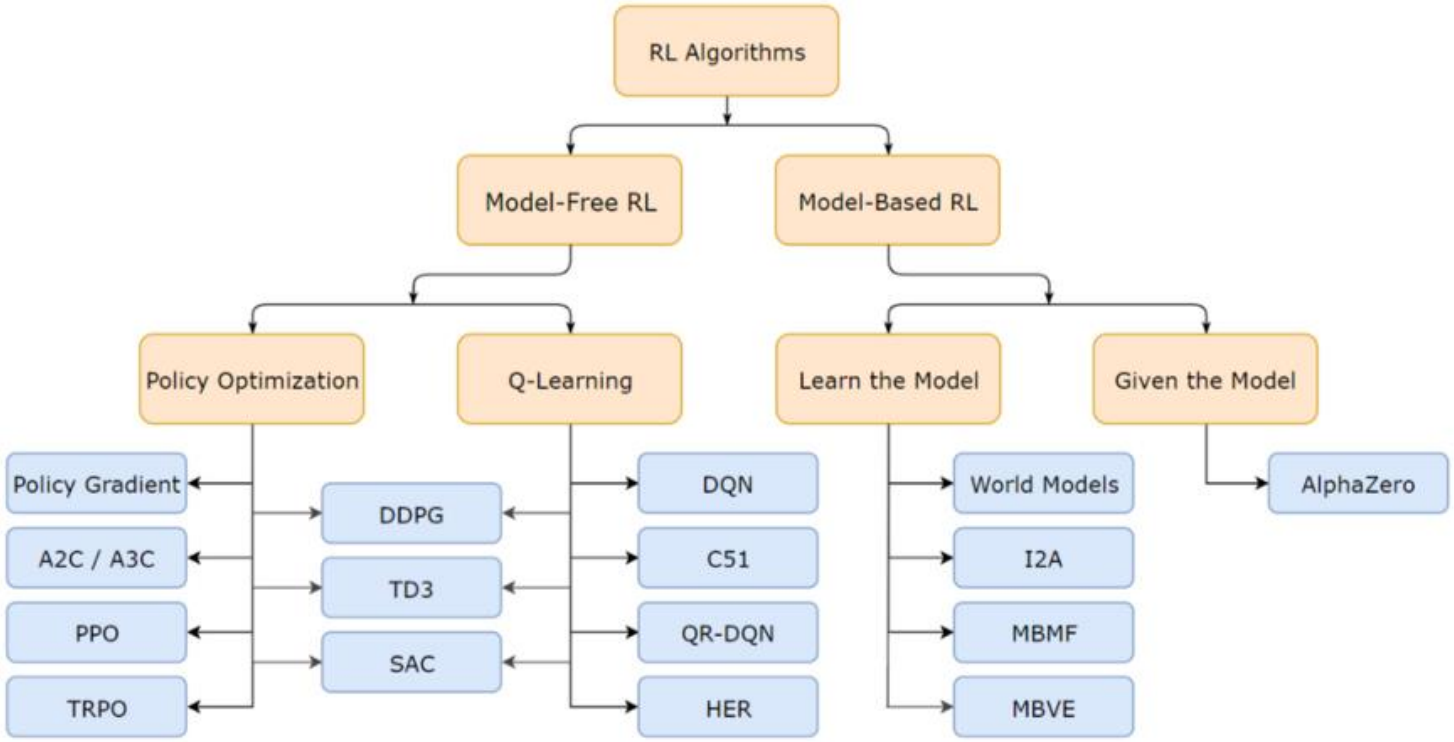


Figura 2 – Taxonomia das técnicas de aprendizado por reforço. Uma distinção básica é feita entre os métodos que se baseiam em um modelo explícito de mundo (*model-based* RL), e aqueles que não requerem esta informação (*model-free* RL). Extraída de https://spinningup.openai.com/.

1.2. Exemplos de aplicação

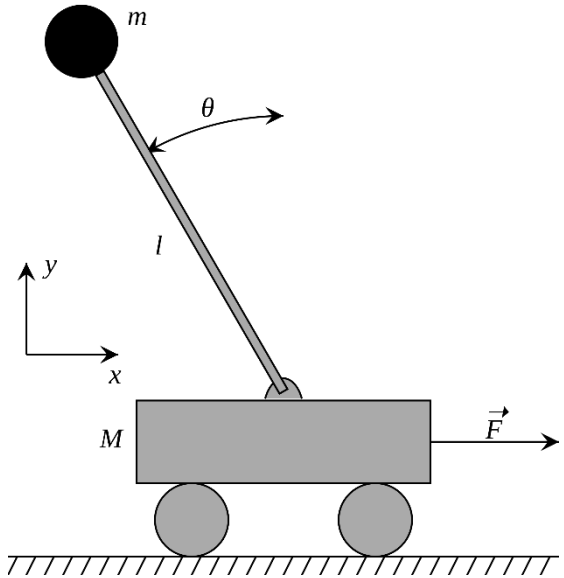
1.2.1. *Cart-pole*

Objetivo: equilibrar o pêndulo sobre uma massa móvel.

Estado: ângulo e velocidade angular do pêndulo, posição e velocidade horizontal do carro.

Ação: força horizontal aplicada sobre o carro.

Recompensa: 1 unidade para cada instante de tempo que o pêndulo se mantém em pé.



1.2.2. Jogos de Atari

Objetivo: completar o jogo com a máxima pontuação.

Estado: *pixels* de imagens do cenário do jogo.

Ação: possíveis comandos no jogo (*e.g.*, mover para cima, mover para baixo etc.); dependendo do jogo, há de duas a dezoito ações possíveis.

Recompensa: aumento ou redução da pontuação a cada instante de tempo.

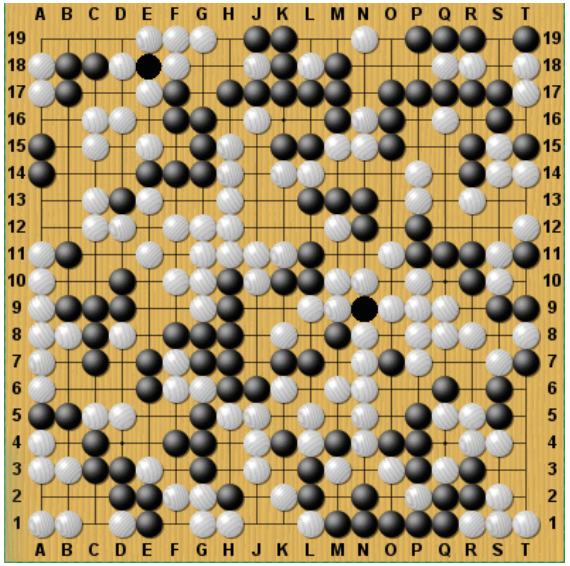
1.2.3. *Go*

Objetivo: vencer o jogo.

Estado: posição de todas as peças.

Ação: onde colocar a próxima peça.

Recompensa: 1, se venceu a partida; 0, caso contrário.



Curiosidades:

- O número de configurações possíveis para o tabuleiro padrão de *Go* (19 × 19) é da ordem de 10^{170} (TROMP & FARNEBÄCK, 2016).
- Algumas das técnicas mais poderosas para jogos como *Go*, xadrez e *shogi* aprenderam jogando contra si próprias; assim, o modelo aprimora as suas habilidades a partir das duas experiências (vitória ou derrota).

1.2.4. *StarCraft II*

Desafios:

- Informação imprecisa / incompleta do ambiente.
- Tomada de decisão em tempo real.
- Planejamento de longo prazo.

Outro grande obstáculo é o fato de termos um gigantesco espaço de soluções candidatas (ONTAÑÓN ET AL., 2013).

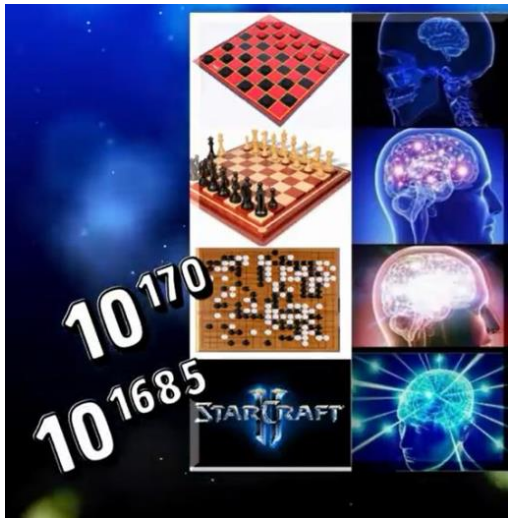


Figura 3 – Número de possíveis estados nos jogos Go e StarCraft 2. Extraída de https://www.youtube.com/watch?v=98V6PnwVXCc.

Figura 4 – Problemas com complexidade crescente sendo abordados com RL. Extraído de https://www.youtube.com/watch?v=cUTMhmVh1qs.

2. Formalização matemática

O aprendizado por reforço está intimamente ligado às áreas de programação dinâmica, controle ótimo e teoria de jogos, sendo uma formulação geral para o problema de tomada de decisão sequencial.

A base matemática para RL está associada à noção de processo de decisão de Markov (MDP, do inglês *Markov decision process*) (BELLMAN, 1957; SUTTON & BARTO, 2018).

2.1. Processo de decisão de Markov

Um MDP é definido pela tupla $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, em que:

- $\mathcal{S}$ é o conjunto de possíveis estados;
- $\mathcal{A}$ é o conjunto de possíveis ações;
- $\mathcal{R}$ é a distribuição de recompensa dado o par estado-ação;
- $\mathcal{P}$ é a distribuição de probabilidade de transição;

- γ é o fator de desconto.

A Figura 5 ilustra um processo de decisão de Markov simples.

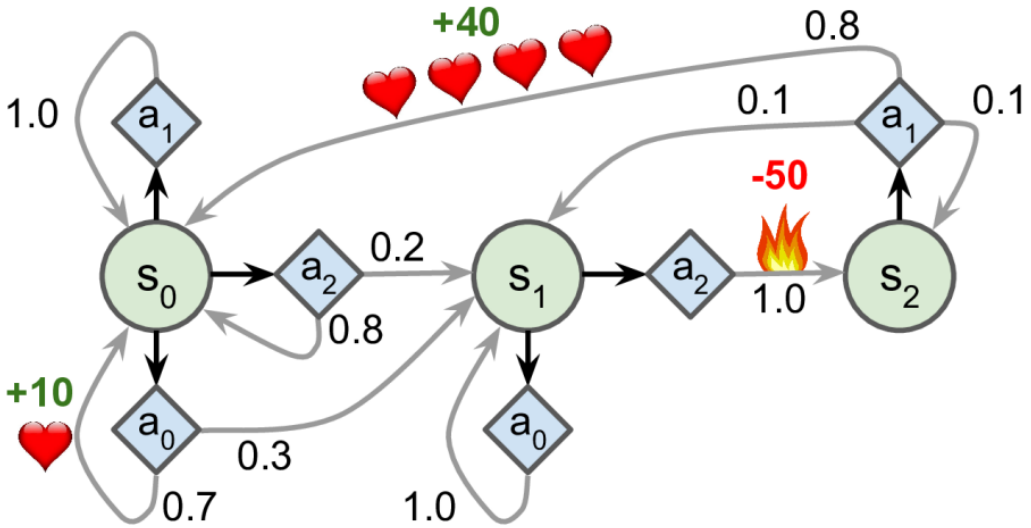


Figura 5 – Exemplo de um MDP. Extraída de (GÉRON, 2019).

Neste caso, o MDP possui três estados (s_0, s_1, s_2) e até três ações possíveis em cada passo (a_0, a_1, a_2). As recompensas ou punições estão associadas às possíveis transições de estados a partir da tomada de uma ação.

Por exemplo, se o sistema está no estado s_2, a única ação possível é a_1, a qual, com maior probabilidade, levará o sistema ao estado s_0, trazendo uma recompensa de

+40. Contudo, é possível que a ação a_1 leve o sistema ao estado s_1 ou, ainda, que o sistema permaneça no mesmo estado. Nestes dois casos, não há recompensa.

De forma geral, podemos descrever o comportamento de um MDP através dos seguintes passos:

Algoritmo 1 – Operação de um MDP.
• Em $t = 0$, o ambiente amostra o estado inicial $s_0 \sim p(s_0)$ • Partindo de $t = 0$, faça: ➢ O agente seleciona uma ação a_t ➢ O ambiente reage com uma recompensa $r_{t+1} \sim \mathcal{R}(\cdot \mid s_t, a_t)$ ➢ O ambiente migra para o próximo estado $s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t)$ ➢ O agente recebe a recompensa r_{t+1} e observa o novo estado s_{t+1} ➢ $t = t + 1$

2.2. Política, retorno esperado e política ótima

Formalmente, uma *política* é um mapeamento dos estados nas probabilidades de o agente escolher cada ação possível. Por exemplo, se o agente está seguindo a

política π no instante t, então $\pi(a|s)$ é a probabilidade de a ação ser $a_t = a$ considerando que o ambiente se encontra no estado $s_t = s$. Ou seja, $\pi(a|s)$ define a distribuição de probabilidade em $a \in \mathcal{A}(s)$ para cada $s \in \mathcal{S}$.

As abordagens de aprendizado por reforço que lidam com um modelo explícito de política e o modificam com base em sua experiência são denominadas *on-policy*.

A adoção de uma política por parte do agente dá origem a uma trajetória na forma:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, \ldots$$

Em RL, a única informação disponível para guiar o aprendizado do agente é a recompensa. Contudo, as recompensas são tipicamente esparsas e atrasadas. Por exemplo, no caso do sistema *cart-pole*, é possível que o agente tenha conseguido manter o pêndulo equilibrado por 100 iterações consecutivas até cair. O valor da recompensa acumulada pode até ser elevado, mas isso não necessariamente significa que todas as ações tomadas dentro da sequência foram boas. Logo, como é

possível saber quais das 100 ações foram boas, e quais foram ruins? Esta questão define o problema de *credit assignment*.

Objetivo: encontrar a política ótima π^* que maximize o retorno acumulado:

$$g_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (1)$$

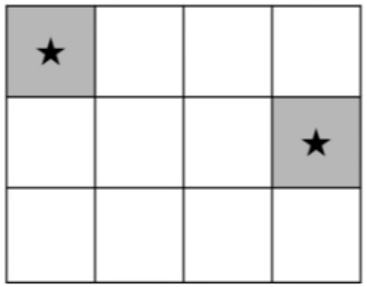
Note que as recompensas são ponderadas de modo a dar maior peso aos eventos mais próximos.

Para lidar com a aleatoriedade presente no ambiente e no modelo (estado inicial, probabilidades de transição etc.), o que se busca é maximizar o valor esperado do retorno acumulado:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\pi \right\} \quad (2)$$

Vejamos a seguir um exemplo clássico de um MDP e do conceito de política.

Grid world:

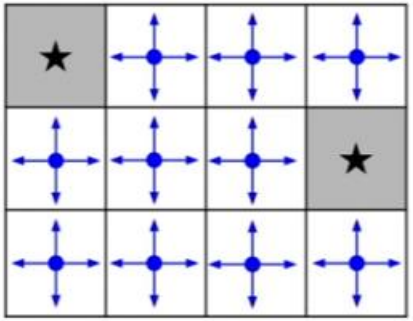


Objetivo: chegar em um dos estados terminais no menor número de movimentos.

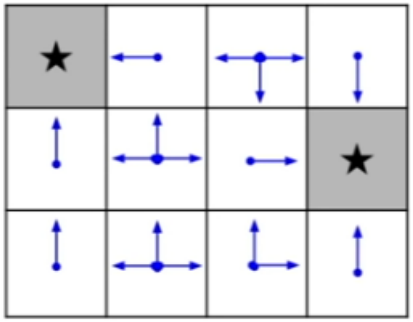
Cada movimento implica uma recompensa negativa (e.g., $r = -1$).

Ações: mover-se para cima, para baixo, para a esquerda ou para a direita.

A Figura 6 apresenta uma política aleatória e a política ótima para este cenário.



(a) Política aleatória



(b) Política ótima

Figura 6 – Exemplos de política para movimentação no *grid world*.

2.3. Funções de qualidade

Quão bom é um estado?

A função valor (*value function*) em um estado s indica o valor esperado do retorno acumulado se seguirmos a política π a partir do estado s:

$$v_\pi(s) = \mathbb{E}\{g_t | s_t = s, \pi\} = \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \, | s_t = s, \pi\right\} \tag{3}$$

Quão bom é um par estado-ação?

A função Q-valor no estado s e ação a corresponde ao valor esperado do retorno acumulado quando a ação a é tomada a partir do estado s e, dali em diante, seguimos a política π:

$$Q_\pi(s,a) = \mathbb{E}\{g_t | s_t = s, a_t = a, \pi\} = \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \, | s_t = s, a_t = a, \pi\right\} \tag{4}$$

Uma *política ótima* π^* está associada à função valor ótima e à função Q-valor ótima, conforme as expressões:

$$v^*(s) = \max_{\pi} v_\pi(s)$$
$$Q^*(s,a) = \max_{\pi} Q_\pi(s,a), \tag{5}$$

para todo $s \in \mathcal{S}$ e $a \in \mathcal{A}$. Observe que $Q^*(s,a)$ indica o retorno esperado ao se tomar a ação a no estado s e, dali em diante, seguir a política ótima.

2.4. Otimalidade de Bellman

> *"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision"* (BELLMAN, 1957).

Na ciência da computação, um problema que pode ser decomposto desta maneira é dito ter uma *subestrutura ótima*.

Interessantemente, o problema envolvido em RL admite o princípio da otimalidade de Bellman, o que leva a algumas equações de consistência que devem ser satisfeitas pela política ótima, ou, analogamente, pelas funções ótimas de qualidade (SUTTON & BARTO, 2018).

Equação de Bellman para a função Q-valor:

$$Q^*(s,a) = \mathbb{E}\left\{ r_{t+1} + \gamma \underbrace{\max_{a'} Q^*(s_{t+1}, a')}_{v^*(s_{t+1})} \,|\, s_t = s, a_t = a \right\} \tag{6}$$

$$= \sum_{s',r} \mathcal{P}(s'|s,a)\left[r + \gamma \max_{a'} Q^*(s', a') \right]$$

Dado qualquer par estado-ação (s,a), o Q-valor ótimo deste par será a recompensa r_{t+1} que recebemos por adotar esta ação, mais o valor do estado seguinte (s_{t+1}), qualquer que seja ele. Uma vez que, por hipótese, conhecemos a melhor política, sabemos que a melhor decisão também será tomada a partir do próximo estado s_{t+1}.

Logo, dentre todas as ações possíveis a' que podemos tomar a partir de s_{t+1}, será escolhida aquela que maximiza $Q^*(s', a')$.

2.5. Considerações

É importante trazer de volta à mente o problema original que pretendemos resolver em RL: desejamos determinar a melhor política para o agente, a qual descreve de forma ótima como ele deve agir considerando todos os possíveis estados do sistema. O critério utilizado para guiar a busca pela política ótima corresponde à maximização do retorno esperado.

Embora exista uma grande variedade de técnicas em RL, conforme mostrado na Figura 2, vamos destacar aqui duas abordagens: gradiente de política e *Q-learning*. No primeiro caso, temos um modelo paramétrico para a política do agente (*e.g.*, dado por uma rede neural) e realizamos uma busca pelos parâmetros da política que maximizem o retorno esperado.

A segunda possibilidade, que será descrita na próxima seção, consiste em explorar a equação de Bellman, definida em (6), para encontrar a função Q-valor ótima, a partir da qual é possível obter a política ótima de maneira relativamente simples.

3. *Q-learning* e *Deep Q-learning*

Considere que os Q-valores ótimos sejam conhecidos para todo par estado-ação. Então, a obtenção de uma política ótima é imediata: se o sistema está no estado s, o agente deve escolher a ação com o maior Q-valor a partir deste estado (IVANOV & D'YAKONOV, 2019):

$$\pi^*(s) = \arg \max_{a \in \mathcal{A}(s)} Q^*(s, a) \tag{7}$$

Este resultado abre uma perspectiva de trabalho interessante: em vez de buscar diretamente uma política ótima π^*, o agente pode buscar os Q-valores ótimos e, então, derivar sua política a partir deles. Esta perspectiva é conhecida como *Q-learning* (SUTTON & BARTO, 2018).

De forma resumida, Q-*learning* se apoia sobre duas ideias principais:

- Explorar a equação de Bellman, dada em (6), como uma regra de atualização iterativa dos Q-valores. Um exemplo desta ideia, combinado com a técnica de diferença temporal (TD, do inglês *temporal difference*), dá origem à seguinte expressão (SUTTON & BARTO, 2018):

$$Q_{t+1}^{*}(s,a) = Q_{t}^{*}(s,a) + \alpha_t \left[r_{t+1} + \gamma \max_{a'} Q_{t}^{*}(s_{t+1}, a') - Q_{t}^{*}(s,a) \right]$$

- Adotar uma política de exploração do MDP subjacente durante o aprendizado do agente, uma vez que ainda não temos os Q-valores ótimos.

  Uma opção bastante usual é chamada de ϵ-*greedy* (SUTTON & BARTO, 2018) e funciona da seguinte forma: em cada passo, o agente pode adotar uma ação aleatória com probabilidade ϵ, ou, então, adotar a ação com maior Q-valor com probabilidade $1 - \epsilon$.

O principal problema associado a Q-*learning* é que ele não é escalável: para MDPs de médio e grande porte, há um crescimento explosivo do número de pares estado-ação, de modo que ajustar $Q^*(s,a)$ se torna computacionalmente infactível.

Uma possível solução consiste em estimar os Q-valores com o auxílio de um aproximador de funções:

$$Q(s,a,\boldsymbol{\theta}) \approx Q^*(s,a)$$

Sendo assim, por que não usar o poder computacional das redes neurais profundas para realizar esta aproximação? Esta ideia está no cerne do ramo conhecido como *deep* Q*-learning* (MNIH ET AL., 2015; IVANOV & D'YAKONOV, 2019).

3.1. *Deep Q-learning*

Nesta vertente, uma rede neural profunda é empregada para estimar os Q-valores referentes a cada par estado-ação (a qual, por isso, é chamada de *deep* Q*-network*, DQN). Interessantemente, isto é feito através de uma única rede que recebe como

entrada o estado do ambiente e produz saídas, uma para cada possível ação, que indicam as estimativas dos Q-valores.

Mas, como a DQN é treinada?

Considere o Q-valor aproximado fornecido pela DQN para o par (s, a), $Q(s, a; \boldsymbol{\theta})$. Graças à equação de Bellman, sabemos que $Q(s, a; \boldsymbol{\theta})$ deve ser tão próximo quanto possível da soma entre a recompensa que é observada após adotar a ação a no estado s, denotada por r, mais o valor do retorno acumulado ao jogar otimamente a partir do estado seguinte, qualquer que seja ele.

Para estimar o retorno acumulado, podemos executar a DQN sobre o próximo estado s' para obter os Q-valores $Q(s', a'; \boldsymbol{\theta})$ para todas as possíveis ações a', e, assim, selecionar o máximo Q-valor. Portanto, a referência para $Q(s, a; \boldsymbol{\theta})$ pode ser escrita como:

$$y_t = r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}_t)$$

Sendo assim, podemos definir uma função custo para o treinamento supervisionado da DQN que force $Q(s, a, \boldsymbol{\theta})$ a se tornar cada vez mais parecido com y_t, como, por exemplo, o erro quadrático.

Com isso, ajustando os parâmetros $\boldsymbol{\theta}$ da rede neural, conduzimos a aproximação da função Q-valor na direção de cumprir a condição estabelecida na equação de Bellman, a qual está atrelada à política ótima.

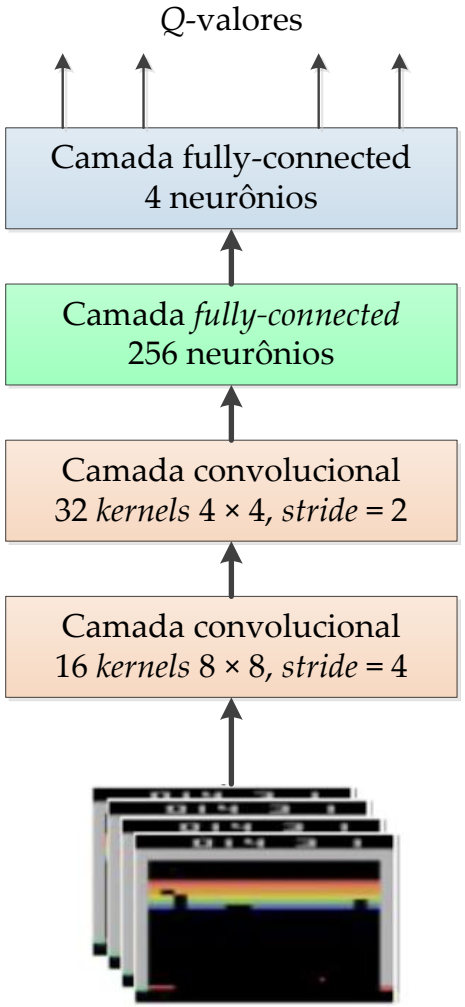
Passo *forward*:

$$\begin{aligned} L_i(\boldsymbol{\theta}_i) &= \mathbb{E}_{s,a\sim\rho(\cdot)}\left\{\left(y_i - Q(s, a; \boldsymbol{\theta}_i)\right)^2\right\} \\ y_i &= \mathbb{E}_{s'\sim\varepsilon}\left\{r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}_{i-1})\right\} \end{aligned} \tag{8}$$

Passo *backward*:

$$\nabla_{\boldsymbol{\theta}_i} L_i(\boldsymbol{\theta}_i) = \mathbb{E}_{s,a\sim\rho(\cdot),s'\sim\varepsilon}\left\{\left(r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}_{i-1}) - Q(s, a; \boldsymbol{\theta}_i)\right) \nabla_{\boldsymbol{\theta}_i} Q(s, a; \boldsymbol{\theta}_i)\right\} \tag{9}$$

Exemplo: *deep Q-learning* para jogar Atari.



Estado: 84 × 84 × 4 - últimos quatro frames
(Após conversão de RGB para escala de
cinza, subamostragem e cropping)

Figura 7 – Arquitetura da DQN utilizada para jogar Atari (MNIH ET AL., 2013).

Um problema em se realizar o aprendizado com base em amostras sequenciais é que elas são altamente correlacionadas. No fundo, cada decisão tomada pelo agente condiciona o estado observado na sequência, limitando as possíveis experiências subsequentes. Contudo, é necessário que o agente explore um conjunto rico de circunstâncias do ambiente para que a DQN possa aproximar bem os Q-valores. Além disso, também existe a possibilidade de o agente "esquecer" o que aprendeu anteriormente por causa das experiências (correlacionadas) que ele passou a vivenciar mais recentemente. Neste contexto, uma técnica chamada de *experience replay* (MNIH ET AL., 2013) pode auxiliar o treinamento da DQN.

Em suma, *experience replay* envolve o armazenamento contínuo das transições (s, a, r, s') experimentadas pelo agente durante os vários episódios do jogo. Então, em cada iteração, um *mini-batch* aleatório destas transições é amostrado, servindo de base para o ajuste dos parâmetros da rede.

O algoritmo mostrado na Figura 8 traz a sequência de passos utilizada em (MNIH ET AL., 2013) para o treinamento da DQN no âmbito de jogos de Atari.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ } *ϵ-greedy*
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Equivalente à Equação (9)

Figura 8 – Algoritmo de RL baseado em *deep Q-learning*. Extraída de (Mnih et al., 2013).

4. Gradiente de política

Nesta abordagem, o agente adota um modelo explícito de política $\pi_{\boldsymbol{\theta}}$, com parâmetros $\boldsymbol{\theta}$ (*e.g.*, na forma de uma rede neural), o qual propõe ações a partir dos estados. O objetivo é encontrar os parâmetros $\boldsymbol{\theta}$ que levem o agente a receber o máximo retorno acumulado:

$$\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}\sim\pi_{\boldsymbol{\theta}}} \left\{ \sum_{t=1} \gamma^{t-1} r_t \right\}, \tag{10}$$

onde $\mathcal{T}$ denota a trajetória realizada pelo agente durante o episódio seguindo a política $\pi_{\boldsymbol{\theta}}$.

À semelhança do treinamento clássico de redes neurais, o problema em (10) pode ser abordado com o auxílio de métodos baseados em gradiente (*e.g.*, gradiente ascendente estocástico).

Intuitivamente, com base nas recompensas acumuladas, o algoritmo deve penalizar aquelas ações que contribuíram para o insucesso do agente, reduzindo suas probabilidades, enquanto as ações promissoras devem ser reforçadas, experimentando um aumento de sua probabilidade.

Observações:

- Explorando alguns truques algébricos, é possível escrever o gradiente da função objetivo em (10) da seguinte forma (IVANOV & D'YAKONOV, 2019):

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\{\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi}(s,a)\}$$

  Métodos como o REINFORCE trabalham, então, com uma aproximação estocástica da esperança e com a substituição da função Q-valor pelo valor do retorno médio acumulado em alguns episódios.

- Uma vez que não é possível experimentar a política atual exaustivamente, temos apenas uma aproximação de sua real qualidade. Por isso, o cálculo do

gradiente apresenta uma alta variância. Existem técnicas que tentam reduzir esta variância para estabilizar o algoritmo (SUTTON & BARTO, 2018; IVANOV & D'YAKONOV, 2019).

5. *Actor-Critic*

Algoritmos de RL do tipo *actor-critic* exploram conjuntamente as ideias de gradiente de política e de *Q-learning* (SUTTON & BARTO, 2018; IVANOV & D'YAKONOV, 2019). Em suma, uma rede neural, denominada *actor* (ator), é responsável pela política, *i.e.*, por decidir qual ação deve ser tomada para cada estado. Uma segunda rede neural, chamada de *critic* (crítico), avalia a qualidade das decisões do ator através da estimação da função valor (ou da função Q-valor). Desta maneira, o ajuste aplicado sobre o ator se baseia na informação de qualidade fornecida pelo crítico, o qual, por sua vez, é adaptado seguindo o raciocínio de *Q-learning*. Interessantemente, a tarefa

do crítico é facilitada uma vez que ele deve estimar os Q-valores somente dos pares ação-estado gerados pela política.

6. Referências bibliográficas

BELLMAN, R., "A Markovian Decision Process", *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679-684, 1957.

BOTVINICK, M., WANG, J. X., MILLER, K. J., KURTH-NELSON, Z., "Deep Reinforcement Learning and its Neuroscientific Implications", *arXiv:2007.03750*, 2020.

BUSONIU. L., BABUSKA, R., DE SCHUTTER, B., ERNST, D., **Reinforcement learning and dynamic programming using function approximators**, CRC Press, 2010.

FRANÇOIS-LAVET, V., HENDERSON, P, ISLAM, R., BELLEMARE, M. G., PINEAU, J., "An Introduction to Deep Reinforcement Learning", *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp 219-354, 2018.

GÉRON, A., **Hands-on Machine Learning with Scikit-Learn, Keras & Tensorflow**, O'Reilly Media, 2ª ed., 2019.

IVANOV, S., D'YAKONOV, A., "Modern Deep Reinforcement Learning Algorithms", *arXiv:1906.10025*, 2019.

MNIH, V., KAVUKCUOGLU, K., SILVER, D. ET AL., "Human-level control through deep reinforcement learning". *Nature*, 518, pp. 529–533, 2015.

MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., RIEDMILLER, M, "Playing Atari with Deep Reinforcement Learning", *arXiv:1312.5602*, 2013.

ONTAÑÓN, S., SYNNAEVE, G., URIARTE, A., RICHOUX, F., CHURCHILL, D., PREUSS, M., "A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 4, pp. 293-311, 2013.

SUTTON, R. S., BARTO, A. G., **Reinforcement Learning: An Introduction**, The MIT Press, 2ª ed., 2018.

TROMP, J., FARNEBÄCK, G., "Combinatorics of Go", Disponível em https://tromp.github.io/go/gostate.pdf, 2016.

VON ZUBEN, F. J., **Notas de Aulas do Curso "Redes Neurais" (IA353),** disponíveis em *http://www.dca.fee.unicamp.br/~vonzuben/courses/ia353.html*