2
Aprendizado de Máquina

Desde que os computadores foram inventados, sempre nos perguntamos se eles poderiam ser feitos para aprender. Se pudéssemos compreender como programá-los para aprender e melhorar automaticamente com experiência, o impacto seria dramático.

Ainda mão se sabe como fazer os computadores aprenderem tão bem como as pessoas aprendem. No entanto, os algoritmos que foram inventados são eficazes para certos tipos de tarefas de aprendizagem e uma compreensão teórica da aprendizagem está começando a surgir. Muitos programas de computador práticos foram desenvolvidos para exibir tipos úteis de aprendizagem, e importantes aplicações comerciais começaram a aparecer, tais como em reconhecimento de voz, mineração de dados, controle de processos, entre outros. Como a nossa compreensão dos computadores continua a amadurecer, parece inevitável que a aprendizagem de máquina irá desempenhar um papel cada vez mais central em ciência da computação e informática (Mitchell, 1997).

Um conceito simples de aprendizado de máquina se mostra na Figura 2, quando um ambiente fornece alguma informação ao elemento de aprendizagem. O elemento de aprendizagem utiliza esta informação para melhorar numa base se conhecimentos e finalmente este processo vai melhorando seu desempenho perfeitamente (Haykin, 2007).

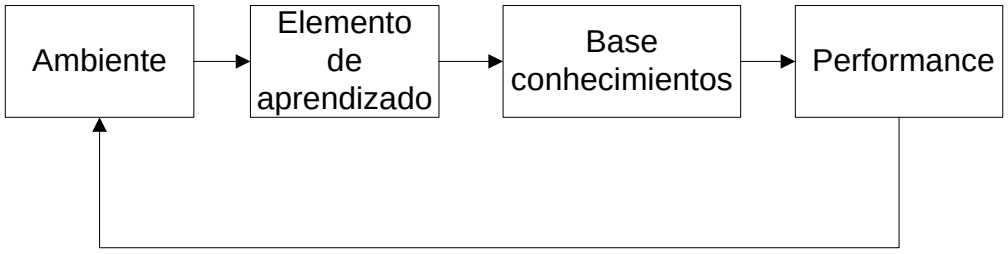

Figura 2 – Modelo simples de um aprendizado de maquina.

2.1.
Tipos de Aprendizado

A característica do aprendizado pode dotar aos agentes de capacidade de desenvolver sua autonomia, pois com o aprendizado o agente está sempre apto a se adaptar aos ambientes. Nesta seção se apresenta de forma breve a classificação do aprendizado segundo a forma de Realimentação (Weiss, 1999).

2.1.1.
Aprendizado Supervisionado

A realimentação especifica a atividade desejada pelo agente e o objetivo do aprendizado é tornar a ação executada e a ação desejada tão próxima quanto possível. A partir da entrada a rede realiza seu processamento e a saída obtida é comparada com a saída esperada. Caso não sejam iguais, um processo de ajuste de pesos é aplicado buscando-se um erro mínimo ou aceitável. O algoritmo de aprendizado supervisionado mais comum é o Backpropagation. O diagrama de blocos do aprendizado supervisionado é apresentado na Figura 3.

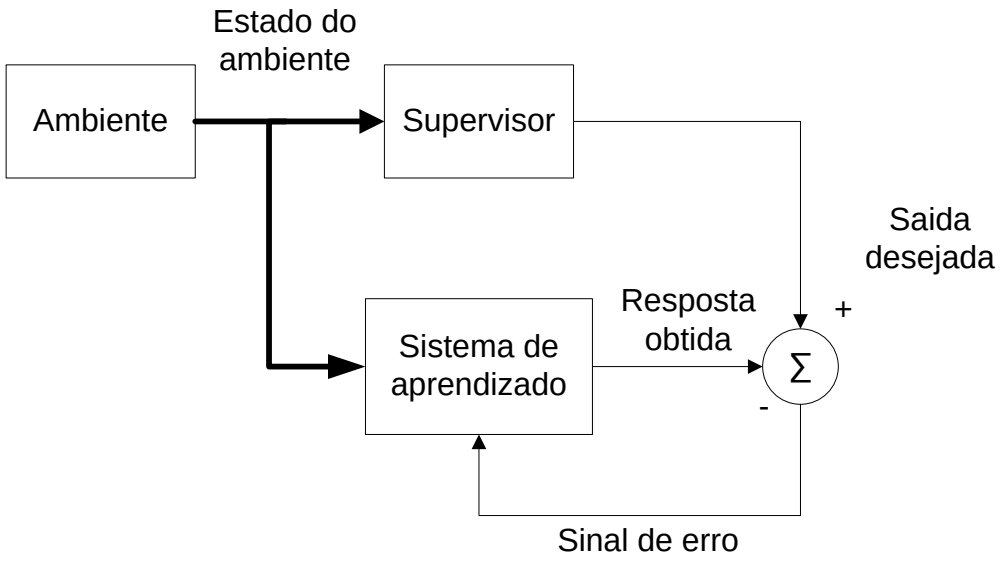

Figura 3 – Diagrama de blocos do aprendizado supervisionado.

Primeiro defini-se o conhecimento como um conjunto de exemplos de entradas e saídas, o supervisor tem o conhecimento do ambiente, este é totalmente desconhecido para a rede neural. Com objetivo de construir o conhecimento o

supervisor entrega para a rede neural a saída desejada (vetor de treinamento), a resposta desejada é a ação ótima a ser melhorada. Os parâmetros da rede neural são ajustados sobre influencia da combinação do vetor de treinamento e a sinal de erros. O sinal de erro é definido pela saída desejada e a resposta obtida por a rede, o ajuste é iterativo passo a passo com o objetivo de emular o supervisor. A emulação do supervisor é presumida com algum senso estatístico. Então o conhecimento do supervisor é transferido para a rede neural (Haykin, 2007).

2.1.2. Aprendizado por Reforço

A realimentação é um valor ou sinal que especifica a utilidade da atividade ou ação realizada pelo agente quando ele se encontra em um determinado estado. O objetivo do aprendizado é maximizar esta utilidade para todos os estados. Mais detalhes sobre aprendizado por reforço são apresentados na seção 2.3.

2.1.3. Aprendizado não Supervisionado

Nenhuma realimentação explícita é feita e o objetivo é descobrir atividades úteis e desejadas através de tentativa-e-erro e processos auto-organizáveis. Em aprendizado não supervisionado ou auto-organizado não existe um supervisor ou algum critico no processo de aprendizagem. Utiliza um sistema de classificação, não existe saída desejada. A rede é treinada através de excitações ou padrões de entrada e então, arbitrariamente, organiza os padrões em categorias. Para uma entrada aplicada à rede, será fornecida uma resposta indicando a classe a qual a entrada pertence. Se o padrão de entrada não corresponde às classes existentes, uma nova classe é gerada (Haykin, 2007). Um diagrama de blocos se mostra na Figura 4.

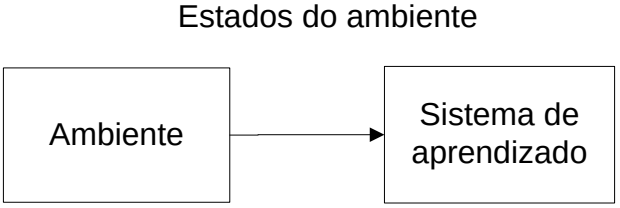

Figura 4 – Diagrama de blocos do aprendizado não supervisionado.

2.2.
Redes Neurais

Uma rede neural artificial é um modelo maciçamente paralelo constituído por simples unidades de processadores, que tem uma propensão natural para armazenar conhecimento experimental e torná-lo disponível para seu uso. Assemelha-se ao cérebro em dois aspectos (Haykin, 2007): (i) O conhecimento é adquirido por seu ambiente através de um processo de aprendizagem; (ii) Pesos associados às interconexões entre neurônios, conhecidos como pesos sinápticos, são utilizados para armazenar o conhecimento adquirido.

As redes neurais são compostas por diversos elementos processadores (neurônios artificiais), altamente interconectados, que efetuam operações simples, transmitindo seus resultados aos processadores vizinhos. A habilidade das redes neurais em realizar mapeamentos não-lineares entre suas entradas e saídas as tem tornado prósperas no reconhecimento de padrões e na modelagem de sistemas complexos (Bishop, 1995) (Almeida, 2008).

Pode-se encontrar muitos tipos de redes neurais, com diferentes arquiteturas e algoritmos de aprendizado (como explicados na seção anterior). Embora exista uma grande quantidade de arquiteturas de redes neurais, a estrutura multicamada (*MLP –Multilayer Perceptron*) é a mais aplicada com sucesso para resolver problemas diversos e com grau de dificuldade, também conhecida e utilizada devido à capacidade de aproximação universal.

Deste modo, neste trabalho, optou-se por avaliar o desempenho da arquitetura *multilayer perceptron feedforward*. A característica *Feedforward* devido ao tipo de estrutura de interconexão, no sentido que o fluxo de dados é sempre em uma única direção. O algoritmo de aprendizado utilizado foi o Backpropagation, no qual se utiliza um conjunto de exemplos de entrada-saída para se efetuar o aprendizado supervisionado. A partir das entradas apresentadas, a rede realiza seu processamento e a saída obtida é comparada com a saída esperada. A partir do erro obtido, um processo de ajuste de pesos é aplicado, buscando-se minimizar o erro, conforme detalhado nas seções subseqüentes (Haykin, 2007) (Almeida, 2008).

2.2.1.
Redes Multilayer Perceptron

A arquitetura *Multilayer Perceptron (MLP)* é a mais utilizada nas aplicações de engenharia, a arquitetura MLP é organizada em diversas camadas: uma camada de entrada, formada pelos neurônios que estão conectados às entradas da rede, isto é, recebem os atributos de entrada; uma camada de saída, contendo os neurônios que apresentam as saídas da rede neural ao ambiente externo; e uma ou mais camadas intermediárias (ou escondidas), compostas de neurônios cujas entradas e saídas estão conectadas somente a outros neurônios, não havendo interação com o ambiente externo à rede.

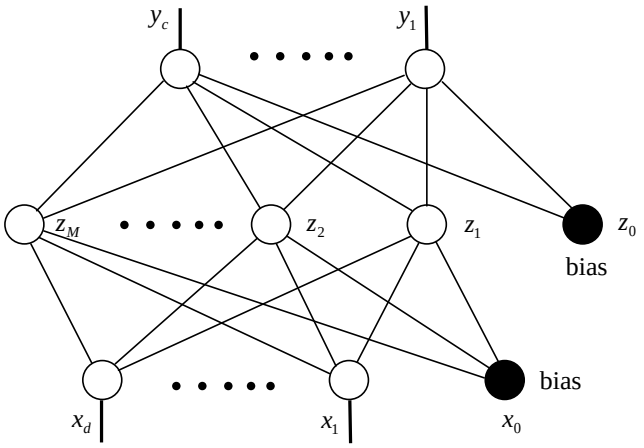

Figura 5 – Representação geral da arquitetura multilayer perceptron.

De modo geral, esta arquitetura possui um vetor de entradas **x** de dimensão d $x = [x_1, x_2, \dots, x_d]$; um vetor de c saídas $y = [y_1, y_2, \dots, y_c]$; e M neurônios na camada escondida (pode possuir mais de uma camada escondida).

Todos os parâmetros adaptáveis (pesos e *bias*) desta arquitetura são agrupados convenientemente em um único vetor w-dimensional $w = [w_1, w_2, \dots, w_w]$ para facilitar tratamentos analíticos.

Além disso, para o cálculo (treinamento) dos parâmetros adaptáveis $w = [w_1, w_2, \dots, w_w]$, utiliza-se um conjunto de observações (dados de treinamento) $D = \{(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \dots, (x^{(N)}, t^{(N)})\}$ de algum processo a ser modelado,

onde $x = \left[x^{(1)}, x^{(2)}, \ldots, x^{(N)}\right]$ é o conjunto de dados de entrada e $t = \left[t^{(1)}, t^{(2)}, \ldots, t^{(N)}\right]$ é o conjunto de dados contendo a saída desejada (Haykin, 2007) (Túpac, 2005).

2.2.2. Algoritmo de Aprendizado Backpropagation

A idéia principal desde algoritmo de aprendizado é que os erros das unidades da camada escondida sejam determinados retro-propagando os erros da camada de saída; motivo pelo qual, este método toma o nome de regra de aprendizado *back-propagation*.

Este algoritmo pode ser aplicado em redes com qualquer número de camadas escondidas. Cabe ressaltar que, para uma rede *feed fordward* foi mostrado que uma única camada escondida é suficiente para aproximar com precisão arbitrária qualquer função contínua, desde que haja uma quantidade suficiente de processadores na camada escondida e que a função de ativação destes processadores seja não linear. Na maioria das aplicações, usa-se uma rede *feed-forward* com uma única camada escondida e com função de ativação sigmóide nos neurônios da camada escondida e com função de ativação linear nos neurônios da camada de saída.

O algoritmo funciona em duas fases:

1. O p-ésimo padrão de entrada $x^{(p)}$ é inserido e propagado em sentido direto através de toda a rede para calcular os valores de saída $y_0^{(p)}$, onde $y_0^{(p)}$, é o valor de saída do processador *o* da camada de saída para a entrada do padrão $x^{(p)}$.

2. A saída $y_0^{(p)}$ é comparada com o valor desejado $t_o^{(p)}$, obtendo-se um valor de erro $\delta_o^{(p)}$ em cada neurônio na camada de saída. Esta fase envolve a propagação em forma reversa do valor do erro através de toda a rede e o cálculo da atualização dos pesos em todos os processadores como segue:

- O peso de uma conexão w_{kj} é ajustado em um valor proporcional ao produto do erro no processador *k* que recebe a entrada y_j e a saída do processador *j* que enviou o sinal através da sua conexão:

$$\Delta_p w_{kj} = \gamma \delta_k^{(p)} y_j^{(p)} \qquad (2.1)$$

Onde

w_{kj}	Peso da conexão entre processadores *j* e *k*
γ	Taxa de aprendizado
$\delta_k^{(p)}$	Erro no processador *k* para o padrão $x^{(p)}$
$y_j^{(p)}$	Saída do processador j para o padrão $x^{(p)}$

Se o processador *k* pertence à camada de saída *o*, o erro é :

$$\delta_o^{(p)} = \left(t_o^{(p)} - y_o^{(p)}\right) F'\left(s_o^{(p)}\right) \tag{2.2}$$

Onde

$t_o^{(p)}$	Valor desejado do processador de saída o para o padrão de entrada $x^{(p)}$
$F'\left(s_o^{(p)}\right)$	1ª derivada da função de ativação da soma ponderada $s_o^{(p)}$ de todas as entradas do neurônio de saída o para o padrão $x^{(p)}$

- Dado que a função de ativação $F(\cdot)$ é sigmoidal, tem-se que:

$$y^{(p)} = F\left(s^{(p)}\right) = \frac{1}{1 + e^{-s^{(p)}}} \tag{2.3}$$

- Calculando a 1ª derivada de $F(s^{(p)})$:

$$F'\left(s^{(p)}\right) = y^{(p)}\left(1 - y^{(p)}\right) \tag{2.4}$$

- Assim, o erro para um processador o da camada de saída pode ser escrito como:

$$\delta_o^{(p)} = \left(t_o^{(p)} - y_o^{(p)}\right) y^{(p)}\left(1 - y_o^{(p)}\right) \tag{2.5}$$

Se o processador k pertence à camada escondida h, o erro é calculado recursivamente a partir dos erros dos processadores aos quais ele está conectado diretamente e dos seus pesos. Para o caso de função de ativação sigmoidal tem-se:

$$\delta_h^{(p)} = F'\left(s_h^{(p)}\right) \sum_{o=1}^{N_o} \delta_o^{(p)} w_{oh} = y_h^{(p)}\left(1 - y_h^{(p)}\right) \sum_{o=1}^{N_o} \delta_o^{(p)} w_{oh} \tag{2.6}$$

Onde:

$\delta_h^{(p)}$	Erro do processador da camada escondida h para o padrão $x^{(p)}$
$F'\left(s_h^{(p)}\right)$	Ativação no processador da camada escondida para o padrão $x^{(p)}$
w_{oh}	Peso da conexão entre o processador da camada escondida e o processador da camada de saída
N_o	Número de processadores da camada de saída

A taxa de aprendizado η é um parâmetro importante a ser definido no aprendizado. Esta não deve ser nem muito pequena, causando um treinamento muito lento, nem muito grande, gerando oscilações. Quando a taxa de aprendizado é pequena, e dependendo da inicialização dos pesos (feita de forma aleatória), a Rede Neural pode ficar presa em um mínimo local. Quando a taxa de aprendizado é grande, a Rede Neural pode nunca conseguir chegar ao mínimo global, pois os valores dos pesos são grandes. A solução para este problema é utilizar uma taxa de aprendizado adaptativa (Almeida, 2008).

Além deste parâmetro, pode-se também utilizar um termo α de momento proporcional à variação no valor do peso sináptico no passo anterior; Com o objetivo de acelerar a convergência, evitando oscilações. A atualização dos pesos fica da seguinte forma (Haykin, 2007):

$$\Delta_p w_{kj}(t+1) = \gamma \delta_k^{(p)} y_j^{(p)} + \alpha \Delta w_{kj}(t) \quad (2.7)$$

Onde:

α	Taxa de momento
$t, t+1$	Ultima e próximas iterações

Como se pode verificar, o algoritmo de aprendizado do Backpropagation tem duas fases, para cada padrão apresentado: Feedforward e Feedbackward.Na primeira etapa, as entradas se propagam pela rede, da camada de entrada até a camada de saída, gerando a saída da rede em resposta ao padrão apresentado. Na segunda etapa, os erros se propagam na direção contrária ao fluxo de dados, indo

da camada de saída até a primeira camada escondida, atualizando os pesos sinápticos. Este procedimento de aprendizado é repetido diversas vezes, até que, para todos os processadores da camada de saída e para todos os padrões de treinamento, o erro seja menor do que o especificado (Almeida, 2008).

Foi demonstrado que o algoritmo Backpropagation é um aproximador universal (Hornik, Stinchcombe e White, 1989), sendo capaz de aprender qualquer mapeamento de entrada-saída, sendo, portanto, útil para modelar processos como será visto no capítulo 4.

A definição do tamanho da rede, isto é, o número de camadas escondidas e número de processadores em cada uma dessas camadas, é um compromisso entre convergência e generalização. Convergência é a capacidade da Rede Neural de aprender todos os padrões do conjunto de treinamento. Generalização é a capacidade de responder corretamente aos padrões nunca vistos (conjunto de teste). O objetivo é utilizar a menor rede possível, de forma a se obter uma boa generalização, que seja capaz de aprender todos os padrões (Feitosa, Vellasco, *et al.*, 1999).

O objetivo do aprendizado Back-Propagation é obter um mapeamento de entrada-saída. Uma rede bem treinada significa que esta aprendeu o modelo suficientemente bem para ter boas estimativas no futuro. A partir desta visão, o processo de aprendizado é uma escolha da parametrização da rede para o conjunto de dados, isto é, escolher, a partir de algumas parametrizações "candidatas", a melhor parametrização segundo algum critério (Túpac, 2005).

Desta forma, a validação cruzada é um princípio atraente onde o conjunto de dados é particionado em dados para treinamento e dados para teste, sendo que os dados de treinamento são subdivididos em dados para estimação e dados para validação.

A idéia é utilizar o conjunto de treinamento para avaliar o desempenho dos modelos candidatos e assim, escolher o melhor. O subconjunto de treinamento permite selecionar o modelo e o subconjunto de validação permite validar o modelo escolhido. Já com o conjunto de testes é verificada a generalização do modelo, isto para evitar o efeito de sobre-ajuste (overfitting). Assim, a validação cruzada é útil tanto para escolher a melhor arquitetura de rede, isto é a seleção de modelos, ou para decidir a hora de finalizar o processo de treinamento (ajuste dos parâmetros) (Stone, 1978) (Stone, 1974)(Túpac, 2005).

2.3.
Aprendizado por Reforço

Uma questão fundamental na área de neurociência comportamental concerne nos processos de decisão, pelo qual animais e seres humanos selecionam ações no sentido da recompensa e punição, e sua neural realização. Na psicologia comportamental, esta questão tem sido investigada em detalhe através dos paradigmas de pavloviano (clássico) e condicionada instrumental (operante), e muitas evidências têm sido acumuladas sobre as associações que controlam aspectos diferentes de comportamento aprendido (Niv, 2009).

O campo computacional o aprendizado por reforço (Sutton e Barto, 1998) apresentou um quadro normativo no qual o comportamento condicionado, como pode ser entendido. Neste, a seleção é a melhor ação com base em previsões de conseqüências futuras ao longo prazo, como a tomada de decisão que auxilia maximizando as recompensas e minimizando a punição (Niv, 2009).

Aprendizado por Reforço é uma abordagem da inteligência artificial na qual é ressaltado o processo de aprendizado em um indivíduo a partir da sua interação tentativa-e-erro com um ambiente dinâmico, ao invés do aprendizado baseado em um supervisor de conhecimento, ou segundo uma modelagem do ambiente inteira, como as abordagens clássicas na Inteligência Computacional (Sutton e Barto, 1998).

O aprendizado por reforço pode ser usado para aprender a controlar um processo, onde, uma vez aplicado a sistemas de agentes autônomos, dele poderá se obter o comportamento desejado. O algoritmo (de aprendizado por reforço) recebe do ambiente o reforço, que é uma avaliação do desempenho atual. Baseado neste reforço, o algoritmo modifica o controlador e tenta evoluir ações que melhor implementem o comportamento esperado.

Existem duas principais estratégias para a solução de problemas de aprendizado por reforço. A primeira é a busca no espaço de comportamentos, a fim de encontrar um bom desempenho do ambiente, a segunda é o uso de técnicas estatísticas e métodos de programação dinâmica para estimar a ação para estimar a utilidade da ação tomada no estado do mundo (Kaelbling, Littman e Morre, 1996).

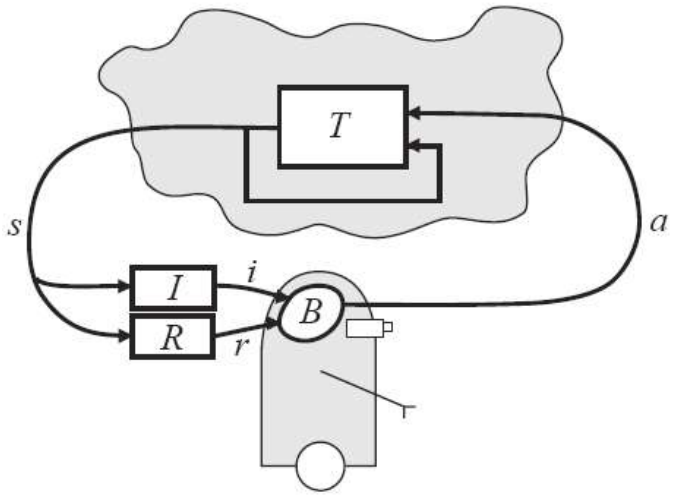

Figura 6 – Diagrama de um sistema básico de aprendizado por reforço.

Aprendizado por reforço consiste em desenvolver o aprendizado do comportamento ((B) na Figura 6) de um agente através da interação com o seu ambiente. As ações executadas têm como conseqüência um sinal de reforço, porém não existe conhecimento prévio da ação correta para cada possível estado. Cabe ao sistema descobrir quais ações levam a um desempenho melhor do sistema. Um modelo de Aprendizado por reforço padrão pode ser visto na Figura 6.

A cada instante de tempo, o agente examina, através da sua percepção, (sensores) o estado *S(I)*. A função *I* define a observabilidade do ambiente. Esta função será considerada como a função identidade, ou seja, os estados apresentam observabilidade total. O agente em um estado *s* deve escolher, baseado em suas observações, uma ação *a* a partir de um conjunto (*A*) discreto de ações disponíveis onde $a \in A$. Após a execução desta ação, o ambiente passa a um estado *s´* com probabilidade que depende da ação *a*, e um sinal de reforço *r* é recebido do ambiente. Este sinal denominado de reforço primário depende do estado anterior *s* e da ação *a*. O agente deve determinar qual ação executar quando está em um estado específico. A função de reforço qualifica uma tarefa realizada pelo agente através de recompensas e punições. Para sistemas que não recebem sinais de reforços primários a cada iteração, premiar ou punir ações passadas que tenham contribuído para o sucesso ou falha do sistema torna-se um problema (*delayed reinforcement)*. Neste tipo de aplicação o sinal de reforço não é conhecido até que o agente atinja o objetivo ou chegue ao final da tarefa. Algumas vezes pode ser

difícil chegar ou identificar o final da tarefa, além de consumir grande quantidade de memória. Para solucionar este problema, o modelo matemático usado é o Processo de Decisão de Markov.

2.3.1. Processos Markovianos

Em aprendizado por reforço, um agente toma decisões baseando-se no estado atual do ambiente. O estado deve incluir as sensações do ambiente imediatas tais como, medições feitas no momento. O estado pode ter mais informação do que medições imediatas, mas não é exigência fornecer toda essa sua informação ao agente. A informação relevante para o agente é a informação imediata e de sensações passadas, sem exceder o histórico completo das sensações passadas. Qualquer sinal fornecido pelo estado que tenha esta característica de informação relevante pode-se dizer que cumpre a Propriedade de Markov (Bellman, 1957) (Tupac, 2000)

Um processo de Markov é um modelo matemático que é utilizado para no estudo de sistemas complexos. O conceito básico de processo de Markov são os *estados* e *transição* de estado. Diz-se que um sistema ocupa um estado quando é descrito completamente por valores de variáveis que definem o estado. O sistema faz transição de estados quando os valores descritos mudam para os valores especificados de um estado a esse especificado para outro (Howard, 1960).

Processos de Decisão de Markov, são também referidos como programação dinâmica estocástica ou problemas de controle estocástico, são modelos para toma de decisões seqüenciais quando os resultados são incertos. Os modelos de processos de decisões de Markov consistem de: Episódios de decisão, estados, ações, retornos e probabilidades de transição. Escolhendo uma ação em um estado gera um retorno (premio) e determina um estado no seguinte episódio de decisão através de uma transição de função de probabilidade. Políticas ou estratégias são prescritas da qual a ação de escolher sobre qualquer eventualidade em todos os episódio de decisão futura. A tomada de decisões procura políticas as quais são ótimas em algum sentido (Puterman, 2005).

2.3.1.1. Propriedade de Markov

Em aprendizado por reforço o agente faz a tomada da decisão como uma função de um sinal proveniente do ambiente, chamado estado do ambiente, o estado é toda informação avaliada pelo agente.

Quando a probabilidade de transição de um estado **s** para um estado **s'** depende apenas do estado **s** e da ação **a** adotada em **s**, isso significa que o estado corrente fornece informação suficiente par ao sistema de aprendizado decidir que ação deve ser tomada , quando o sistema possui estas características diz-se que ele satisfaz a propriedade de Markov (Bellman, 1957) (Figueiredo, 2008).

Premissa: Supõe-se um agente genérico com número de estados e retornos finito. Seja considerada a resposta em $t+1$ (estado seguinte) para uma ação efetuada em t , a resposta depende de todo o que aconteceu anteriormente, segundo a seguinte dinâmica de probabilidades:

$$P_r\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0\} \quad (2.8)$$

Onde a probabilidade que o estado s_{t+1} seja o estado s' (Pr) é uma função do reforço r_{t+1} do estado seguinte que, a sua vez, depende de todos os estados, ações e reforços passados: s_t , a_t , s_{t-1} , a_{t-1} , s_0 , a_0.

Propriedade de Markov: A resposta do ambiente em $t+1$ depende apenas dos estados e reforços em t . Assim, essa resposta tem a expressão dada na eq (2.9).

$$P_r\{s_{t+1} = s', r_{t+1} = r|s_t, a_t|\} \quad (2.9)$$

A Propriedade de Markov é de fundamental importância no aprendizado por reforço uma vez que, tanto as decisões e os valores são supostos como uma função apenas do estado atual, abrindo a possibilidade de métodos de soluções incrementais, isto é, pode-se obter soluções a partir do estado atual e os estados imediatamente sucessivos, como é feito no método de Programação Dinâmica.

2.3.1.2. Processos de Decisão de Markov

Um Processo de Decisão de Markov PDM (*Markovian Decision Process*) (Bellman, 1957)é definido como um conjunto de estados $s \in S$, ações $a \in A(s)$ um conjunto de transições entre estados associadas com as ações e um conjunto de probabilidades P sobre o conjunto S que representa uma modelagem das transições entre os estados. Assim, dado um par de estado e ação (s,a), a probabilidade do estado s passar a um estado s' é:

$$P_{ss'}^{a} = \{s_{t+1} = s' | s_t = s, a_t = a|\} \tag{2.10}$$

Onde Pr é o operador de probabilidade; neste caso representa-se a probabilidade do estado s_{t+1} ser s', sempre que o estado s_t tenha sido s e a ação tomada no tempo t seja a ação a Desta forma, a dependência que o seguinte estado s_{t+1} seja o estado s' está relacionada a tomar a ação a no tempo t .

De forma análoga, se, dados um estado e ação atuais e um estado seguinte s', o valor esperado do retorno é:

$$R_{ss'}^{a} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \tag{2.11}$$

Onde $E\{\ \}$ é o operador de valor esperado do retorno r_{t+1} , sempre que o estado s_t no tempo t passe a ser o estado s' no tempo $t+1$.

Os valores de probabilidade $P_{ss'}^{a}$ e retorno esperado $R_{ss'}^{a}$ determinam os aspectos mais importantes da dinâmica de um PDM finito. Assim, para quase todos os problemas de aprendizado por reforço é suposto que o ambiente tenha a forma de um Processo de Decisão de Markov, desde que seja cumprida a Propriedade de Markov (seção 2.3.1.1) no ambiente. Nem todos os algoritmos de RL necessitam uma modelagem PDM inteira do ambiente, mas é necessário ter-se pelo menos a visão do ambiente como um conjunto de estados e ações (Sutton e Barto, 1998) (Tupac, 2000).

2.3.2.
Métodos de Aprendizado por Reforço

Para solucionar o problema de aprendizado por reforço, existem três classes de métodos fundamentais, a saber: A Programação Dinâmica, métodos de Monte Carlo e métodos de Diferenças Temporais (Sutton e Barto, 1998).

Cada um destes métodos apresentam vantagens e desvantagens: Programação Dinâmica (Bellman, 1957) possui um bom desenvolvimento matemático, mas exige uma modelagem bem precisa do ambiente como um Processo de Decisão de Markov. Os métodos de Monte Carlo (Rubinstein, 1981) não precisam da modelagem do ambiente e se apresentam simples em termos conceituais. Entretanto, os métodos de Monte Carlo não são viáveis quando a solução do problema é possível apenas de forma incremental (utilizando somente o estado atual e os estados imediatos), porque para se atualizar, os métodos de Monte Carlo exigem que seja alcançado o estado final no processo. Os métodos de Diferenças Temporais não exigem um modelo exato do sistema e permitem ser incrementais, mas são complexos de analisar (Sutton e Barto, 1998) (Tupac, 2000). Nas subseções seguintes será analisado cada método separadamente.

2.3.2.1.
Programação Dinâmica

A programação dinâmica é um procedimento matemático de otimização que surgiu no início da década de 50. Richard Bellman foi responsável não somente pela formulação do principal aspecto teórico do método, mas também pela sua sistematização.

A programação dinâmica oferece algumas vantagens em relação a outras técnicas de programação matemática. A primeira dessas vantagens é poder tratar funções descontínuas, não diferenciáveis, não convexas, determinísticas ou estocásticas. Além disso, não exige o isolamento prévio de uma região convexa para a aplicação do procedimento. A última e não menos importante dessa série de vantagens é o fato de a programação dinâmica oferecer um algoritmo mais barato que a simples enumeração de todas as possibilidades de um problema combinatório, o que, em alguns casos, se torna impraticável ou extremamente caro.

O método possui também desvantagens entre as quais as principais são a necessidade de separabilidade e monotonicidade. Uma função é dita separável se, e somente se, a função de retorno ótimo depender apenas do estágio em que é avaliada, não sendo afetada pelos demais estados do sistema. Funções monotônicas, por sua vez, são aquelas em que um crescimento na função de retorno ótimo acarreta um crescimento na função objetivo. Normalmente, essas desvantagens não representam problemas para a aplicação da programação dinâmica na maioria dos problemas práticos (Marreco). A programação dinâmica oferece uma abordagem alternativa ao problema de controle ótimo (Kirk, 1970) que explora de forma mais direta a estrutura dinâmica do problema.

Programação Dinâmica é uma coleção de algoritmos que podem obter políticas ótimas sempre que exista uma modelagem perfeita do ambiente como um Processo de Decisão de Markov (Sutton e Barto, 1998), isto é, como um conjunto de estados, ações, retornos e as probabilidades da transição em todos os estados. Os algoritmos clássicos de programação dinâmica são usados de forma limitada no aprendizado por reforço, uma vez que, a modelagem perfeita do ambiente como MDP inclui um grande custo computacional. Porém, a Programação Dinâmica fornece um bom fundamento para o conhecimento dos outros métodos usados na solução do problema de aprendizado por reforço.

Partindo do pressuposto que o ambiente é completamente modelado como um Processo de Decisão de Markov, isto é, como um conjunto de estados *S*, ações *A*(*s*), onde $s \in S$. As dinâmicas do sistema são dadas por um conjunto de probabilidades de transição de estado, $P_{ss'}^{a} = Pr\{s_{t+1} = s' | s_t = s, a_t = a|\}$ e por um conjunto de reforços imediatos esperados, $R_{ss'}^{a} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$ para todo *s*, *s'*$\in S$, $a \in A(s)$.

A Programação Dinâmica organiza e estrutura a busca de boas políticas a partir das funções de valor. Deste modo, políticas ótimas são obtidas sempre que funções de valor ótimas são obtidas. Usualmente, as funções de valor ótimas são denotadas por V* (s) ou Q*(s, a) as quais satisfazem as equações de otimização de Bellman (Bellman, 1957), como é expresso nas eq (2.12) e eq (2.13) respectivamente:

$$V^*(s) = max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \quad (2.12)$$
$$= max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^*(s')]$$

$$Q^*(s,a) = E\{r_{t+1}, max_{a'} \gamma Q^*(s_{t+1}, a') | s_t = s, a_t = a\} \quad (2.13)$$
$$= \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma max_{a'} Q^*(s', a')]$$

Na eq. (2.12) a função de valor ótimo $V^*(s)$ é encontrada como o máximo das funções de valor esperadas segundo a ação selecionada, *a*.

2.3.2.1.1. Atualização da função valor e melhora na política

Para atualizar as funções de valor com a finalidade de melhorar a política, utiliza-se a iteração de valores (Sutton e Barto, 1998). A função de valor $V_{k+1}(s)$ do estado *s* para o $k + 1$ passo de avaliação é encontrada como na eq (2.14):

$$V_{k+1} = max_a E\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a\} \quad (2.14)$$

onde o valor atualizado $V_{k+1}(s)$ é encontrado a partir dos valores armazenados no passo *k* da seqüência de iterações, aplicando a equação de otimização de Bellman (eq. (2.12)). Esta seqüência de iterações deve alcançar no ponto final a política ótima $V^*(s)$ (Bertsekas, 1995) (Tupac, 2000).

O método de procura da política ótima da Programação Dinâmica exige a varredura de todos os estados no espaço de estados do modelo MDP, fazendo com que exista um grande custo computacional para modelagens complexas, resultando em uma desvantagem do método (Tupac, 2000) (Figueiredo, 2008).

2.3.2.2. Métodos de Monte Carlo MC

Os métodos de Monte Carlo são uma forma de resolver o problema de aprendizado por reforço baseando-se no cálculo da média de uma amostra de reforços. Para assegurar-se que exista um valor de retornos bem definido, os métodos de Monte Carlo são utilizados apenas para tarefas episódicas, isto é,

assumirmos que a experiência é dividida em episódios que de algum modo alcançam o estado final sem importar as ações que foram selecionadas. Desta forma, somente depois da conclusão de um episódio o valor de retorno é obtido e as políticas são atualizadas.

Uma vantagem dos métodos de Monte Carlo é que apenas necessitam das amostras da experiência como seqüências de dados, ações e reforços a partir de uma interação real ou simulada com o ambiente.

O aprendizado a partir de experiência real é notável, desde que não exige o conhecimento a priori das dinâmicas do ambiente, e, ainda, pode levar a um comportamento ótimo. Embora seja requerida uma modelagem, esta requer apenas gerar algumas transições de estados, sem precisar todo o conjunto de distribuições de probabilidade para todas as possíveis transições, como é exigido para um modelo que aplicam Programação Dinâmica (Sutton e Barto, 1998) (Tupac, 2000).

2.3.2.3. Método de Diferenças Temporais

O método de Diferenças Temporais (TD) resulta da combinação das idéias do método de Monte Carlo com as idéias da Programação Dinâmica. O aprendizado é feito diretamente a partir da experiência, sem a necessidade de uma modelagem PDM completa do ambiente, como característica do método de Monte Carlo, e atualizar suas estimativas com base em outras estimativas já aprendidas em estados sucessivos, nem necessitar alcançar o estado final de um episódio antes da atualização, como é característico em programação dinâmica.

Neste caso, a avaliação de uma política é abordada como um problema de predição, isto é, de estimar a função de valor V^{π} sob a política π.

2.3.2.3.1. Predição de Diferenças Temporais

Tanto diferenças temporais como o método de Monte Carlo utilizam a experiência para resolver o problema de predição. Dada certa experiência sob a política π se é visitado um estado intermediário s_t, ambos os métodos atualizam suas estimativas $V(s_t)$ baseando-se no acontecido depois de visitado o estado.

Sendo que o método de Monte Carlo espera até que o retorno total seja conhecido já que é utilizado para tarefas episódicas como se explicou na seção anterior.

Os métodos de Diferenças Temporais não necessitam alcançar o estado final de um episódio, mas o estado seguinte no tempo $t+1$. Em TD são utilizados o valor de reforço imediato r_{t+1} e a função de valor estimada $V(s_{t+1})$ ao invés do valor real de retorno R_t do Monte Carlo. Com estas condições, no método de Diferenças Temporais é como mostra a equação:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V_k(s_{t+1}) - V(s_t)] \tag{2.15}$$

Onde o objetivo para atualização é o valor $r_{t+1} + \gamma V_k(s_{t+1}) - V(s_t)$ que precisamente define a diferença no tempo t e $t+1$, característica esta que neste método toma o nome de Diferenças Temporais. Como a atualização é feita a partir do estado seguinte, os métodos TD são conhecidos como métodos *single-step*.

A vantagem mais notável do método TD é a relacionada com o método de Programação Dinâmica: TD não necessita uma modelagem PDM do ambiente, de seus retornos e das distribuições de probabilidade das transições dos seus estados, o que nos servirá de ajuda no capítulo 5.

A vantagem seguinte é em respeito ao método de Monte Carlo, visto que TD pode ser implementado de forma totalmente incremental para aplicações *On-Line*; os métodos de Monte Carlo devem aguardar até o final de um episódio para obter o retorno verdadeiro, enquanto TD só necessita aguardar até o estado seguinte. Se um problema gera episódios longos, para MC o aprendizado deve aguardar até o final de cada episódio, fazendo-o demorar. Outras aplicações são do tipo contínuas e o conceito de episódio não é aplicável com facilidade.

Embora as atualizações das funções de valor não sejam feitas a partir de retornos reais, mas de valores supostos, é garantida a convergência até a resposta correta (Jaakkola, Jordan e Singh, 1994).

2.3.2.3.2. Q-learning

Um dos mais importantes avanços na área de aprendizado pro reforço foi o desenvolvimento de um algoritmo baseado em Diferenças Temporais que dispensa a política, (*off-policy methods*) conhecido como *Q-Learning* (Watkins, 1989). A versão mais simples, *One-step Q-Learning*, é definida pela seguinte expressão:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (2.16)$$

Onde a função de valor do estado atual $Q(s_t,a_t)$ é atualizada a partir do seu valor atual, do reforço imediato r_{t+1}, e da diferença entre a máxima função de valor no estado seguinte (encontrando e selecionando a ação do estado seguinte que a maximize) e o valor de função do estado atual no tempo atual. O fato de selecionar a ação que maximize a função de valor seguinte permite achar de uma forma simples a função de valor estimada.

Uma característica do *Q-Learning* é que a função de valor *Q* aprendida aproxima-se diretamente da função de valor ótimo *Q** sem depender da política que está sendo utilizada. Este fato simplifica bastante a análise do algoritmo e permite fazer testes iniciais de convergência. A política ainda mantém algum efeito ao determinar qual dos pares estado-ação deve-se visitar e atualizar. A convergência exige que todos os pares estado-ação sejam visitados, fazendo com que *Q-Learning* seja um método *off-policy* (Sutton e Barto, 1998) (Tupac, 2000).

Q-learning foi o primeiro método RL a possuir fortes provas de convergência (Dayan, 1992) (Jaakkola, Jordan e Singh, 1994). É uma técnica muito simples que calcula diretamente as ações sem avaliações intermediárias e sem uso de modelo. O algoritmo *Q-Learning* é mostrado de forma seqüencial na figura seguinte:

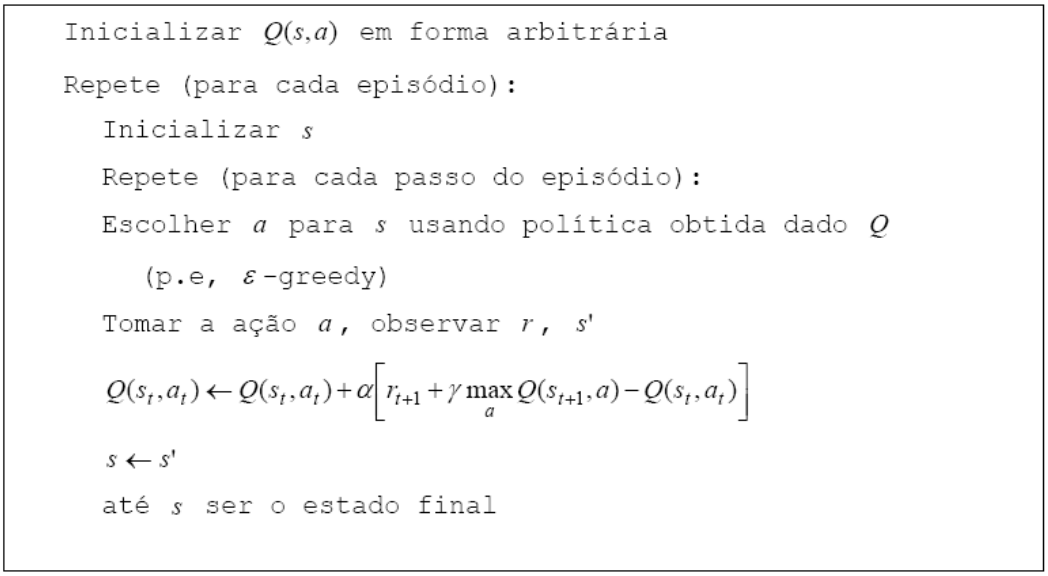

Figura 7 – Algoritmo Q-Learning.

Dados os valores Q, existe uma política definida pela execução da ação a, quando o agente está em um estado s, que maximiza o valor Q(s,a). (Watkins, 1989) mostrou que se cada par estado-ação for experimentado um número suficientemente grande de vezes e α decrescer apropriadamente, as funções de valores-Q irão convergir com probabilidade um para Q* e a política irá convergir para uma política ótima.

A convergência do algoritmo *Q-learning* não depende do método de exploração usado. Um agente pode explorar suas ações a qualquer momento – não existem requisitos para a execução de ações estimadas como as melhores. No entanto, para melhorar o desempenho do sistema é necessária, durante o aprendizado, a busca das ações que maximizam o retorno.

Existem duas classes principais de algoritmos TD: *on-policy* e *off-policy*. Métodos *on-policy* avaliam ou melhoram a política que está sendo usada para tomar decisões, enquanto métodos *off-policy* avaliam ou melhoram uma política diferente daquela que está sendo seguida. No aprendizado *off-policy*, a política usada para selecionar as ações é chamada política de comportamento (*behavior policy*) e a política que está sendo melhorada é chamada de política de estimação (*estimation policy*). Em aprendizados *on-policy* existe uma única política que é usada para escolher as ações durante o período em que está sendo aprimorada. Os dois tipos de política usam métodos estocásticos para escolher ações, como por exemplo a política *ε-greedy*.

A política *ε-greedy* é definida no algoritmo pela escolha da ação que possui o maior valor esperado, com probabilidade definida por (1- ε), e de ação aleatória, com probabilidade ε. É um meio efetivo e popular de balancear a exploração e explotação em aprendizado por reforço. Este processo permite que o algoritmo explore o espaço de estados e esta é uma das condições necessárias para garantir que algoritmos RL encontrem a ação ótima (Figueiredo, 2008).

Resumidamente, pode-se enumerar os mais importantes aspectos do algoritmo *Q-Learning*:

- O objetivo do uso do algoritmo *Q-Learning* é achar uma regra de controle que maximize cada ciclo de controle.
- O uso do reforço imediato é indicado sempre que possível e necessário, desde que ele contenha informação suficiente que ajude o algoritmo a achar a melhor solução.
- *Q-Learning* é adotado quando o número de estados e ações a serem selecionadas é finito e pequeno.

O algoritmo *Q-learning* utiliza uma tabela para mapear estados/ações. Quando o espaço de estados é muito grande e/ou contínuo, o método torna-se inviável. Este problema é denominado na literatura como *curse of dimensionality* (Bellman, 1957) (Figueiredo, 2008).