



REINFORCEMENT LEARNING

UNIDADE II **TOMADA DE DECISÕES COMPLEXAS**

Elaboração

Natasha Sophie Pereira

Atualização

Bruno Iran Ferreira Maciel

Produção

Equipe Técnica de Avaliação, Revisão Linguística e Editoração

SUMÁRIO

UNIDADE II

TOMADA DE DECISÕES COMPLEXAS.....	5
-----------------------------------	---

CAPÍTULO 1

O AMBIENTE: PROCESSO DE DECISÃO DE MARKOV	6
---	---

CAPÍTULO 2

ALGORITMOS PARA SOLUÇÃO DE MDPS	19
---------------------------------------	----

CAPÍTULO 3

PROCESSOS DE DECISÃO DE MARKOV PARCIALMENTE OBSERVÁVEIS	35
---	----

REFERÊNCIAS	40
-------------------	----

TOMADA DE DECISÕES COMPLEXAS

UNIDADE II

Nesta unidade, abordaremos as questões computacionais envolvidas na tomada de decisões em diferentes ambientes estocásticos. Enquanto na unidade anterior a preocupação estava com problemas de decisão instantânea, em que a utilidade do resultado de cada ação era bem conhecida, aqui a preocupação é com problemas de decisão sequencial, em que a utilidade do agente depende de uma sequência de decisões. Problemas de decisão sequencial incorporam utilidades, incerteza e percepção, e incluem os problemas de busca e planejamento como casos especiais. O capítulo 1, desta unidade explica como os problemas de decisão sequencial são definidos e como eles podem ser resolvidos para gerar um comportamento ótimo que equilibre os riscos e as recompensas de agir em um ambiente incerto. O capítulo 2 desta unidade estende essas ideias ao caso de ambientes parcialmente observáveis e trata a questão algorítmica para agentes de teoria da decisão em ambientes parcialmente observáveis.

CAPÍTULO 1

O AMBIENTE: PROCESSO DE DECISÃO DE MARKOV

Como vimos na Unidade I, problemas de RL envolvem dois elementos, o **ambiente** e o **agente**. A cada instante de tempo, o ambiente se encontra em um estado cuja evolução é condicionada pelas ações que o agente toma. O agente, por sua vez, é capaz de ler, a cada momento, o estado do ambiente, e responder com uma **ação**. Após cada ação, o agente recebe um sinal de **recompensa** do ambiente. O objetivo do agente é maximizar a recompensa acumulada ao longo do tempo.

A separação entre **agente** e **ambiente** é flexível, e depende da formulação do problema: o agente pode ser, por exemplo, nosso corpo, recebendo sinais físicos através dos sentidos, ou, então, apenas o cérebro, recebendo sinais eletroquímicos do resto do corpo através dos neurônios. Ou, ainda, o agente pode ser um grupo de neurônios interagindo com o resto do cérebro. Por isso, o fato de a recompensa ser externa ao agente não significa que ela venha de fora do corpo humano, nesse caso.

Note também que determinada ação executada pelo agente pode levar a períodos intermediários de recompensa negativa, para só mais tarde trazer bons resultados. Nesse sentido, o agente deve ser capaz de realizar um planejamento visando não apenas uma boa recompensa imediata, mas uma boa recompensa acumulada em longo prazo. Em outras palavras, o agente não deve ser guloso em relação à recompensa imediata.

Cenários como esses podem ser representados utilizando os processos de decisão de Markov, usados para representar situações e que é necessário executar ações em sequência em ambientes com incerteza. Um cenário que pode ser modelado como um processo de decisão de Markov abrange um sistema com vários estados, assim como ações que provavelmente modifiquem o estado do sistema e a probabilidade de perceber – mesmo que indiretamente – o resultado de cada ação executada. Conhecida a descrição do problema, resolvê-lo significa encontrar uma política ótima que determine a cada momento que ação tomar para maximizar a recompensa esperada (ou minimizar o custo esperado).

Os elementos delineados até agora podem ser definidos de maneira mais formal: o ambiente é visto como um processo de decisão de Markov, e o agente expressa-se em uma política que associa estados do ambiente com probabilidades de executar cada uma das ações pertencentes a um conjunto de ações possíveis.

Um *Markov Decision Process* (MDP ou Processo de Decisão de Markov) é uma maneira de construir processos em que as transições entre estados são probabilísticas, sendo possível observar em que estado o processo está e interferir no processo de tempos

em tempos, executando ações (HAUSKRECHT, 1997; PUTERMAN, 1994). Ações têm recompensas – ou custos – que decorrem do estado em que o processo se encontra. Alternativamente, recompensas podem ser definidas apenas por estado, sem que essas dependam da ação executada. Chamamos de “Markov” (ou “Markovianos”) os processos que foram construídos obedecendo à propriedade de Markov.

O resultado de uma ação em um estado está sujeito à ação e ao estado atual do sistema (e não como o processo chegou a este estado), e são chamados de “processos de decisão” porque modelam a possibilidade de uma agente (ou “tomador de decisões”) interferir periodicamente no sistema executando ações, diferentemente de **Cadeias de Markov**, em que não se trata de como interferir no processo.

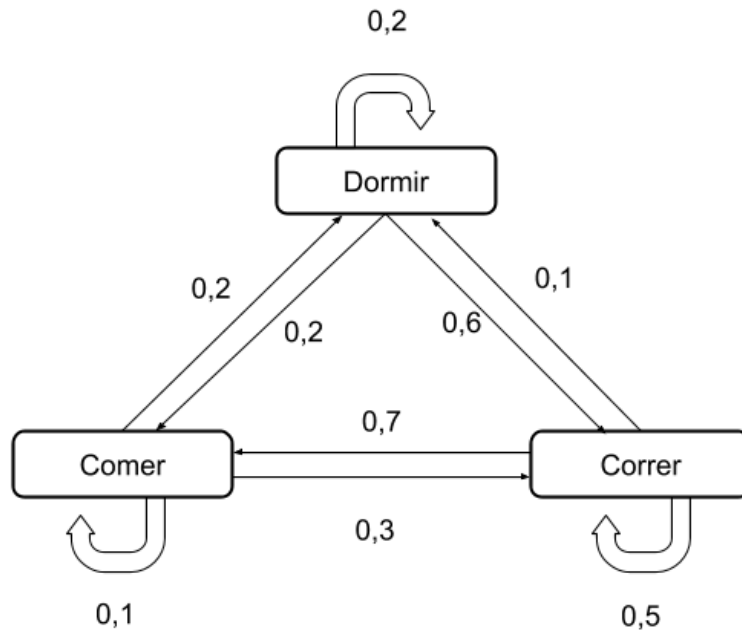
Em resumo: um problema de decisão sequencial para um ambiente completamente observável, estocástico, com um modelo de transição de Markov e recompensas aditivas, é chamado de processo de decisão de Markov ou MDP (*Markov Decision Process*), e consiste de um conjunto de estados (com estado inicial s_0); um conjunto de AÇÕES(s) de ações aplicáveis em cada estado; um modelo de transição e uma função de recompensa $R(s)$.

Vejamos essa definição com outros olhos, se for afirmado que: “Dado o presente, o futuro independe do passado”. É possível refletir e aplicar o conceito de Cadeias de Markov de maneira mais leve de se entender, imagine que um estado seguirá essa propriedade caso todos os estados anteriores a ele não influenciem na decisão do próximo estado, o único que influencia é apenas o estado atual – seria dentro de um exemplo médico, dizer que um paciente não tem histórico de doenças, caso ele esteja doente, só conta o que ele está sentindo hoje. Mas vale lembrar que não são todos os estados que obedecem à Propriedade de Markov, mas caso o faça, o problema torna-se ainda mais simples.

1.1. Exemplo da Cadeia de Markov

Vamos considerar que eu tenha algumas atividades completamente independentes para realizar. Com elas eu consigo identificar também os graus de probabilidade de ocorrência de uma, dada à outra, como é visto na Figura 20.

Figura 20. Exemplo da Cadeia de Markov simplificado.



Fonte: Elaborado pelo autor.

A ideia é simples. Vamos analisar as probabilidades de as ações continuarem ou serem revertidas em outras atividades. No caso, se eu estou avaliando minhas três ações, particularmente independentes, mas que podem levar uma à outra. Meu conjunto de ações é:

$$S = \{\text{Comer}, \text{Dormir}, \text{Correr}\}$$

As probabilidades associadas, no exemplo, mostram que a correlação probabilística entre as ações é a seguinte:

- » $P(\text{Correr}) = 0,5;$
- » $P(\text{Comer}) = 0,1;$
- » $P(\text{Dormir}) = 0,2;$
- » $P(\text{Comer} \mid \text{Dormir}) = 0,2;$
- » $P(\text{Dormir} \mid \text{Comer}) = 0,2;$
- » $P(\text{Correr} \mid \text{Dormir}) = 0,1;$
- » $P(\text{Dormir} \mid \text{Correr}) = 0,6;$
- » $P(\text{Comer} \mid \text{Correr}) = 0,3;$
- » $P(\text{Correr} \mid \text{Comer}) = 0,7;$

Então, se nosso agente estiver correndo, a probabilidade de que ele continue na mesma ação é de 50%. Se ele está comendo, a probabilidade de que ele continue comendo é de 10%. Assim como a probabilidade de ele estar correndo e isso influenciar na próxima ação dele sendo dormir é de apenas 10%, até considerando a situação fica bem intuitivo imaginar que o agente não vai comumente sair de uma corrida e dormir em seguida; fisiologicamente, o corpo precisa de uma atividade de descanso – por exemplo, entre elas. Da mesma forma que é bem provável que após a corrida, o agente vá comer, no nosso exemplo, 70% de chance.



Uma cadeia de Markov pode ser assim descrita: seja $S = \{s_1, s_2, \dots, s_r\}$ um conjunto de estados. O processo começa em um desses estados e move-se sucessivamente de um estado para outro. Cada movimento é chamado de passo. Se a cadeia está atualmente no estado s_i , então ela se move para o estado s_j no próximo passo com uma probabilidade denotada por s_{ij} e essa probabilidade não depende dos estados ocorridos nos passos anteriores, apenas do estado atual. A probabilidade p_{ij} é chamada probabilidade de transição. O processo pode permanecer no estado que se encontra e isso ocorre com probabilidade p_{ii} .

Exemplo

Foram observados alguns dados do time de futebol Flamengo. E verificou-se que ele nunca empata dois jogos seguidos. Se ele empata, as probabilidades de ganhar ou perder o próximo jogo são iguais. Se a vitória ocorreu no jogo atual, com a prorrogação, a probabilidade de ganhar ou empatar no próximo jogo é de 1/2 e 1/4, respectivamente. Se a derrota vier no jogo atual, a probabilidade de ganhar na próxima partida diminui para 1/4 e a de perder novamente aumenta para 1/2.

Perceba que as probabilidades de resultado da partida atual não dependem de resultados anteriores ao atual, podemos, então, modelar uma cadeia de Markov.

Com as informações acima,

$$\begin{array}{c}
 V \quad E \quad D \\
 \begin{array}{c} V \\ E \\ D \end{array} \begin{bmatrix} 0,5 & 0,25 & 0,25 \\ 0,5 & 0 & 0,5 \\ 0,25 & 0,25 & 0,5 \end{bmatrix}
 \end{array}$$

podemos determinar as probabilidades de transição, representaremos numa matriz, onde D é derrota, E é empate e V é vitória. Texto retirado de Golmakani *et al.* (2014, pp. 3-4).

1.2. Etapas para definição do processo de Markov

A seguir são indicados alguns passos que permitem a visualização do processo como um todo da cadeia de Markov.

1.2.1. Definição de um MDP

Um MDP é uma sequência ordenada (tupla) (S, A, T, R) , onde:

- » S é um conjunto de estados em que o processo pode se encontrar;
- » A é um conjunto de ações que pode ser conseguida em distintas épocas de decisão;
- » $T: S \times A \times S \mapsto [0, 1]$ é uma função que dá a probabilidade de o sistema passar para um estado $s' \in S$, dado que o processo estava em um estado $s \in S$ e o agente considerou concretizar uma ação $a \in A$ (denotada por $T(s' | s, a)$);
- » $R: S \times A \mapsto \mathbf{R}$ é uma função que dá a recompensa (ou custo) por adotar uma decisão $a \in A$ quando o processo está em um estado $s \in S$.

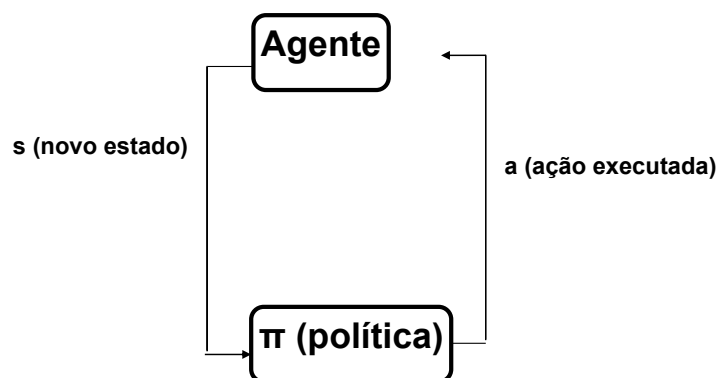
Pode também ser definido, para cada estado $s \in S$, um conjunto de ações possíveis naquele estado (A_s). Dessa forma, o conjunto de todas as ações poderia ser $A = \bigcup_s A_s$. Será usado apenas “ A ”, a fim de facilitar a notação.

Os conjuntos S e A podem ser finitos ou infinitos.

1.2.2. O Agente de um MDP

Um agente inserido em um processo de decisão de Markov deve, para cada estado, escolher uma ação. A Figura 21 mostra a dinâmica de funcionamento de um sistema modelado como processo de decisão de Markov.

Figura 21. Funcionamento de um sistema modelado como MDP.



Fonte: Elaborado pelo autor

O agente verifica o estado em que o sistema (s) se encontra, verifica uma política (π) e efetiva uma ação (a). A ação executada pode ter um efeito sobre o ambiente e, assim, modificar o estado atual. O agente então verifica o novo estado para que possa tomar a próxima decisão.

A cada época de decisão, o agente utiliza uma regra de decisão para determinar a próxima ação. Uma regra de decisão simples seria um mapeamento direto de estados em ações, por exemplo, uma regra de decisão definida por $d: S \rightarrow A$. Chamamos de política, o conjunto de todas as regras de decisão.

Usualmente é requerida uma política que aprimore um certo critério de desempenho das decisões. Uma política pode ser, segundo Pellegrini e Wainer (2003):

- » **Total**, onde cada uma de suas regras de decisão é estabelecida para todos os estados do processo de decisão de Markov;
- » **Parcial**, onde alguma de suas regras de decisão é indicada apenas para alguns estados do processo de decisão de Markov.

Em relação às épocas de decisão, uma política pode ainda ser entendida como:

- » **Estacionária**, quando a ação recomendada não depende da época de decisão (ou seja, $d_k = d_j$ para todo k e j);
- » **Não estacionária**, quando a ação executada depende da época de decisão.

Se uma política é estacionária, $\pi(s)$ podemos usar como sinônimo $d_k(s)$. Ainda assim, é mais frequente usar políticas estacionárias para horizonte infinito e políticas não estacionárias para horizonte finito, tanto as políticas estacionárias quanto as não estacionárias são suportadas por qualquer horizonte: um sistema com horizonte finito consegue ter a mesma política para cada época de decisão. Em um sistema com horizonte infinito, a política pode ser modificada periodicamente.

As políticas podem ser ainda classificadas como:

- » **Determinística** em que cada estado é sempre mapeado em uma única ação;
- » **Não determinística** (randomizada ou estocástica) em que um estado é mapeado em um conjunto de ações, no qual cada ação tem a possibilidade de ser escolhida. Assim, cada regra de decisão é uma função $d_k: S \times A \rightarrow [0, 1]$.

Por fim, uma política ainda pode ser definida como:

- » **Markoviana (ou sem memória)**, na qual a escolha da ação sujeita-se apenas ao estado corrente;
- » **Não Markoviana**, na qual a escolha da ação se sujeita a todo histórico de ações e estados do sistema até o presente momento. Pode-se definir as regras de decisão como funções $d_k: H_k \rightarrow A$, onde H_k é o histórico de ações e estados até a época de decisão k .

Ao efetivar uma política, o agente irá adquirir recompensas em cada época de decisão. Se quisermos comparar duas políticas, será necessário um critério de desempenho (caso contrário a comparação é impossível). Podem-se definir vários critérios de otimalidade (ou desempenho) para MDPs, e dentre os mais comuns podemos apontar:

- » a recompensa média por época de decisão, $\frac{1}{Z} \sum_{k=0}^{Z-1} r_k$;
- » a recompensa esperada total, $E \left[\sum_{k=0}^{Z-1} r_k \right]$;
- » a recompensa esperada descontada, $E \left[\sum_{k=0}^{Z-1} \gamma^k r_k \right]$.

A recompensa na época de decisão k é denotada por r_k e $\gamma \in [0,1]$ é um fator de desconto. O fator de desconto é usado com horizonte infinito para garantir a convergência do valor da recompensa total esperada. Quando o horizonte é infinito, usa-se o limite no infinito sobre o critério de otimalidade (por exemplo, $\frac{1}{Z} \sum_{k=0}^{Z-1} r_k$ para recompensa média).

1.2.3. Política e função valor de um MDP

Quase todos os algoritmos de aprendizado por reforço envolvem a estimativa de valor de funções – funções de estados (ou de pares de ação do estado) que estimam quão bom é para o agente estar em determinado estado (ou como é bom executar determinada ação em determinado estado). O termo **quão bom** pode ser definido em termos de recompensas futuras que podem ser esperadas ou para ser mais preciso, em termos de retorno esperado. Claro que as recompensas que o agente pode esperar receber no futuro dependem das ações. Assim, **função valor** pode ser definida como as formas particulares de agir, chamadas **políticas**.

Formalmente, uma política é um mapeamento dos estados para probabilidades de selecionar cada ação possível. De acordo com Sutton e Barto (2018), política e função valor podem ser definidas como:

1.2.4. Definição: política para um MDP

A regra de decisão para um processo de decisão de Markov em uma época de decisão k pode ser definida pela função $d_k: S \mapsto A$, que estabelece a ação a ser efetivada, dado o estado do sistema.

Uma política para um MDP é uma sequência de regras de decisão $\pi = \{d_0, d_1, \dots, d_{z-1}\}$, uma para cada época de decisão.

É necessário também que seja definida a ideia de **valor** de uma política, determinada por uma **função valor**.

1.2.5. Definição: função valor de uma política para um MDP

Uma função valor da política π para um MDP $M = (S, A, T, R)$ é definida como a função $V^\pi: S \mapsto R$, tal que $V^\pi(s)$ atribuído ao valor que se espera da recompensa para esta política, consoante com o critério de otimalidade.

1.2.6. Política ótima e função valor ótima de um MDP

Sempre que dada política for processada a partir do estado inicial, a natureza estocástica do ambiente pode levar a um histórico de ambiente distinto. Portanto, a qualidade de uma política pode ser medida pela utilidade esperada dos históricos de possíveis ambientes concebidos por essa política. Uma **política ótima** é, portanto, uma que fornece a utilidade que se espera mais alta (RUSSELL e NORVIG, 2010).

Resolver uma tarefa de aprendizado de reforço significa, grosso modo, encontrar uma política que alcance muita recompensa em longo prazo. Para MDPs finitos, podemos definir precisamente uma política ótima da seguinte forma:

1.2.7. Definição: política ótima para um MDP

Um MDP com horizonte z pode ser definido como $M = (S, A, T, R)$.

Uma função valor V^* é ótima para M se $\forall U \in V, \forall S \in S, V_k(s) \geq U_k(s)$ para todo $k \in \mathbb{N}$ tal que $0 \leq k < z$.

Uma política π é ótima para M se $\forall \pi' \in \Pi, \forall S \in S, Q_k^\pi(s, d_k^\pi(s)) \geq Q_k^{\pi'}(s, d_k^{\pi'}(s))$ para todo k tal que $0 \leq k < z$.

Há sempre ao menos uma política que é melhor ou igual a todas as outras. Esta é uma política ótima. Embora possa haver mais de uma, denotamos todas as políticas ótimas por π . Dado π , consultando sua percepção atual o agente decide o que fazer, informando

o estado atual s , e em seguida executando a ação $\pi(s)$. A função do agente é representada explicitamente por uma política. Políticas ótimas compartilham a mesma função valor de estado chamada de função valor ótima, denotada por v e definida como:

$$v(s) = \max_{\pi} v_{\pi}(s)$$

Para todo $s \in S$.

As políticas ótimas também compartilham a mesma função valor ótima, denotada por:

$$q(s, a) = \max_{\pi} q_{\pi}(s, a)$$

Para todo $s \in S$ e $a \in A(s)$. Para o par da ação (s, a) , a função fornece o retorno esperado para a ação a no estado s e depois seguindo uma política ótima.

Portanto, podemos escrever q em termos de v , como segue:

$$q(s, a) = E[R_{t+1} + \lambda v(S_{t+1}) | S_t = s, A_t = a]$$

Muitas definições matemáticas às vezes dificultam o entendimento sobre a execução do processo. Vamos tentar entender melhor com um exemplo?

1.3. Exemplo da Leoa

Imagine que uma leoa em sua fase adulta necessita de 6kg de comida por dia, para assim, conseguir armazenar em seu corpo aproximadamente 30kg de comida (o que significa que ela pode permanecer até 6 dias sem comer). A média de biomassa comestível nas zebras é de 164kg, pode-se afirmar que é grande o suficiente para alimentar diversos leões. Se uma leoa fêmea decide caçar em grupo, acredita-se que a probabilidade de sucesso na caça (isso quer dizer, a captura e morte da zebra) por perseguição aumenta com o tamanho do grupo até certo ponto, e cada perseguição consome 0,5kg da leoa para gerar energia.

Suponha que as probabilidades de sucesso na caçada sejam dadas por:

- » $p(1) = 0,15$;
- » $p(2) = 0,33$;
- » $p(3) = 0,37$;
- » $p(4) = 0,4$;
- » $p(5) = 0,42$;
- » $p(\geq 6) = 0,43$;

No qual $p(n)$ representa a probabilidade de sucesso na captura quando o grupo tem 'n' leas. Formule este problema como um MDP onde o estado representa as reservas energéticas da leoa, as ações representam a decisão de caçar ou não (e com quantas outras leas). Presuma que uma caçada por dia será feita, e que o objetivo da leoa é maximizar a probabilidade de sobrevivência por T dias. Presuma também que havendo uma captura, as leas dividem a carne igualmente. Além disso, por mais desagradável que seja, ignore os leões machos e filhotes (nenhuma parcela será dada a eles).

Para resolver esta questão, precisamos primeiro observar que a leoa faz suas decisões baseando-se em seu nível energético, distinguido por três níveis (além da condição está representada, a seguir, a nomenclatura de suas variáveis):

1. **Com fome** – sem_comida;
2. **Comeu no dia de hoje** – comida_hoje,
3. **Comeu o suficiente** – satisfeita.

Esta tem a possibilidade de escolher entre não comer, caçar, e caçarem grupo de 'n' outras leas. Com isto, pode-se definir o conjunto de **estados 'S'**:

$$S = \{\text{sem_comida; comida_hoje; satisfeita}\}$$

E o conjunto de **ações A(S)**:

$$A(\text{sem_comida}) = \{\text{caçar, caçar2, caçar3, caçar4, caçar5, caçar6}\}$$

Onde, '**caçarn**' representa a ação de caçar com outras 'n' leas. E '**caçar6**' o ato de caçar com seis ou mais leas.

$$A(\text{comida_hoje}) = \{\text{não_caçar, caçar2, caçar3, caçar4, caçar5, caçar6}\}$$

$$A(\text{satisfeita}) = \{\text{não_caçar}\}$$

Por este ser um MDP finito, pode-se escrever as probabilidades de transição e recompensas esperadas como na Tabela 2. Algumas considerações foram levadas em conta no processo de modelagem, que são:

- » Considerando a média de biomassa de uma zebra, admitiu-se nesta modelagem que uma leoa, ao se alimentar de uma zebra torna-se satisfeita. Esta condição é válida até 5 leas. Para 6 ou mais, a probabilidade de uma leoa tornar-se satisfeita é $\text{prob_captura}/(2*n)$ onde n é o número de leas e prob_captura é a probabilidade de sucesso na captura de uma zebra. O 2 entra na conta, pois se torna igualmente possível que a leoa se torne satisfeita ou vá para o estado "comida_hoje";

- » A pior recompensa para a leoa é -3. A recompensa 0 é aquela em que a ação não lhe ajuda e nem lhe prejudica.

Tabela 2. Modelagem MDP do Problema da Leoa.

$S = S_t$	$a = a_t$	$S' = S_{t+1}$	$P_{SS'}^a$	$R_{SS'}^a$
sem_comida	não_caçar	sem_comida	1	-2
		comida_hoje	0	0
		satisfeita	0	0
	caçar	sem_comida	0,85	-3
		comida_hoje	0	0
		satisfeita	0,15	3
	caçar2	sem_comida	0,66	-3
		comida_hoje	0	0
		satisfeita	0,33	3
	caçar3	sem_comida	0,63	-3
		comida_hoje	0	0
		satisfeita	0,37	3
	caçar4	sem_comida	0,6	-3
		comida_hoje	0	0
		satisfeita	0,4	3
	caçar5	sem_comida	0,58	-3
		comida_hoje	0	0
		satisfeita	0,42	3
	caçar6	sem_comida	0,57	-3
		comida_hoje	0,43/2n	2
		satisfeita	0,43/2n	3
comida_hoje	não_caçar	sem_comida	0,5	-3
		comida_hoje	0,5	0
		satisfeita	0	0
	caçar2	sem_comida	0,66	-3
		comida_hoje	0	0
		satisfeita	0,33	3
	caçar3	sem_comida	0,63	-3
		comida_hoje	0	0
		satisfeita	0,37	3
	caçar4	sem_comida	0,6	-3
		comida_hoje	0	0
		satisfeita	0,4	3
	caçar5	sem_comida	0,58	-3
		comida_hoje	0	0
		satisfeita	0,42	3
	caçar6	sem_comida	0,57	-3
		comida_hoje	0,43/2n	2
		satisfeita	0,43/2n	3
satisfeita	não_caçar	sem_comida	0	0
		comida_hoje	0	0
		satisfeita	1	0

Fonte: Batista, 2008.

1.4. Exemplo da Zebra

Considere agora que uma zebra precisa andar frequentemente até um lago para beber água, como qualquer outro animal que necessita de água para sua sobrevivência. Em um ambiente como uma selva, é preciso avaliar as condições para chegar até esse lago, e há dois caminhos que ela pode escolher, e em ambos há o risco de ataque por leões (as mesmas que analisamos no exemplo anterior).

Então, estes são os dois caminhos que podem ser levados em consideração:

1. As leões atacam com mais frequência:
 - a. 70% do tempo;
 - b. Em grupos menores (entre 1 e 4).
2. As leões atacam menos frequentemente:
 - a. 40% do tempo;
 - b. Em grupos maiores (entre 3 e 6, com igual probabilidade).

Como as leões não se expõem até o momento da perseguição, as zebras precisam observar o ambiente para tentar descobrir qual o melhor caminho, antes de fazer a escolha. E tem-se algumas observações:

- » Outras zebras voltando por um ou por outro caminho, embora não sejam garantia alguma, são um forte indicativo de que aquele caminho está livre. 80% das vezes em que zebras são avistadas vindo por um caminho, este caminho está livre de leões;
- » Pegadas de leões podem indicar que há leões em um dos caminhos, mas a zebra não pode ter certeza de quando as pegadas foram produzidas. No entanto, ela pode ter uma ideia do tamanho do grupo que deixou as pegadas: 70% das vezes a zebra consegue determinar com exatidão o número de leões que deixaram as pegadas. Outras 20% das vezes, erra por um o número de leões. Nas outras 10% das vezes, erra por dois;
- » A zebra sabe que 80% das vezes em que havia leões em um dos caminhos, elas continuavam lá na vez seguinte.

Formule este problema como um POMDP, ponto de vista da zebra. O óbvio objetivo dela é chegar com vida ao lago. Ela deve repetir esta decisão 'n' vezes.

Para resolver o problema das zebras vamos considerar que 'S' seja o conjunto de estados que representam os caminhos pelos quais as leões podem estar.

$$S = \{\text{caminho1; caminho2}\}$$

E que $A(S)$ é o conjunto das ações possíveis que as zebras podem executar.

$$A(S) = \{\text{Observar; Ir_caminho1; Ir_caminho2}\}$$

Como se sabe que em **Caminho1** as leoas atacam 70% das vezes em grupos de até 4 leoas e em **Caminho2** elas atacam 40% das vezes em grupos de 3 a 6 leoas, podemos inferir que, em função disto, é possível definir as recompensas:

$$R = \{ +3, \text{ caso a decisão seja ir por um caminho onde não há leoas; } -3, \text{ caso a decisão seja ir por um caminho onde há leoas} \}$$

O conjunto de observações Ω é dado por:

$$\Omega = \{ \text{zebras voltando, pegadas, leoas no caminho} \}$$

Com isto, este conjunto Ω , com as possíveis considerações, constitui a base que nos fornece as seguintes probabilidades:

- » 80% de chance de não haver leoa caso existam zebras voltando pelo caminho;
- » 20% de chance de não haver leoa caso fossem observadas leoas no caminho;
- » 70% de determinar o número de leoas caso seja possível observar as pegadas;
- » 20% de errar por um o número de leoas caso seja possível observar as pegadas;
- » 10% de errar por dois o número de leoas caso seja possível observar as pegadas.

CAPÍTULO 2

ALGORITMOS PARA SOLUÇÃO DE MDPS

Existe uma grande quantidade de algoritmos para a solução de MDPs. Abordaremos alguns desses algoritmos a seguir. Alguns deles trabalham diretamente com políticas, enquanto outros trabalham com funções valor.

Neste capítulo abordaremos um algoritmo chamado **iteração de valor**, usado para calcular uma política ótima. O propósito é calcular a utilidade de cada estado e, posteriormente, utilizar as utilidades do estado para eleger uma ação ótima em cada estado.

Veremos também o algoritmo de **iteração por política** que sugere um caminho alternativo para encontrar políticas ótimas, que mostra que, quando uma ação é claramente melhor que todas as outras, a magnitude exata das utilidades nos estados envolvidos não precisa ser exata.

2.1. Iteração de política

É possível conseguir uma política ótima até mesmo quando a estimativa de função utilidade não é exata. Caso se tenha uma ação melhor que todas as outras, a grandeza exata das utilidades nos estados implicados não necessita ser precisa. Isso sugere um caminho alternado para encontrar políticas ótimas (RUSSELL e NORVIG, 2010).

O algoritmo de iteração política, que será estudado adiante, alterna a avaliação de política e a melhoria de política começando com uma política inicial π_0 .

2.1.1. Avaliação de política

Dada uma política π_i , calcular $U_i = U^{\pi_i}$, se a utilidade de cada estado π_i fosse executada. A avaliação de política pode ser expressa como uma versão simplificada da equação de Bellman, relacionando a utilidade de s (sob π_i), às utilidades de seus vizinhos:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

Segundo Russell e Norvig (2010, p. 573), “para espaços de estados pequenos, a avaliação de política usando métodos de solução exata em geral é a abordagem mais eficiente”. Para os espaços de estados grandes, o tempo do algoritmo sendo $O(n^3)$ talvez não seja o recomendado. No entanto, não é necessário fazer a avaliação de política exata. Ao invés disso, podemos executar algum número de passos de iteração de valor simplificada para

fornecer uma aproximação razoavelmente boa das utilidades. Nesse caso, a atualização de Bellman simplificada para esse processo é:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

Onde é repetida k vezes para produzir a próxima estimativa de utilidade. O algoritmo que implementa a atualização de Bellman simplificada é chamado de **avaliação de política iterativa**.

Sutton e Barto (2018) propuseram um algoritmo que implementa a avaliação de política iterativa, como pode ser visto no Quadro 1.

Observe como o algoritmo lida com a terminação. Formalmente, a avaliação de política iterativa converge apenas no limite, mas, na prática, deve ser interrompida. Dessa forma, o algoritmo testa a quantidade $\max_{s \in S} |U_{i+1}(s) - U_i(s)|$ depois de cada varredura e para quando é suficientemente pequeno.

Quadro 1. Algoritmo de avaliação de política iterativa.

```

função AVALIAÇÃO-DE-POLÍTICA-ITERATIVA
entrada: a política  $\pi$  a ser avaliada
variáveis locais: um limiar pequeno  $\theta > 0$  determinando a precisão da estimativa
inicialize  $U(s)$ , para todo  $s \in S$ , exceto  $V=0$ 
repita
 $\Delta \leftarrow 0$ 
para cada estado  $s \in S$  faça
 $u \leftarrow U(s)$ 
 $U(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U(s')$ 
 $\Delta \leftarrow \max(\Delta, |u - U(s)|)$ 
até  $\Delta < \theta$ 
    
```

Fonte: Elaborado pelo autor.

2.1.2. Melhoria de política

O processo de fazer uma nova política que melhora uma política original, tornando-a gulosa no que diz respeito à função valor da política original, é chamado de melhoria de política.

Suponha que a nova política gulosa, π' é tão boa quanto, mas não melhor que, a política original π . Em outras palavras, vamos considerar a nova política gulosa π' , dada por:

$$\pi'(s) = \operatorname{argmax}_a \sum_{s',r} P(s',r|s,a) [r + \gamma v_{\pi}(s')]$$

Onde argmax_a denota o valor de a em que a expressão que se segue é maximizada. A política gulosa toma a ação que parece melhor em curto prazo, de acordo com v_{π} .

2.1.3. Teorema da melhoria de política

Seja π e π' qualquer par de políticas determinísticas tais que, para todo $s \in S$,

$$q_{\pi}(s, \pi'(s)) \geq v_{\pi}(s)$$

Então, a política π' deve ser tão boa ou melhor que π . Dessa forma, podemos verificar que a política gulosa satisfaz as condições do teorema da melhoria política, e é tão boa quanto, ou melhor, que a política original.

Suponha que a nova política gulosa π' é tão boa quanto, mas não melhor que a política original π . Então $v_{\pi} = v_{\pi'}$, para todo $s \in S$, temos:

$$v_{\pi'}(s) = \max_a \sum_{s', r} P(s', r | s, a) [r + \gamma v_{\pi'}(s')]$$

Você deve estar se perguntando se essa equação é a mesma equação de otimização de Bellman, e, portanto, $v_{\pi'}$ pode ser v e as duas políticas π e π' devem ser políticas ótimas. Melhoria de política, portanto, deve nos dar uma política estritamente melhor, exceto quando a política original já é ótima.

2.1.4. Algoritmo da iteração de política

Uma vez que uma política π tenha sido melhorada usando v_{π} para produzir uma política melhor π' , podemos, então, calcular $v_{\pi'}$ e melhorá-la para obter um π'' ainda melhor.

Podemos, assim, obter uma melhor sequência de políticas e funções valor, como pode ser visto na equação a seguir, descrita por Sutton e Barto (2018).

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_3 \xrightarrow{E} v_{\pi_3}$$

Onde \xrightarrow{E} denota uma avaliação de política e \xrightarrow{I} denota uma melhoria de política. Cada política é garantidamente uma melhoria estrita em relação à anterior (a menos que já seja ótima). Como um MDP finito tem apenas um número finito de políticas, esse processo deve convergir para uma política ótima e função valor ótima em um número finito de iterações.

A maneira para encontrar uma política ótima é chamada de iteração de política. O algoritmo ITERAÇÃO-DE-POLÍTICA, proposto por Russell e Norvig (2010) é apresentado no Quadro 2.

Observe que cada avaliação de política é iniciada com a função valor da política anterior. Isso normalmente resulta em um grande aumento na velocidade de convergência da

avaliação de políticas (presumidamente porque a função valor muda pouco de uma política para outra).

Quadro 2. Algoritmo de iteração de política para calcular uma política ótima.

```

função ITERAÇÃO-DE-POLÍTICA(mdp) retorna uma política
  entradas: mdp, um MDP com estados S, ações A(s), modelo de transição  $P(s' | s, a)$ 
  variáveis locais: U, um vetor de utilidades para estados em S, inicialmente zero e,
   $\pi$ , um vetor de política indexado pelo estado, inicialmente aleatório

  repita
     $U \leftarrow \text{AVALIAÇÃO-DE-POLÍTICA}(\pi, U, mdp)$ 
    inalterado?  $\leftarrow$  verdadeiro
    para cada estado s em S faça
      se  $\max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s'] > \sum_{s'} P(s' | s, \pi[s]) U[s']$  então faça
         $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s']$ 
        inalterado?  $\leftarrow$  falso
    até inalterado?
  retornar  $\pi$ 

```

Fonte: Russell e Norvig (2010, p. 574).

2.1.5. A equação de Bellman

Russell e Norvig (2010) apontam que a utilidade de uma sequência de estados é a soma das recompensas descontadas obtidas durante a sequência. Ao se decidir dessa forma é possível comparar políticas, comparando as utilidades esperadas obtidas quando as executamos.

A partir dessa afirmação, segue que há uma relação direta entre a utilidade de um estado e a utilidade de seus vizinhos: “a utilidade de um estado é a recompensa imediata correspondente a esse estado mais a utilidade descontada esperada do próximo estado, assumindo que o agente escolha a ação ótima” (RUSSELL; NORVIG, 2010, p. 569).

A utilidade de um estado *s* é conhecida como **Equação de Bellman** e pode ser definida como:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')$$

Onde:

γ é o fator de desconto;

$U(s)$ é a recompensa total a “longo prazo” de *s*;

$R(s)$ é a recompensa a “curto prazo” por estar em *s*.

Toda função valor satisfaz essa equação, dada uma política π .



Understanding RL: The Bellman Equations

Assista ao vídeo: The Bellman Equations...

The Bellman Equation

2.2. O algoritmo de iteração de valor

A equação de Bellman é a base do algoritmo de iteração de valor para a solução de MDPs. Caso haja n estados possíveis, teremos n equações de *Bellman*, sendo uma para cada estado. Assim sendo, n equações contêm n incógnitas – as utilidades dos estados (RUSSELL e NORVIG, 2010).

Porém, ao tentar resolver essas equações simultaneamente para encontrar as utilidades, nos deparamos com um problema. As equações não são lineares porque o operador *max* não é um operador linear. Ao passo que sistemas de equações lineares normalmente são solucionados rapidamente desde que se faça uso de técnicas de álgebra linear. É sabido que sistemas de equações não lineares são mais difíceis de ser solucionados.

Uma abordagem iterativa pode ser usada para sanar esse problema. Iniciaremos com valores aleatórios para as utilidades, assim será calculado o lado direito da equação e o resultado será inserido no lado esquerdo, atualizando, dessa forma, a utilidade de cada estado a partir das utilidades de seus vizinhos. Esse processo é repetido até que se chegue a um equilíbrio. Seja $U_i(s)$ o valor de utilidade para cada estado s na i -ésima iteração. Essa fase de iteração, é conhecida como atualização de Bellman, podendo ser assim definida:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U_i(s')$$

Nela, assumimos que a atualização será aplicada juntamente a todos os estados em cada iteração. Ao aplicar a atualização de Bellman com frequência infinita, podemos garantir o alcance de um equilíbrio e, dessa forma, os valores finais de utilidade devem ser soluções para as equações de Bellman. Na prática, eles igualmente são as únicas soluções, e a política equivalente é ótima (RUSSELL; NORVIG, 2010).



A iteração de valor é, portanto, um método de aproximações sucessivas para encontrar uma função valor ótima sem requerer o conhecimento prévio de uma política ótima, sendo a política ótima obtida a partir da função valor ótima.

Russell e Norvig (2010) propôs o algoritmo chamado ITERAÇÃO-DE-VALOR, mostrado no Quadro 3.

O algoritmo de iteração de valor pode ser usado para resolver MDPs com horizonte infinito, porque converge para uma função valor ótima. Seja um horizonte finito, a equação de otimalidade pode ser assim definida:

$$\begin{aligned} U(s) &= HU(s) \\ &= \max_{a \in A} h(s, a, U) \\ &= \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} P(s' | s, a) U(s') \right] \end{aligned}$$

Onde, H é definido como um operador $H: U \mapsto U$ de melhoria da política, que determina a melhor ação de um estado usando os valores U dos estados numa precedente época de decisão.

Seja o operador H , e a equação de otimalidade para processos de decisão de Markov determinada por uma função valor em uma época de decisão a contar da função valor da época de decisão anterior.

Quadro 3. Algoritmo de iteração valor para calcular utilidades de estados.

```
função ITERAÇÃO-DE-VALOR(mdp,  $\varepsilon$ ) retorna uma função utilidade
entradas: mdp, um MDP com estados  $S$ , ações  $A(s)$ , modelo de transição  $P(s' | s, a)$ , recompensa  $R(s)$ , desconto  $\gamma$  e  $\varepsilon$  o erro máximo permitido na utilidade de cada estado
variáveis locais:  $U, U'$ , vetores de utilidades para estados em  $S$ , inicialmente zero e,  $\delta$  a mudança máxima na utilidade de qualquer estado em uma iteração

repita
   $U \leftarrow U'; \delta \leftarrow 0$ 
para cada estado  $s$  em  $S$  faça
   $U'(s) \leftarrow R[s] + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s]$ 
se  $|U'[s] - U[s]| > \delta$  então  $\delta \leftarrow |U'[s] - U[s]|$ 
até  $\delta < \varepsilon(1 - \gamma)/\gamma$ 
retornar  $U$ 
```

Fonte: Russell e Norvig (2010, p. 570).

Para o caso de horizonte finito, o critério de parada é quando o número de iterações é igual ao horizonte do problema. Assim, o algoritmo produz uma política para cada época de decisão, sendo a política não estacionária. A complexidade assintótica do algoritmo é dada por $O(z |A| |S|^2)$.

Para processos com horizontes infinitos (ou indefinidos), o algoritmo é interrompido quando os valores para cada estado tiverem convergido. Na prova de convergência dos algoritmos de iteração de valores define-se um conceito essencial, conhecido como **contração em um espaço Banach**.

2.2.1. Definição: espaço de Banach

Uma norma $||\cdot||$ num espaço vetorial X é um espaço de Banach se toda sequência de Cauchy (x_0, x_1, \dots, x_n) de elementos de X tem um limite $x_k \in X$.

2.2.2. Definição: contração

Dado X um espaço de Banach. Um operador $F: X \rightarrow X$ é uma contração quando para quaisquer dois elementos $U, V \in X$:

$$||FV - FU|| \leq \beta ||V - U||$$

Onde $0 \leq \beta \leq 1$ é o fator de contração.

2.2.3. Teorema do ponto fixo de Banach

Seja X um espaço de Banach. Seja $F: X \rightarrow X$ uma contração e seja $N = (x_0, x_1, \dots, x_k)$ uma sequência com um ponto inicial arbitrário $x_0 \in X$, tal que $x_i = Fx_{i-1}$. Então:

F tem um único ponto fixo x tal que $Fx = x$;

A sequência N converge para x .

Seja $||\cdot||$ uma norma no espaço V tal que $||V|| = \max_{s \in S} |V(s)|$. V é um espaço de Banach. Ademais, o operador H é uma contração em V (a prova dessa afirmação pode ser vista em Puterman (1994)). Assim, o teorema de Banach garante que H tem um ponto fixo único v e que a sequência $v_k = Hv_{k-1}$, iniciando de uma função valor qualquer, converge para este ponto fixo.

Para que se alcance uma função valor precisa do algoritmo; não é suficiente apenas a garantia da convergência. É preciso que se determine também quando se deve interromper o cálculo de aproximações sucessivas da função valor. O Teorema do Erro de Bellman é um critério de convergência bastante conhecido para isso.

2.2.4. Teorema do Erro de Bellman

Se a diferença entre $v_k(s)$ e $v_{k-1}(s)$ é no máximo δ para todo estado s , então $v_k(s)$ nunca difere de $v(s)$ por mais de $\frac{\delta}{1-\gamma}$

Assim, o critério de parada deve ser:

$$\forall_s \in S, |V(s) - V'(s)| \leq \frac{\varepsilon(1-\gamma)}{2\gamma}$$

Para que a precisão da solução seja no mínimo ε .

2.3. Exemplo do Aluguel de carros do Jack

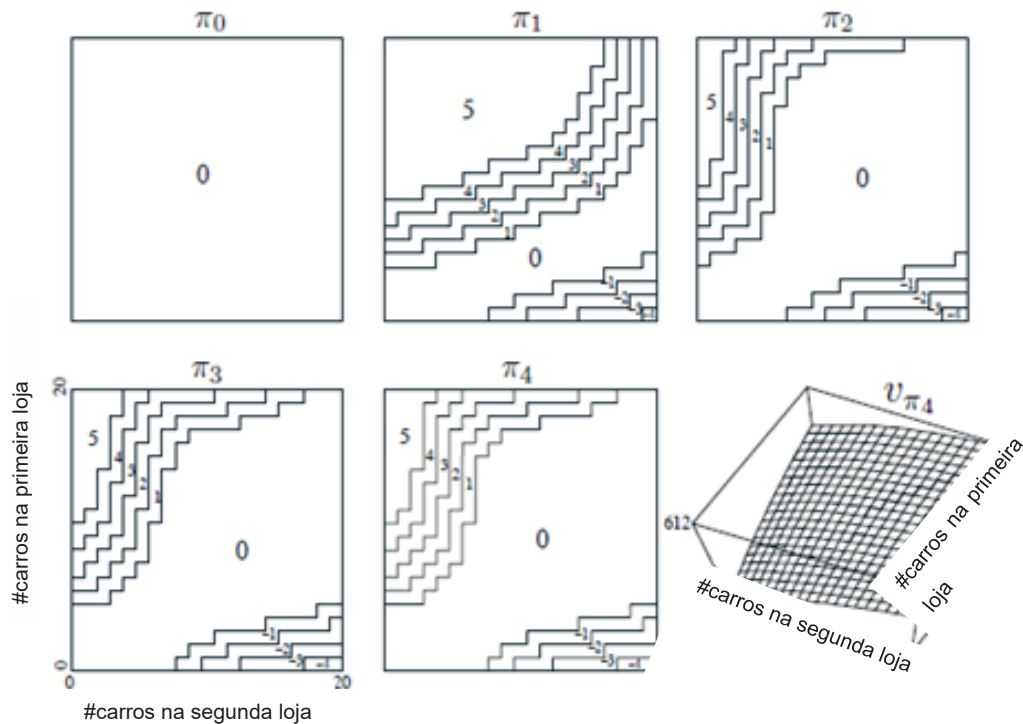
Exemplo adaptado de Sutton e Barto (2018).

Jack administra duas lojas para uma empresa nacional de aluguel de carros. Todos os dias, clientes entram em suas lojas e alugam carros. Quando Jack aluga um carro ele recebe \$ 10 da empresa nacional. Quando Jack não está em nenhuma das duas lojas, então ele não recebe. Os carros ficam disponíveis para serem alugados um dia depois que foram devolvidos. Para ajudar a garantir que os carros estarão disponíveis nas lojas quando necessário, Jack resolveu movê-los entre as duas lojas durante a noite a um custo de \$2 por carro removido. Assumimos que o número de carros solicitados e retornados em cada loja são variáveis aleatórias de Poisson, o que significa que a probabilidade de que o número seja n é $\frac{\lambda^n}{n!} e^{-\lambda}$, onde λ é o número esperado. Suponha que λ seja 3 e 4 para solicitações de locação na primeira e segunda loja e 3 e 2 para devoluções. Para simplificar o problema, vamos assumir que não pode haver mais de 20 carros em cada loja (quaisquer carros adicionais são devolvidos para a empresa em todo o país, e assim desaparece o problema) e um máximo de cinco carros podem ser movidos de um local para outro em uma noite. Tomaremos a taxa de desconto como $\gamma = 0.9$ e formular o problema como um MDP finito contínuo, onde os passos de tempo são dias, o estado é o número de carros em cada loja no final do dia e, as ações são o número de carros movidos entre as duas lojas durante a noite. A Figura 22 mostra a sequência de políticas encontradas pela iteração de políticas a partir da política que nunca move nenhum carro.

Os cinco primeiros diagramas mostram para cada número de carros em cada loja no final do dia, o número de carros a serem removidos da primeira loja para a segunda (números negativos indicam transferências da segunda loja para a primeira). Cada política sucessiva é uma melhoria estrita sobre a política anterior, e a última política é ótima.

A iteração de política geralmente converge em surpreendentemente poucas iterações, como ilustra o exemplo do aluguel de carros do Jack. O diagrama de fundo inferior mostra a função valor para a política aleatória equiprovável, e o diagrama inferior direito mostra uma política gulosa para essa função valor. O teorema de melhoria de políticas nos garante que essas políticas são melhores que a política original. Nesse caso, no entanto, essas políticas não são apenas melhores, mas ótimas, indo para o estado terminal no número mínimo de etapas. Neste exemplo, a iteração de políticas encontraria a política ideal após apenas uma iteração.

Figura 22. Sequência de políticas encontradas pela iteração de políticas.



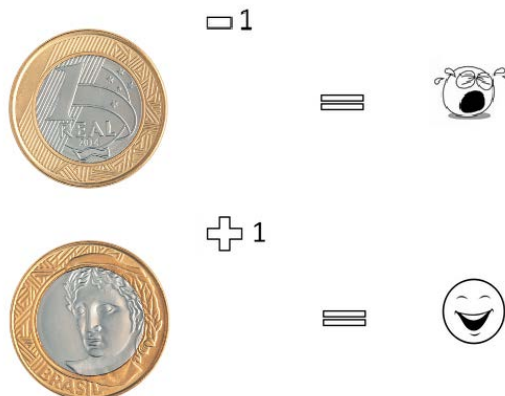
Fonte: Sutton e Barto (2018, p. 103).

2.4. Problema do jogador

Exemplo adaptado de Sutton e Barto (2018).

Um jogador tem a oportunidade de apostar nos resultados de uma sequência de lançamentos de uma moeda. Se ao jogar a moeda a face for “cara”, ele ganhará uma quantidade de dólares que apostou naquela face. Se a face da moeda for “coroa”, ele perde a aposta, como mostra a Figura 23.

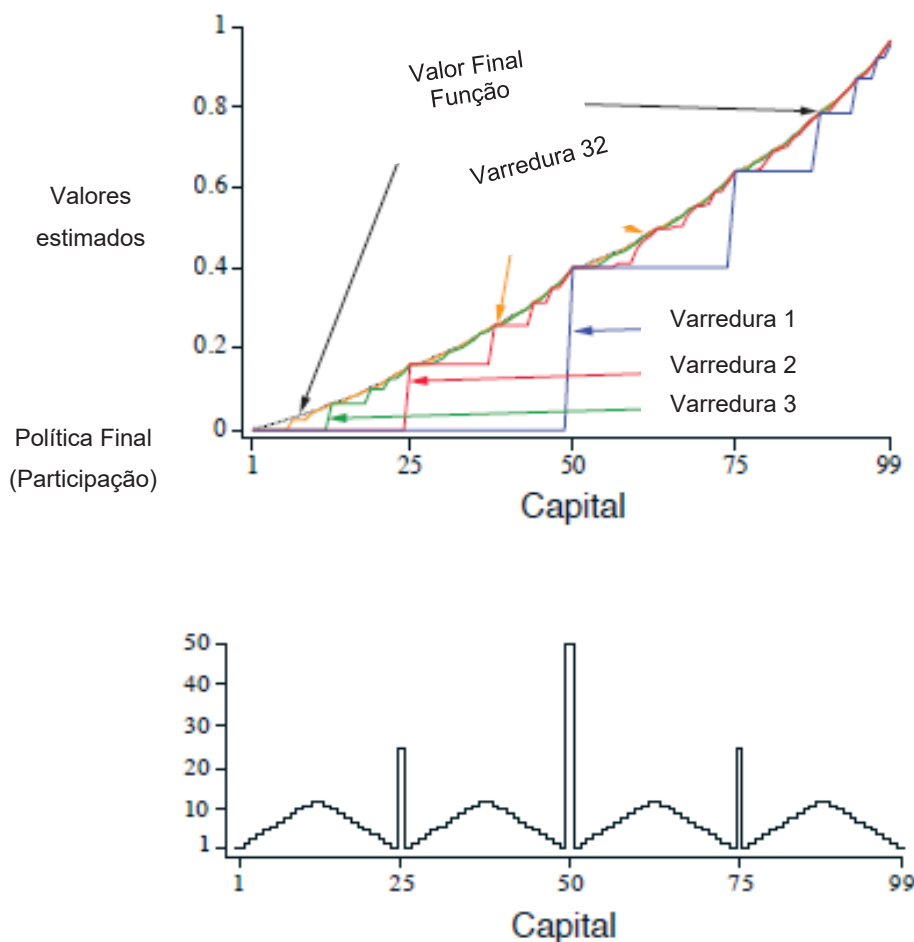
Figura 23. Representação do jogo, indicando a vitória (quando a recompensa é positiva) para quando se tira “cara” na moeda e derrota (quando a recompensa é negativa) quando “coroa”.



Fonte: Elaborado pelo autor.

O jogo termina quando o apostador ganha, atingindo seu objetivo de \$100,00, ou perde, ficando sem dinheiro. Em cada jogada, o apostador precisa decidir que parte do seu capital apostará, em números inteiros de dólares. Esse problema pode ser formulado como um MDP finito. O estado é o capital do jogador, $s \in \{1, 2, \dots, 99\}$ e as ações são as apostas, $a \in \{0, 1, 2, \dots, \min(s, 100 - s)\}$. A recompensa é zero em todas as transições, exceto aqueles em que o jogador atinge o seu objetivo, quando é +1. A função valor do estado dá, então, a probabilidade de ganhar em cada estado. Uma política é um mapeamento de níveis de capital que o jogador participa. A política ótima maximiza a probabilidade de atingir o objetivo. Seja p_h a probabilidade de a moeda chegar a uma das faces. Se p_h é conhecido, então todo o problema é conhecido e pode ser resolvido, por exemplo, por iteração de valor. A Figura 24 mostra a mudança na função valor por meio de varreduras sucessivas de iteração de valor, e a política final encontrada, para o caso de $p_h = 0.4$. Essa política é ótima, mas não exclusiva. De fato, há toda uma família de políticas ótimas, todas correspondendo a laços para a seleção de ações argmax com relação à função valor ótima.

Figura 24. Solução para o problema do jogador quando $p_h = 0.4$.



Fonte: Sutton e Barto (2018, p. 103).

2.5. Exemplo do Super Mario

E que tal vermos essas mesmas definições e exemplos de maneira diferente? Acredito que você já ouviu falar do jogo do Super Mario.

Então, nesta disciplina estamos aprendendo uma técnica que tem como objetivo treinar um agente para interagir, dado um ambiente definido, sempre querendo garantir o objetivo proposto, de acordo com as ações possíveis. No aprendizado por reforço, leva-se em consideração recompensas ou punições de acordo com as ações do agente, assim, ele vai aprender quais delas deve seguir para obter o maior número de recompensas e atingir o objetivo.

O nosso exemplo leva em consideração o Mario em uma fase do jogo, com o objetivo (que é bem comum), de salvar a Princesa das garras do Bowser. O objetivo somente será atingido se ele seguir algumas regras e caminhos, no caso, ele precisa passar por várias fases, derrotando os inimigos que aparecem pelo caminho e pegando os brindes que aparecerem. Para passar de uma fase no jogo, há a necessidade de um algoritmo treinado, imagine que é sabido quando, quantas vezes e quais botões, em quais sequências é preciso apertar para cada tipo de desafio na fase.

Dado todo aquele conceito da Cadeia de Markov, podemos compará-lo ao jogo de acordo com os seus estados, como mostra também a Figura 25, da seguinte maneira:

- » Andar;
- » Ficar parado;
- » Pular.

Figura 25. Cadeia de Markov no jogo do Mario.



Fonte: Vasconcellos, 2018.

Como já é sabido, na Cadeia de Markov, o estado em que o Mario vai estar no futuro só depende do estado em que ele está no momento. Considere que ele está parado. O fato de ele estar parado vai indicar no seu próximo movimento e estado, mas como ele estava antes de estar parado não importa. Isto é, não interessa se Mario ficou pulando loucamente para passar um espaço vago, se ele está parado nos últimos minutos, é o que precisa ser analisado.

E qual a importância dessa informação? Porque através da avaliação probabilística é possível calcular a probabilidade de ele escolher o estado seguinte ao seu estado atual. Neste caso, pode-se definir e saber quais as reais chances de Mario ter ido do estado parado para o estado pulando ou andando. Para analisar essas possibilidades, vamos analisar a matriz de transição:

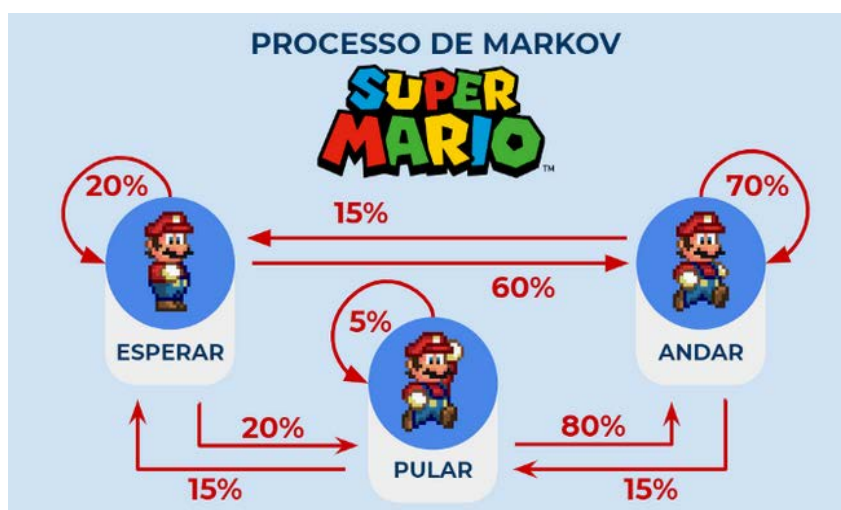
Tabela 3. Tabela de transição do jogo do Mario com as suas probabilidades de mudança de estados.

	Matriz de transição de Mario		
	Andar	Ficar parado	Pular
Andar	0,7	0,15	0,15
Ficar parado	0,6	0,2	0,2
Pular	0,8	0,15	0,05

Fonte: Elaborado pelo autor.

A Tabela 3 mostra a matriz de transição que nada mais é que uma tabela em que cada linha e cada coluna é um possível estado do Mario. Como só estamos analisando três estados, a tabela é uma matriz cruzada $M_{3 \times 3}$.

Figura 26. Processo de Markov para o jogo do Mario.







Fonte: Vasconcellos, 2018.

Os valores de probabilidade associados de cada célula da tabela é o quão provável Mario irá para aquele próximo estado. A regra é sair do estado que está representado na linha, para o próximo estado, representado na coluna. Vamos analisar alguns pontos descritivamente:

- » Quando Mario está andando, é mais provável, 70% de chances de ele continuar neste estado;
- » Quando Mario está parado, é mais provável, 60% de chances de ele andar;
- » Quando Mario está pulando, é menos provável, 5% de chances de ele continuar pulando.

Estas mesmas informações são vistas no processo de Markov ilustrado na Figura 26.

Figura 27. Matriz de estados do jogo do Mario de forma simplificada.

			+1 
			-1 
			

Fonte: Vasconcellos, 2018.

Como vimos ao longo destes capítulos o processo de decisão de Markov é representado por algumas variáveis, que são:

1. **S (conjunto finito de estados):** é literalmente o estado em que o seu personagem ou o seu objeto pode se encontrar. No jogo do Mario, os exemplos de estados foram está parado, andando ou pulando;

Porém, lembre-se de que isto é avaliado para cada caso. Imagine que eu estou jogando xadrez, o estado “pulando” não iria se encaixar.

2. **A (conjunto finito de ações):** igualmente intuitivo, a ação é o ato de fazer algo acontecer. Se quero que o Mario pule, eu preciso tomar alguma iniciativa, fazer alguma ação para que ele vá para o estado “pulando”.

Como o assunto é o jogo do Mario, as ações são os movimentos que o jogador faz para que seu avatar (Mario, Luigi) vá para um estado, por exemplo, eu cliquei no botão “↑↑↑”, essa ação faz com que meu Mario pule 3 vezes.

3. **P (modelo de probabilidade):** é a probabilidade de uma ação que será escolhida pelo jogador para um estado futuro, ser baseada no estado atual do Mario. Se a fase é aquela em que tenho que “ir para o céu”, então a probabilidade de eu escolher a ação “↑” e ir para o estado pulando é muito maior do que a ação “←” e ir para o estado andando.

4. **R (recompensa):** é uma variável que representa um número que o Mario vai receber depois de executar a ação passada para ele e ele entrar no estado atual. Neste caso, há duas possibilidades como vimos nos conceitos, uma de recompensa positiva e uma de recompensa negativa.

Imagine que você tentou pular para ir para o céu, mas errou e caiu no abismo, sua ação levou a uma recompensa negativa, você foi punido, você perdeu uma vida! Mas se você pulou certo e chegou a um bloco que lhe deu um cogumelo, sua ação o levou a uma recompensa positiva, e você foi beneficiado com esta ação.

Portanto, se seu caminho tiver mais recompensas positivas, o quanto você maximizar esse resultado, mais rapidamente atingirá o seu objetivo final, que é finalizar a fase atual, matar o chefe, acabar todas as fases, enfim, salvar a princesa Peach.

5. **γ (fator de desconto):** é um número, que normalmente varia entre 0 e 1, que molda o total de recompensa que o agente vai ganhar no futuro. Vamos supor que no jogo, se Mario pegar o caminho A qualquer, que tem um fator de desconto de 0,7 e pode pegar 100 moedas. Mas se ele escolher o caminho B, que tem um fator de desconto 0,9, ele pode pegar 300 moedas.

Isto significa dizer que suas ações atuais (que não têm influência nas ações futuras), ainda assim, podem interferir nelas – maximizar a recompensa futura, não só a atual.

O aprendizado por reforço é feito por tentativa e erro, assim conseguindo compreender bem quais ações são melhores para que estados futuros proporcionem melhores recompensas. Certo, mas vamos visualizar isso ainda no jogo do Mario, porém de maneira simplificada. Observe a Figura 27, que tem uma matriz com quadros que mostram o estado inicial

do Mario, os pontos por onde ele pode passar, e personagens que podem proporcionar recompensas ou punições, e ainda tem um complicador na posição 2 x 2, algo que o impede de seguir por ali.

Na Figura 28 é ilustrado este mesmo ambiente, no qual podemos ver quais os possíveis estados do Mario – quase todos da matriz 4 x 4, só não a posição 2 x 2. As conhecidas ações do Mario neste ambiente são:

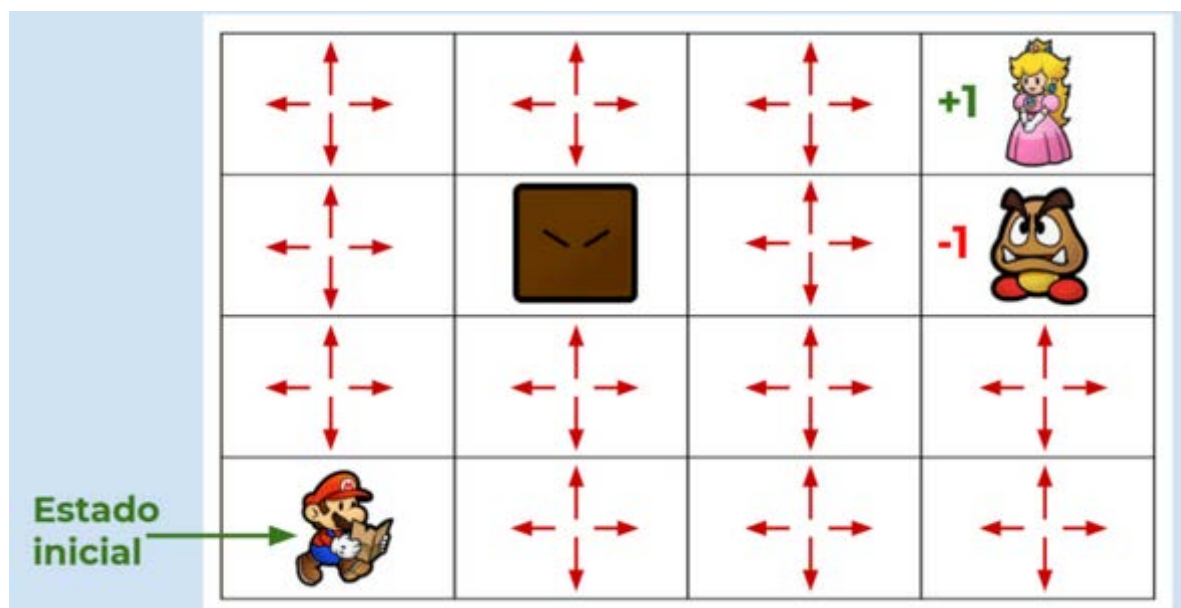
$A = \{\text{Ir para cima, ir para baixo, ir para esquerda, ir para direita}\}$

Neste caso, todos os estados possibilitam tomar qualquer uma das ações disponíveis para o Mario. Por isso que há setinhas para cima, por exemplo, na posição 1 x 1.

Sabemos que em uma representação de jogo como essa, sempre haverá limitações, já que o ambiente nos jogos não é infinito – ele pode até fazer parecer, mas não é.

Mas como podemos analisar e no aprendizado oferecer recompensas ou punições para ações como estas que não vão levar o Mario a canto nenhum? Simples, é preciso setar os valores das recompensas que ele vai ganhar de acordo com as suas ações. A Figura 29 ilustra exatamente isto. Na posição 3 x 1 da matriz, há um estado mostrando que, para aquela posição, se o Mario agir para cima, para a direita ou para baixo, ele terá uma recompensa maior do que ele agir para a esquerda – que tem uma parede e, de fato, ele não vai se mexer, vai continuar parado.

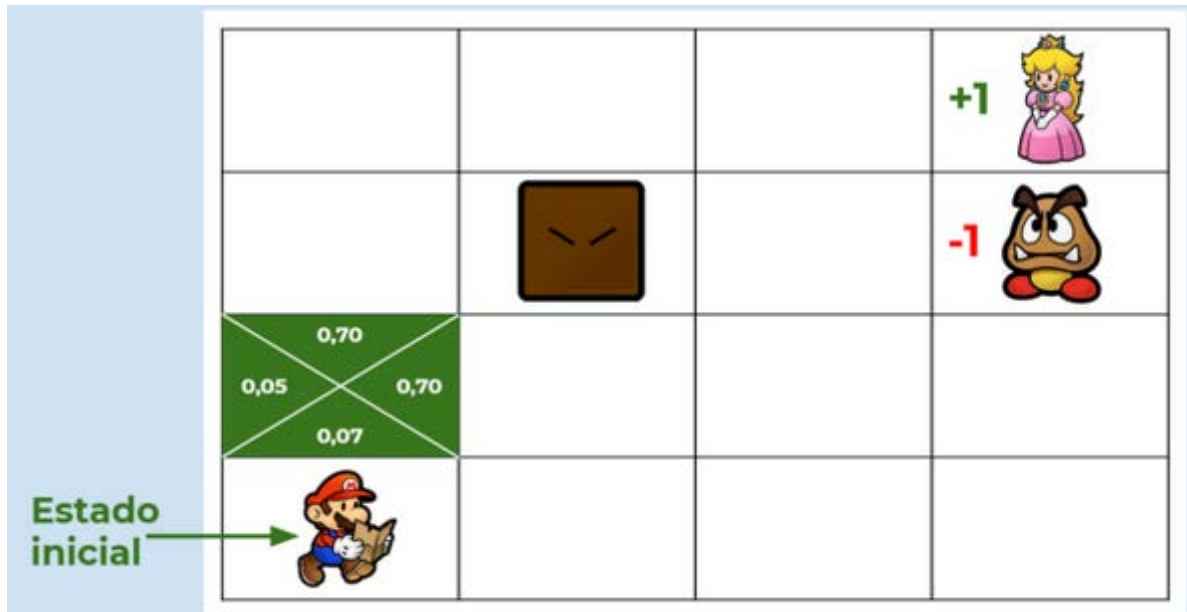
Figura 28. Matriz do jogo do Mario com todas as ações possíveis em todos os estados.



Fonte: Vasconcellos, 2018.

É preciso considerar também que cada estado vai ser representado por uma recompensa diferente. Este mesmo modelo de recompensa não pode ser aplicado no estado de posição 4 x 4. É claro que esses valores também podem ser otimizados ao longo dessas posições para obrigar você a pegar o menor ou o melhor caminho, claro que sempre querendo chegar ao objetivo final; se for possível fazer isto maximizando os ganhos, melhor ainda.

Figura 29. Matriz do jogo do Mario com a posição 3 x 1 mostrando os graus de recompensa por cada possível ação.



Fonte: Vasconcellos, 2018.

Que tal você mesmo tentar definir alguns valores a cada um dos estados do nosso exemplo e traçar caminhos diferentes para calcular o valor de sua recompensa? É tipo um jogo de labirinto, mas que você já sabe o caminho que o levará à recompensa, você só não sabe se vai optar pelo que lhe concede a melhor recompensa.

CAPÍTULO 3

PROCESSOS DE DECISÃO DE MARKOV PARCIALMENTE OBSERVÁVEIS

Um processo de decisão de Markov parcialmente observável (POMDP) é uma generalização de MDPs na qual o estado atual do sistema não é obrigatoriamente conhecido. Pelo contrário, o agente recorda das ações que ele efetuou e das observações que reparou ao longo do tempo, tentando utilizar ainda essas informações para perceber sua próxima decisão. Por exemplo, é provável que ao contrário de um “estado atual do sistema”, uma distribuição de possibilidades seja mantida durante o tempo em que as decisões são tomadas. POMDPs são mais difíceis de solucionar que os MDPs, no entanto, mas são mais significativos.

Um POMDP pode ser utilizado para moldar problemas em diversas áreas. Cassandra (1998) mostrou oportunidades em manutenção de máquinas, navegação de robôs, controle de elevadores, visão computacional, modelagem de comportamento em ecossistemas, aplicações militares, diagnóstico médico, educação e diversas outras áreas. Podemos evidenciar casos notáveis de aplicação, como o trabalho de Hauskrecht (1997) que modelou doenças isquêmicas do coração; enquanto, o trabalho de Pineau (2004) usou POMDPs para moldar o comportamento de um robô que ajudou idosos a se recordarem de seus compromissos, acompanhando-os e guiando-os em (de maneira limitada) diálogos. Finalmente, podemos citar o trabalho de Poupart (2005), que modelou um sistema que acompanhava o comportamento de pacientes com demência, utilizando uma câmera para monitorá-los, além de ajudá-los com os passos para lavar as mãos.

Para melhor esclarecer como funciona um POMDP, podemos considerar o problema a seguir:

Numa sala com duas portas, encontra-se uma pessoa. Em uma das portas encontra-se um tigre, do qual a pessoa deve esquivar-se; na outra porta encontra-se uma saída segura do local onde a pessoa se encontra. A pessoa pode escolher entre três ações possíveis, certo número de vezes:

- » ouvir através da porta (buscando precisar atrás de qual porta o tigre se encontra);
- » abrir a porta à esquerda;
- » abrir a porta à direita.

Após algum tempo, o agente pode decidir abrir uma das portas; dessa forma, ele deve finalmente deparar com uma das opções: encontrar o tigre ou sair em segurança. Todavia,

ao ouvir através da porta, ele pode alterar seu grau de certeza sobre em qual porta o tigre está escondido. Assim, toda vez que ele ouve, obtém uma das percepções a seguir:

- » o tigre parece estar atrás da porta da esquerda;
- » o tigre parece estar atrás da porta da direita.

Essas percepções podem não condizer com a realidade, porque a pessoa pode ter se equivocado ao tentar decidir de que porta vem o som produzido pelo tigre. Todavia, ele pode ouvir através das portas mais vezes, aumentando, desse modo, seu grau de certeza. Como podemos ver, haverá a incerteza quanto ao estado atual do sistema, a pessoa (agente) não tem certeza sobre em qual porta o tigre está; depois de cada ação, uma percepção é concebida, sendo que não necessariamente é informado com total certeza o estado atual. Finalmente, cada ação resultará em uma recompensa, seja ela positiva ou negativa.

3.1. Definição de POMDP

Um POMDP ou processo de decisão de Markov parcialmente observável é uma sequência ordenada (tupla) (S, A, T, R, Ω, O) , em que:

- » S representa um conjunto de estados em que o processo pode se encontrar;
- » A representa um conjunto de ações que podem ser realizadas em diversas épocas de decisão;
- » $T: S \times A \times S \mapsto [0, 1]$ é uma função que oferta a probabilidade de o sistema ir para um estado s' , dado que estava no estado s e a ação executada foi a ;
- » $R: S \times A \mapsto R$ é uma função que oferta a recompensa por tomar uma decisão a quando o processo está em um estado s ;
- » Ω é um conjunto de observações que são conseguidas em cada época de decisão;
- » $O: S \times A \times \Omega \mapsto [0, 1]$ é uma função que dá a probabilidade de uma observação ser verificada, dado um estado s e a última ação a executada.

Num POMDP não temos a oportunidade de verificar de forma direta o estado em que o sistema se encontra em certo momento, ou seja, o agente não conhece o estado atual s (de forma análoga ao que acontece num problema construído como MDP). Todavia, cada ação resulta em alguma percepção que é probabilisticamente relacionada ao estado do sistema. O estado atual do sistema não é acessível ao agente; dessa forma, é possível usar o histórico anterior de ações e percepções para escolher a melhor ação.

3.2. Estado de informação

O estado de informação l_k representa o conhecimento que se tem sobre o sistema na época de decisão k e consiste de:

- » uma distribuição de probabilidades sobre os estados na primeira época de decisão, denotado por b_o ;
- » o histórico completo de ações e observações, desde a primeira época de decisão.

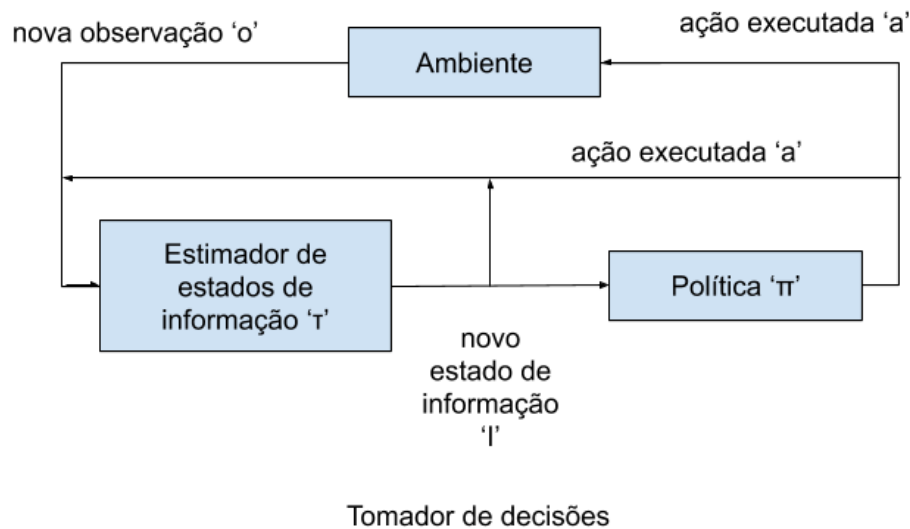
3.3. Dinâmica de sistemas modelados como POMDPs

Podemos observar para entender melhor como funciona a dinâmica de um sistema que tem a modelagem do POMDP. Simplificando essa modelagem, vemos na Figura 30 que, a cada época de decisão, o responsável por tomar decisões observa e confere o estado de informação atual ' l ', como sendo a última ação executada e a última observação compreendida do ambiente.

Estes dados servem como base para que um estimador de estados de informação ' τ ' defina um novo estado de informação, que, neste caso, é utilizado como entrada para uma política ' π ', que, como já vimos anteriormente, determina a próxima ação a ser tomada pelo agente.

A definição de agente foi bastante debatida ao longo deste curso, como base do estudo da inteligência artificial, e claro, sabe-se que a ação do agente tem efeito sobre o ambiente, que, em decorrência, retorna uma observação ' o '. Na época consequente de decisão, esta observação, assim como a última ação executada são utilizadas para determinar o próximo estado de informação. Ainda analisando a Figura 30, vemos que nela há uma aresta levando o novo estado de informação de ' τ ' a ' τ ' para enfatizar o fato de que o estado de informação da época de decisão anterior é utilizado para determinar o da próxima época de decisão. Literalmente há uma dependência de novos resultados embasados em resultados anteriores.

Figura 30. Funcionamento de um sistema modelado com POMDP.



Fonte: Pellegrini e Wainer (2007).

Portanto, podemos concluir que a maneira mais direta de apresentar os estados de informação é como uma série de ações e observações que são executadas a partir da primeira época de decisão. Isso significa que isto é válido além da distribuição inicial de probabilidades sobre aqueles possíveis estados.

O estado de informação na época de decisão 'k' poderia ser representado por:

$$I_k = (b_o, a_o, o_o, a_1, o_1, \dots, a_{k-1}, o_{k-1})$$

Então, podemos dizer que b_o é a distribuição inicial de probabilidades em relação aos estados. A modificação de um estado de informação é representada desta maneira, que, de certa forma, é básica. No caso, seria preciso adicionar o par (a, o) em relação à última ação e a sua observação equivalente (Pellegrini e Wainer, 2007).

Uma política para um POMDP deve mapear estados de informação em ações.

Podemos formular o problema do tigre citado anteriormente como um POMDP:

- » S : há dois estados, *tigre_esq* e *tigre_dir*;
- » A : há três ações, *ouvir*, *abrir_esq* e *abrir_dir*;
- » T : nenhuma ação pode alterar o estado do sistema (mesmo que o agente queira abrir portas ou ouvir através delas, não fará o tigre mudar de lugar!);
- » R : há uma recompensa positiva igual a 30 para abrir a porta onde o tigre não se encontra, e uma recompensa negativa (um custo igual a 100) para abrir a porta onde o tigre está;

- » Ω : há duas observações, `tigre_esq` e `tigre_dir`;
- » O : a ação ouvir dá a observação equivalente ao estado atual (ou seja, a observação “correta”) com probabilidade 0.9, e a observação oposta com probabilidade 0.1. As ações abrir portas dão sempre a observação correta.

O problema modelado dessa forma pode ser resolvido por algoritmos que determinam uma política ótima para POMDP. Essa política recebe como entrada o estado de informação atual, e a saída segue a possibilidade de uma ação que pode maximizar a recompensa esperada pelo agente.

3.4. Política para um POMDP

A política para um POMDP pode ou não ser markoviana, determinística e estacionária, de forma análoga a políticas para MDPs. Para um POMDP de crença, uma política pode ser entendida como uma política para o MDP sobre estados de informação. Definiremos a seguir uma política para POMDP markoviana com relação aos estados de informação, mas não markoviana com relação aos estados do POMDP, como descrito anteriormente.

Definição: Dado um POMDP $P = (S, A, T, R, \mathcal{O}, O)$, uma regra de decisão P em uma época de decisão k é uma função $d_k: I \mapsto A$ que determina a ação, dado um estado de informação P .

Uma política para um POMDP é um conjunto $\pi = \{d_0, d_1, \dots, d_{Z-1}\}$ de regras de decisão para P .

É possível definir políticas reativas para POMDPs, onde o agente verifica apenas a percepção mais recente de forma a selecionar a próxima ação (fazendo-se exclusivamente markovianas, uma vez que não dependem do histórico de ações e observações passadas). Basta que as regras de decisão sejam definidas como funções $d_k: \Omega \mapsto A$ que apontam uma ação, dada uma percepção. As políticas reativas podem ter comportamento inferior à das políticas que utilizem o estado de informação. Singh, Jaakola e Jordan (1994) mostraram, em seu trabalho, que políticas reativas determinísticas podem ser arbitrariamente piores que políticas reativas estocásticas.

Os mesmos critérios de otimalidade usados para MDPs podem ser usados para POMDPs, e a definição de política ótima para POMDP é semelhante para MDPs, a menos por um episódio: a política para POMDPs deve mapear estados de informação, e não estados, em ações. Igualmente, funções valor são definidas para POMDPs, mapeando estados de informação em valores.

REFERÊNCIAS

- ABBEEL, P.; ANDREW, Y. Apprenticeship learning via inverse reinforcement learning. **Proceedings of the twenty-first international conference on Machine learning**. ACM, 2004.
- BARTO, A. G.; SUTTON, R. S.; ANDERSON, C. W. Neuronlike adaptative elements that can solve difficult learning control problems. **IEEE Transactions on Systems, Man and Cybernetics**, v. 3, n. 5, p. 834-846, 1983.
- BATISTA, A. F. de M. **Processos de decisão de Markov**. 2008. Disponível em: <http://alepho.info/cursos/ia/trab/andre/8/ExercicioMDP.pdf>.
- BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: a review and new perspectives. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 35, n. 8, p. 1.798-1828, Aug 2013.
- BERTSEKAS, D. P.; TSITSIKLIS, J. N. Neuro-Dynamic Programming. **Athena Scientific**. Belmont, M.A., 1996.
- BERTSEKAS, D. P.; YU, H. **Q-learning and enhanced policy iteration in discounted dynamic programming**. In: Decision and Control (CDC), 2010, 49th, IEEE Conference on. p. 1.409-1.416.
- CABRAL, D. **Aprendizado por reforço – um conto de Natal**. (2018). Disponível em: <https://www.deviant.com.br/noticias/aprendizado-por-reforco-um-conto-de-natal/> Acesso em: 30 abr. 2019.
- CARVALHO, A. C. P. L. F.; BRAGA, A. P.; LUDEMIR, T. B. Fundamentos de redes neurais artificiais. **XI Escola Brasileira de Computação**, 1998.
- CASSANDRA, A. R. **A survey of POMDP applications**. Presented at the AAAI Fall Symposium, 1998.
- CLEMEN, R. T. **Making hard decisions**: an introduction to decision analysis. 2. ed. Belmont: Duxbury, 1996.
- COPELAND, M. **What is the difference between artificial intelligence, machine learning, and deep learning?** (2016). Disponível em: <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>. Acesso em: 4 abr. 2019.
- DAHL, G. E.; SAINATH, T.; HINTON, G. E. **Improving deep neural networks for LVCSR using rectified linear units and dropout**. 2013. Disponível em: <https://ieeexplore.ieee.org/document/6639346>. Acesso em: 27 maio 2019.
- DEAN, J. *et al.* Large scale distributed deep networks. In: **Proceedings of the 25th International Conference on Neural Information Processing Systems**, NIPS'12, p. 1.223-1.231, 2012.
- DENG, C.; ER, M. J. Real-time dynamic fuzzy Q-learning and control of mobile robots. In: **Control Conference**, 2004. 5th Asian. Vol. 3. p. 1.568-1.576.
- DEVARAKONDA, M.; TSOU, C.-H. Automated problem list generation from electronic medical records in IBM Watson. In **Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence**, AAAI'15, p. 3.942-3.947. AAAI Press, 2015.
- DEEPMIND. **AlphaGo**. Disponível em: <https://deepmind.com/research/case-studies/alphago-the-story-so-far>. Acesso em: 30 maio 2020.
- ELMAN, J. L. Finding Structure in Time. **Cognitive Science**, n. 14, p. 179-211, 1990.

- GARCIA, A. B.; VEZHNEVETS, A.; FERRARI, V. An active search strategy for efficient object class detection. In **2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 3.022-3.031. IEEE, 2015.
- GERA, D. L. **Ancient greek ideas on speech, language and civilization**. [S.l.]: Oxford University Press, 2003.
- GOLMAKANI, A.; SILVA, A. A.; FREIRE, E. M. S.; BARBOSA, M. K.; CARVALHO, P. H. G.; ALVES, V. L. **Cadeias de Markov**, 2014. Disponível em: <http://www.im.ufal.br/evento/bsbm/download/minicurso/cadeias.pdf>. Acesso em: 27 maio 2019.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. MIT Press, 2016.
- HAUSKRECHT, M. Dynamic decision making in stochastic partially observable medical domains: Ischemic heart disease example. In: KERAVNOU, E. *et al.* (Ed.). **6th Conference on Artificial Intelligence in Medicine**. [S.l.]: Springer, 1997. (Lecture Notes in Artificial Intelligence, v. 1211), p. 296-299.
- HAUSKRECHT, M. **Planning and control in stochastic domains with imperfect information**. Tese (Doutorado). EECS, Massachusetts Institute of Technology, 1997.
- HAYKIN, S. **Neural networks: a comprehensive foundation**. [S.l.]: Prentice Hall PTR, 1994.
- HINTON, G.; DENG, L.; YU, D.; DAHL, G. E.; MOHAMED, A.-r.; JAILTY, N.; SENIOR, A.; VANHOUCHE, V.; NGUYEN, P.; SAINATH, T. N.; and KINGSBURY, B. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. **IEEE Signal Processing Magazine**, 29(6):82-97, 2012.
- HINTON, G; *et al.* **Improving neural networks by preventing co-adaptation of feature detectors**. 2012. Disponível em: <https://arxiv.org/pdf/1207.0580.pdf>. Acesso em: 27 maio 2019.
- HOCHREITER, S.; BENGIO, Y.; FRANCONI, P. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: KOLEN, J.; KREMER, S., editors, **Field Guide to Dynamical Recurrent Networks**. IEEE Press, 2001.
- JAIN, S. **An Overview of Regularization Techniques in Deep Learning (with Python code)**. 2018. Disponível em: <https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/>. Acesso em: 27 maio 2019.
- JONES, M. T. **Um mergulho profundo nas redes neurais recorrentes**, 2017. Disponível em: <https://imasters.com.br/data/um-mergulho-profundo-nas-redes-neurais-recorrentes>. Acesso em: 27 maio 2019.
- KAEHLING, L. P; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: a survey. **Journal of Artificial Intelligence Research**, V. 4, p. 237-285, 1996.
- KLOPF, A. H. **Brain function and adaptive systems**. A heterostatic theory. Bedford, Massachusetts: Air Force Cambridge Research Laboratories, 1972.
- KOFINAS, P.; DOUNIS, A. I. Fuzzy Q-Learning agent for online tuning of PID controller for DC motor speed control. **Algorithms**, v. 11, 2018.
- KRIZHEVSKY, A; SUTSKEVER, I; HINTON, G. E. **ImageNet classification with deep convolutional neural networks**. **Advances in Neural information processing systems**, 2012.
- KUZMIN, V. **Connectionist Q-learning in robot control task**. 2002.
- LANDELIUS, T. **Reinforcement Learning and Distributed Local Model Synthesis**. PhD thesis. Linköping University, Sweden. SE-581 83 Linköping, Sweden. Dissertation n. 469, 1997.

REFERÊNCIAS

- LANGE, S.; RIEDMILLER, M. Deep auto-encoder neural networks in reinforcement learning. *In Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN)*, p. 1-8, 2010.
- LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2.278-2.324, 1998.
- LIU, C.; CAO, Y.; LUO, Y.; CHEN, G.; VOKKARANE, V.; MA, Y.; CHEN, S.; HOU, P. A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. *IEEE Transactions on Services Computing*, PP(99):1-13, 2017.
- LOPES, R. A. S.; BRAGA, V. G. de M.; LAMAR, M. V. **Sistema baseado em redes neurais e q-learning para jogar jogos eletrônicos**. 2017.
- MATARIC, M. J. **Introdução à robótica**. [S.l.]: UNESP, 2014.
- MAYER, Z. D. **Advanced Deep Learning with Keras in Python**. 2019. Disponível em: <https://www.datacamp.com/courses/advanced-deep-learning-with-keras-in-python>. Acesso em: 27 maio 2019.
- McCULLOCK, J. **Q-Learning**.*, 2012. Disponível em: <http://mnemstudio.org/path-finding-q-learning-tutorial.htm>. Acesso em: 4 abr. 2019.
- MEDEIROS, F. G. Neto; PINTO, R. F. Júnior; ROCHA, M. G. O; SÁ, J. J. M. Júnior; PAULA, I. C. Júnior. Aprendizado profundo: conceitos, técnicas e estudo de caso de análise de imagens com Java. **III Escola Regional de Informática do Piauí**. Livro Anais – Artigos e Minicursos, v. 1, n. 1, p. 465-486, jun., 2017.
- MNIH, V. *et al*. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 518, n. 7540, p. 529-533, fev. 2015.
- MOHAMED, A.; DAHL, G.; HINTON, G. Deep belief networks for phone recognition. **NIPS Workshop on deep learning for speech recognition and related applications**, 2009.
- MOREIRA, S. **Rede Neural Perceptron Multicamadas**, 2018. Disponível em: <https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas-f9de8471f1a9>. Acesso em: 27 maio 2019.
- MOTA, I. C., (2018). **Aprendizado por reforço utilizando Q-Learning e redes neurais artificiais em jogos eletrônicos**. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-n4, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 50p.
- MOUJAHID, A. **A practical introduction to deep learning with Caffe and Python**, 2016. Disponível em: <http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/>.
- NAIR, V.; HINTON, G. E. Rectified Linear Units improve Restricted Boltzmann Machines. In **Proceedings of the 27th International Conference on Machine Learning (ICML-10)**, pages 807-814, 2010.
- NIELSEN, M. A. **Neural networks and deep learning**. Determination Press, 2015.
- PELLEGRINI, J.; WAINER, J. On the use of POMDPs to model diagnosis and treatment of diseases. *In: Encontro Nacional de Inteligência Artificial (ENIA)*. Campinas, SP, Brazil: Sociedade Brasileira de Computação, 2003.
- PELLEGRINI, J.; WAINER, J. **Processos de Decisão de Markov: um tutorial**. Instituto de Computação, Unicamp, Campinas-SP, Brazil. **BITA**. Volume XIV. Número 2. 2007.
- PENG, J.; WILLIAMS, R. J. Incremental multi-step Q-learning. *In: Machine Learning*. Morgan Kaufmann, p. 226-232, 1996.

- PINEAU, J. **Tractable Planning under uncertainty**: exploiting structure. Tese (Doutorado). Robotics Institute, Carnegie-Mellon University, 2004.
- PIPE, A. G. An architecture for building “potential field” cognitive maps in mobile robot navigation. *In: Systems, man and cybernetics*, 1998. IEEE International Conference on. Vol. 3. p. 2.413-2.417.
- POUPART, P. **Exploiting structure to efficiently solve large scale partially observable Markov decision processes**. Tese (Doutorado). University of Toronto, 2005.
- PUTERMAN, M. L. **Markov decision processes**: discrete stochastic dynamic programming. New York, NY: Wiley-Interscience, 1994.
- ROBIN, J. **Aprendizado por reforço**, 2002. Disponível em: www.cin.ufpe.br/~compint/aulas-IAS/ias/ias-021/rl.ppt. Acesso em: 27 maio 2019.
- RUSSELL, S.; NORVIG, P. **Artificial intelligence**: a modern approach. 3. ed. New Jersey: Pearson Education, 2010.
- SANTOS, M. R.; DACORSO, A. L. R. (2016). Intuição e racionalidade: um estudo sobre tomada de decisão estratégica em empresas de pequeno porte. **Rev. Adm. UFSM**, Santa Maria, v. 9, número 3, p. 448-463, jul.-set. 2016.
- SCHMIDHUBER, J. Deep learning in neural networks: An overview. **Neural networks**, 61:85-117, 2015.
- SIEBEL, N. T; SOMMER, G. Evolutionary Reinforcement Learning of Artificial Neural Networks. **International Journal of Hybrid Intelligent Systems**. p. 171-183, 2007.
- SIKCHI, H. **Towards Safe Reinforcement Learning**. 2018. Disponível em: <https://medium.com/@harshitsikchi/towards-safe-reinforcement-learning-88b7caa5702e>. Acesso em: 27 maio 2019.
- SILVA, T. L; GOUVÊA, M. M. Aprendizado por reforço clássica e conexionista: análise de aplicações. **Anais do EATI – Encontro Anual de Tecnologia da Informação**. Instituto Politécnico – Pontifícia Universidade Católica de Minas Gerais, p. 299-302, 2015.
- SILVER, D., *et al.* Mastering the game of Go with deep neural networks and tree search. **Nature**, 529:484-489, 2016.
- SIMON, P. **Too big to ignore**: The Business Case for Big Data. [S.l.]: Wiley, 2013.
- SINGH, S.; JAAKKOLA, T.; JORDAN, M. I. Learning without state-estimation in partially observable markovian decision processes. *In: International Conference on Machine Learning (ICML)*. New Brunswick, NJ, USA: Morgan Kaufmann, 1994.
- SKYMIND. **A Beginner’s Guide to LSTMs and Recurrent Neural Networks**. 2017. Disponível em: <https://skymind.ai/wiki/lstm#feedforward>. Acesso em: 27 maio 2019.
- SRIVASTAVA, N. *et al.* Dropout: a simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**. 2014, vol. 15, p. 1.929-1.958.
- SUTTON, R. S. **Temporal credit assignment in reinforcement learning**. Tese de doutorado, University of Massachusetts Amherst, 1984.
- SUTTON, R. S. **Learning to predict by the method of temporal differences**. Machine Learning, vol. 3, p. 9-44, 1988.
- SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: an introduction**, 2. ed. MIT Press, Cambridge, Massachusetts, EUA, 2018.

REFERÊNCIAS

- SUTTON, R. S.; SATINDER, S. P. **Reinforcement learning with replacing eligibility traces**. Cambridge: Dept. of Computer Science, MIT, 1996.
- SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. *In: Proceedings of 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 1-9, 2015.
- TCH, A. **The mostly complete chart of Neural Networks, explained**, 2017. Disponível em: <https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>. Acesso em: 27 maio 2019.
- TSITSIKLIS, J. N. **Asynchronous stochastic approximation and Q-learning**. Boston: Kluwer Academic Publishers, 1994.
- VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. **Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres**. *In: Proceedings of the XXIX Conference on Graphics, Patterns and Images*, p. 1-4. Sociedade Brasileira de Computação, 2016.
- VASCONCELLOS, P. **Explicando Deep Reinforcement Learning com Super Mario ao invés de matemática**. 2018. <https://paulovasconcellos.com.br/explicando-deep-reinforcement-learning-com-super-mario-ao-inv%C3%A9s-de-matem%C3%A1tica-4c77392cc733>.
- WAIBEL, A. *et al.* Phoneme Recognition Using Time-Delay Neural Networks. **IEEE Transactions on Acoustics, Speech and Signal Processing**, Volume 37, n. 3, p. 328-339, 1989.
- WATKINS, C. J. C. H. **Learning from delayed rewards**. PhD Thesis. University of Cambridge. Cambridge, England, 1989.
- WATKINS, C. J. C. H.; DAYAN, P. Q-learning. *In: Machine Learning*. Vol. 8, p. 279-292, 1992.
- WIENER, N. **The human use of human beings: cybernetics and society**. [S.l.]: Free Association Books, 1989.
- Watkins, C. J. Models of delayed reinforcement learning, Master's thesis, PhD thesis, Psychology Department, Cambridge University, Cambridge, United Kingdom. 1989
- Mitchell, T. M. Does Machine Learning Really Work?. *AI Magazine*, 18(3), 11. 1997.