Inteligência Artificial

Fabrício Olivetti de França

07 de Junho de 2018

Aprendizado por Reforço

Aprendizado por Reforço

Até esse instante assumimos que o agente tem um modelo completo do ambiente, sabe exatamente o estado atual, o conjunto de ações e as recompensas para cada estado, em muitos casos não temos conhecimento de nenhum desses.

Imagine jogar um jogo sem conhecer as regras e, após algumas centenas de jogadas, o seu oponente anuncia: *Você perdeu!*.

Vamos assumir um ambiente:

- **Totalmente observável:** o agente consegue conhecer o estado atual do ambiente.
- O agente não sabe em que cada ação resulta.
- O efeito das ações são probabilísticas.

Isso torna impossível aplicar os algoritmos aprendidos na aula anterior para encontrar a política ótima.

Podemos tentar:

- Aprender uma **função utilidade** de cada estado e usar tal função para selecionar as ações que maximizam o valor total esperado.
- Aprender uma **função-Q** $Q(s, a)$ que retorna um valor esperado de executar uma ação no estado atual.
- Aprender uma **função reflexiva** que retorna uma ação dado um estado.

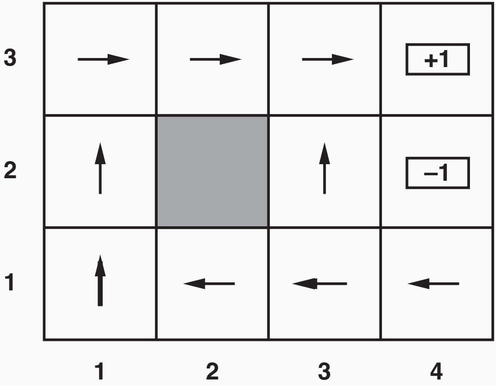
Aprendizado Passivo

Aprendizado Passivo

No **aprendizado passivo** desejamos estimar a função utilidade através de observações utilizando uma **política fixa**.

O objetivo é apenas determinar quão boa é uma política, estimando U^{π}.

Considere a seguinte política:



Aprendizado Passivo

O aprendizado passivo executa diversas **tentativas** de seguir a política até algum estado final.

Aprendizado Passivo

Considere essas três tentativas:

$$
\begin{aligned}
(1,1)_{-0.04} \mapsto (1,2)_{-0.04} \mapsto (1,3)_{-0.04} &\mapsto (1,2)_{-0.04} \mapsto (1,3)_{-0.04} \\
&\mapsto (2,3)_{-0.04} \mapsto (3,3)_{-0.04} \mapsto (4,3)_{+1}
\end{aligned}
$$

$$
\begin{aligned}
(1,1)_{-0.04} \mapsto (1,2)_{-0.04} \mapsto (1,3)_{-0.04} &\mapsto (2,3)_{-0.04} \mapsto (3,3)_{-0.04} \\
&\mapsto (3,2)_{-0.04} \mapsto (3,3)_{-0.04} \mapsto (4,3)_{+1}
\end{aligned}
$$

$$
(1,1)_{-0.04} \mapsto (2,1)_{-0.04} \mapsto (3,1)_{-0.04} \mapsto (3,2)_{-0.04} \mapsto (4,2)_{-1}
$$

Aprendizado Passivo

Com isso temos uma amostra da utilidade de cada estado segundo a política π.

Aprendizado Passivo

Observando a primeira tentativa:

$$\begin{aligned}(1,1)_{-0.04} \mapsto (1,2)_{-0.04} \mapsto (1,3)_{-0.04} \mapsto (1,2)_{-0.04} \mapsto (1,3)_{-0.04} \\ \mapsto (2,3)_{-0.04} \mapsto (3,3)_{-0.04} \mapsto (4,3)_{+1}\end{aligned}$$

Aprendizado Passivo

Temos:

s	U
(1,1)	0.72
(1,2)	0.76
(1,3)	0.80
(1,2)	0.84
(1,3)	0.88
(2,3)	0.92
(3,3)	0.96
(4,3)	1.00

Aprendizado Passivo

Qual seria uma boa estimativa para $U^\pi(1,2)$?

s	U
(1,1)	0.72
(1,2)	0.76
(1,3)	0.80
(1,2)	0.84
(1,3)	0.88
(2,3)	0.92
(3,3)	0.96
(4,3)	1.00

$$U^{\pi}(1,2) = (0.76 + 0.84)/2 = 0.8$$

s	U
(1,1)	0.72
(1,2)	0.76
(1,3)	0.80
(1,2)	0.84
(1,3)	0.88
(2,3)	0.92
(3,3)	0.96
(4,3)	1.00

Aprendizado Passivo

Com a segunda tentativa:

$$
\begin{aligned}
(1,1)_{-0.04} \mapsto (1,2)_{-0.04} &\mapsto (1,3)_{-0.04} \mapsto (2,3)_{-0.04} \mapsto (3,3)_{-0.04} \\
&\mapsto (3,2)_{-0.04} \mapsto (3,3)_{-0.04} \mapsto (4,3)_{+1}
\end{aligned}
$$

Aprendizado Passivo

Temos:

s	U
(1,1)	0.72
(1,2)	0.76
(1,3)	0.80
(2,3)	0.84
(3,3)	0.88
(3,2)	0.92
(3,3)	0.96
(4,3)	1.00

Aprendizado Passivo

Como atualizamos $U^\pi(1,2) = (0.76 + 0.84)/2 = 0.8$?

s	U
(1,1)	0.72
(1,2)	0.76
(1,3)	0.80
(2,3)	0.84
(3,3)	0.88
(3,2)	0.92
(3,3)	0.96
(4,3)	1.00

Aprendizado Passivo

Como atualizamos $U^\pi(1,2) = (0.76 + 0.84 + 0.76)/3 = 0.79$?

s	U
(1,1)	0.72
(1,2)	0.76
(1,3)	0.80
(2,3)	0.84
(3,3)	0.88
(3,2)	0.92
(3,3)	0.96
(4,3)	1.00

Aprendizado Passivo

E na terceira tentativa:

$$(1,1)_{-0.04} \mapsto (2,1)_{-0.04} \mapsto (3,1)_{-0.04} \mapsto (3,2)_{-0.04} \mapsto (4,2)_{-1}$$

Aprendizado Passivo

Temos:

s	U
(1,1)	-1.16
(2,1)	-1.12
(3,1)	-1.08
(3,2)	-1.04
(4,2)	-1.00

Aprendizado Passivo

O objetivo é obter:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

Estimativa Direta da Utilidade

Uma forma simples para estimar a função de utilidade $U^\pi(s)$ para um dado s é calcular a média da utilidade obtida em todas as ocorrências desse s.

Estimativa Direta da Utilidade

Alternativamente, podemos utilizar a média móvel das n últimas ocorrências de s, dessa forma se nosso ambiente é alterado com o tempo, teremos sempre uma estimativa atual da utilidade.

Estimativa Direta da Utilidade

Em um ambiente estacionário, com o número de amostras tendendo ao infinito, obtemos o valor real da utilidade para cada estado percorrido pela política π.

Estimativa Direta da Utilidade

Note que essa forma de estimar a utilidade remete a um **aprendizado supervisionado** ou **indutivo**, em que temos uma amostra de valores de utilidade para cada estado e desejamos estimar um $\hat{U}(s) \approx U^{\pi}(s)$.

Estimativa Direta da Utilidade

Na disciplina de **Aprendizado de Máquina** são apresentados diversos algoritmos desse tipo.

Aprendizado por Diferença Temporal

Uma outra forma de atualizar os valores de U^π é levar em conta as restrições da equação de Bellman.

Aprendizado por Diferença Temporal

Partindo do algoritmo anterior, digamos que calculamos a estimativa de $U^{\pi}(s)$ para os estados $(1,3)$ e $(2,3)$ após a primeira amostra:

$$\begin{aligned}(1,1)_{-0.04} \mapsto (1,2)_{-0.04} &\mapsto (1,3)_{-0.04} \mapsto (1,2)_{-0.04} \mapsto (1,3)_{-0.04} \\ &\mapsto (2,3)_{-0.04} \mapsto (3,3)_{-0.04} \mapsto (4,3)_{+1}\end{aligned}$$

$$U^{\pi}(1,3) = 0.84$$

$$U^{\pi}(2,3) = 0.92$$

Na segunda amostra temos:

$$
\begin{aligned}
(1,1)_{-0.04} \mapsto (1,2)_{-0.04} &\mapsto (1,3)_{-0.04} \mapsto (2,3)_{-0.04} \mapsto (3,3)_{-0.04} \\
&\mapsto (3,2)_{-0.04} \mapsto (3,3)_{-0.04} \mapsto (4,3)_{+1}
\end{aligned}
$$

Pela equação de Bellman, temos que a utilidade do estado $(1, 3)$ deveria obedecer ($\gamma = 1$):

$$U^\pi(1, 3) = -0.04 + U^\pi(2, 3)$$

E no nosso caso:

$$U^{\pi}(1,3) = -0.04 + 0.92 = 0.88$$

Sendo que a estimativa é 0.84.

Podemos então atualizar U^π como:

$$U^\pi(s) = U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s))$$

com α sendo a taxa de aprendizado e $0 \leq \alpha \leq 1$.

Aprendizado por Diferença Temporal

Em nosso exemplo, seria:

$$U^\pi(1,3) = U^\pi(1,3) + \alpha(-0.04 + 0.92 - U^\pi(1,3))$$

Que para um $\alpha = 1$:

$$U^{\pi}(1,3) = 0.84 + (-0.04 + 0.92 - 0.84) = 0.88$$

Mas para $\alpha = 0.1$:

$U^\pi(1,3) = 0.84 + 0.1(-0.04 + 0.92 - 0.84) = 0.844$

Ele atualiza um pouquinho na direção apontada pela equação de Bellman.

Note que a atualização do estado s envolve apenas o estado seguinte s' que foi amostrado.

Lembrando que dada uma ação a podemos chegar em diferentes estados s' probabilisticamente, um estado s' raro (e que afeta negativamente o sistema) poderia subestimar ou superestimar o valor da utilidade.

Aprendizado por Diferença Temporal

Mas, como trabalhamos com amostragens e essa equação leva ao cálculo de um valor médio, um evento raro afeterá pouco no resultado.

Se temos um α inversamente proporcional a quantas vezes um estado s foi visitado (quanto maior o número de visitas, menor o α), garantimos a convergência para o valor correto de $U^\pi(s)$.

Um exemplo é fazer:

$\alpha(n) = \frac{60}{59+n}$

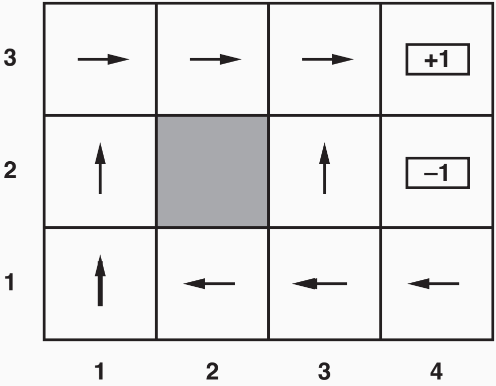
Aprendizado por Diferença Temporal

Se temos um espaço de busca com muitos estados, o algoritmo pode se tornar intratável.

Uma alternativa é utilizar uma heurística para atualizar apenas determinados estados.

Exercício

Dada a política:



Exercício

E as amostras:

$$
\begin{aligned}
(1,1)_{-0.04} \mapsto (1,2)_{-0.04} \mapsto (1,3)_{-0.04} &\mapsto (1,2)_{-0.04} \mapsto (1,3)_{-0.04} \\
&\mapsto (2,3)_{-0.04} \mapsto (3,3)_{-0.04} \mapsto (4,3)_{+1}
\end{aligned}
$$

$$
\begin{aligned}
(1,1)_{-0.04} \mapsto (1,2)_{-0.04} \mapsto (1,3)_{-0.04} &\mapsto (2,3)_{-0.04} \mapsto (3,3)_{-0.04} \\
&\mapsto (3,2)_{-0.04} \mapsto (3,3)_{-0.04} \mapsto (4,3)_{+1}
\end{aligned}
$$

Exercício

Estime os valores da utilidade utilizando a média simples com a primeira amostra e diferença temporal com a amostra seguinte e $\alpha = 0.1, \gamma = 1$.

Exercício

$$U(1,1) = 0.72$$
$$U(1,2) = 0.76$$
$$U(1,3) = 0.80$$
$$U(1,2) = 0.84$$
$$U(1,3) = 0.88$$
$$U(2,3) = 0.92$$
$$U(3,3) = 0.96$$
$$U(4,3) = 1.00$$

Exercício

$$U(1,1) = 0.72$$
$$U(1,2) = (0.76 + 0.84)/2 = 0.80$$
$$U(1,3) = (0.80 + 0.88)/2 = 0.84$$
$$U(2,3) = 0.92$$
$$U(3,3) = 0.96$$
$$U(4,3) = 1.00$$

Exercício

$$U(s) = U(s) + 0.1(-0.04 + U(s') - U(s))U(1,1) \quad = 0.72 + 0.1(-0.0$$

$$U(1,2) = 0.80 + 0.1(-0.04 + 0.84 - 0.80) = 0.80$$

$$U(1,3) = 0.84 + 0.1(-0.04 + 0.92 - 0.84) = 0.844$$

$$U(2,3) = 0.92 + 0.1(-0.04 + 0.96 - 0.92) = 0.92$$

$$U(3,3) = 0.96 + 0.1(-0.04 + 0 - 0.96) = 0.86$$

$$U(3,2) = 0 + 0.1(-0.04 + 0.86 - 0) = 0.082$$

$$U(3,3) = 0.86 + 0.1(-0.04 + 1 - 0.86) = 0.87$$

$$U(4,3) = 1.00$$

Aprendizado por Reforço Ativo

Estimando a utilidade e a política

Nos algoritmos anteriores focamos em estimar a função de utilidade para uma política fixa, assumindo que ela era ótima.

Estimando a utilidade e a política

No Aprendizado Ativo, queremos também estimar a política ótima.

Estimando a utilidade e a política

Basicamente temos:

```python
def ativo(U, pi):
    U  = estima(U, pi)
    pi = politica(U)
    return U, pi
```

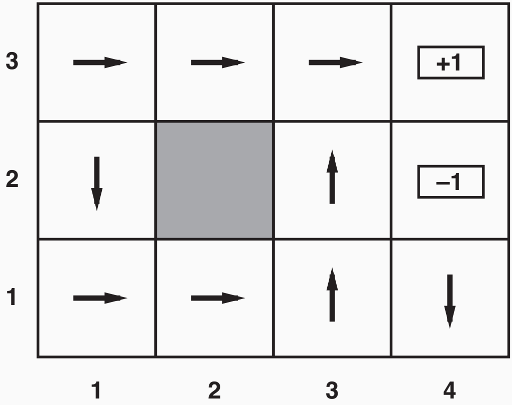
Estimando a utilidade e a política

Ou seja, dados U, π, fazemos uma nova simulação para aprimorar a estimativa de U (através dos algoritmos anteriores) e, então, estimamos um novo π, baseado no U atual (iteração de política).

Estimando a utilidade e a política

O uso desse algoritmo leva a criação de um **agente guloso** pois ele sempre irá caminhar pelo melhor caminho encontrado até o momento.

Considere que em certo momento o agente encontrou a seguinte política que o levou até a posição $+1$:



Estimando a utilidade e a política

Como essa foi a melhor solução encontrada até então, o agente passará a **reforçar** esse caminho com poucas alterações causadas pela probabilidade de execução de ação.

Dificilmente ele irá encontrar a nossa solução ótima uma vez que ele entra nessa auto-alimentação da solução atual.

Estimando a utilidade e a política

Portanto, o agente guloso tem um problema em que ele não explora novos caminhos caso tenha encontrado algo *bom*.

Embora a estimativa de U seja ótima segundo a política π atual, não necessariamente a política é ótima.

Exploração vs Explotação

Uma forma de evitar isso é balancear a **exploração** e a **explotação**.

Exploração vs Explotação

- **Exploração:** percorrer caminhos pouco (ou ainda não) percorridos.
- **Explotação:** reforçar o melhor caminho encontrado até então.

Exploração vs Explotação

Um bom equilíbrio entre exploração e explotação pode determinar o sucesso de uma heurística.

Exploração vs Explotação

Uma ideia simplista é favorecer caminhos não explorados até que estes tenham sido explorados por um n número de vezes.

Exploração vs Explotação

Alguns algoritmos alternam entre exploração e explotação probabilisticamente.

Aprendizado Ativo com Diferença Temporal

Q-Learning é um algoritmo de diferença temporal que aprende uma função $Q(s, a)$ que indica a qualidade em escolher a ação a no estado s.

Q-Learning

Essa função é relacionada com a função utilidade de tal forma que:

$$U(s) = \max_a Q(s,a)$$

Similar a diferença temporal, no algoritmo Q-Learning atualizamos a função $Q(s, a)$ como:

$$Q(s, a) = Q(s, a) + \alpha(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

que é executado toda vez que aplicamos uma ação a em s levando ao estado s'.

Q-Learning

Como não precisamos estimar $P(s' \mid s, a)$ e dizemos que Q-Learning é um algoritmo **livre de modelo** (*model free*).

Algoritmo Q-Learning

```python
def qlearn(percept, state, goals, A, gamma, alpha, f):
    sn, rn         = percept
    s, a, r, N, Q = state

    # se for estado final, usa a recompensa
    if sn in goals:
        Q[(sn, None)] = rn
```

Algoritmo Q-Learning

```
# atualiza Q
else:
    maxQ     = max(Q[(sn,ai)] for ai in A[sn])
    Q[(s,a)] = Q[(s,a)] + alpha * (r + gamma*maxQ - Q[(s,a)])
```

Algoritmo Q-Learning

```
# avalia o proximo estado e a acao que deve ser tomada
qvals    = [(Q[(sn,ai)], ai) for ai in A[sn]]
an       = argmax(qvals)
state    = (sn, an, rn, N, Q)

return an, state
```

Algoritmo Q-Learning

Sendo o estado s atual um estado final, atualiza o valor de $Q(s,)$ para a recompensa final. Note que não temos uma ação a ser executada.

Algoritmo Q-Learning

Caso não seja um estado final, atualiza o valor de $Q(s, a)$ para o novo valor utilizando a diferença temporal.

Algoritmo Q-Learning

Em seguida, escolhe a próxima ação que maximize $Q(s', a)$

SARSA: State-Action-Reward-State-Action

Um problema do Q-Learning é que ele não leva em conta as consequências da ação atual, ou seja, ele assume que a ação escolhida para executar terá consequências futuras ótimas.

Uma forma de aliviar tal problema é modificando a equação de atualização de $Q(s, a)$ para:

$$Q(s, a) = Q(s, a) + \alpha(R(s) + \gamma Q(s', a') - Q(s, a)),$$

ou seja, além do estado e ação atual s, a, também já deve ser conhecidos o estado e ação futura s', a'. Basta alterar a linha pertinente no algoritmo anterior.

Exploração vs Explotação

- **Exploração:** aprender possíveis valores de $Q(s, a)$ para combinações de estado e ação ainda não observados.
- **Explotação:** seguir um caminho mais provável de maximar a recompensa, dado o conhecimento atual.

Exploração vs Explotação

Com certa probabilidade escolhe entre explorar e explotar. Se escolher explotar, o agente segue a ação que maximiza o valor de $Q(s', a)$.

Exploração vs Explotação

Caso seja escolhido explorar, ou escolhe-se uma ação completamente aleatória, ou uma dentre aquelas que foram pouco exploradas.

Além disso, podemos fazer um passo de aprendizado adaptativo com $\alpha = \alpha_0 / t$ sendo t o número de episódios jogados até então.

Espaço de busca

De todo modo, o algoritmo Q-Learning ainda sofre em ambientes contendo muitos possíveis estados. Esse algoritmo costuma ser mais lento do que aprendizado passivo, porém permite adaptar facilmente para os casos em que não temos um ambiente como o requisitado pelo MDP.

Espaço de busca

Uma alternativa para tratar o problema de espaço de busca muito grande, é tentar encontrar uma função fechada para $Q(s, a)$ que não necessite de uma tabela com as informações dos valores Q.

Generalizando para Estados não vistos

Estimando uma função

Uma outra forma de pensarmos na função utilidade é de ela depender de uma variável θ de dimensão menor que a representação dos estados.

Dessa forma desejamos obter $U'(\theta) \approx U(s)$ tal que θ codifica s.

Estimando uma função

Seguindo nosso exemplo, poderíamos pensar em $\theta = [x, y]$, ou as coordenadas do mundo de nosso agente.

Estimando uma função

Dada uma amostra de tentativas, codificamos cada estado s em uma variável θ e, então, utilizamos um algoritmo de aprendizado supervisionado para gerar um modelo de função U'.

Estimando uma função

```
def utilityModel(nTrials, S, R, s, goals, gamma, nextState):
    dataX, dataY = makeNTrials(nTrials)

    lr = LinearRegression()
    lr.fit(dataX, dataY)

    model = lambda s: lr.predict([[1, s[0], s[1]]])[0]

    return { s : model(s) for s in S }
```

Estimando uma função

Para o nosso problema com 1000 execuções, obtemos a seguinte equação:

$$0.11x + 0.17y - 0.06$$

Estimando uma função

Que se traduz em:

0.56	0.68	0.79	0.90
0.40		0.62	0.74
0.23	0.34	0.46	0.57

Estimando uma função

E na política:

->	->	->	0.90
^		^	0.74
^	->	^	^

Estimando uma função

Se acrescentarmos uma nova variável como $(x - x_g)^2 + (y - y_g)^2$, obtemos:

0.54	0.51	0.50	0.48
0.38		0.34	0.32
0.23	0.20	0.18	0.17

Estimando uma função

E na política:

->	<-	->	0.48
^		^	0.32
^	<-	^	^

Estimando uma função

Essa forma de encontrar uma política ótima tem a vantagem de codificar o espaço de estados, que pode ser bem numeroso, para um conjunto de estados gerenciáveis.

Estimando uma função

Além disso, por estimar uma função, os modelos de aprendizado conseguem generalizar para estados que ainda não foram observados através de interpolação.

Busca por uma Política

Busca por uma Política

Uma última alternativa para estimar a política ótima é fazer a busca no **espaço de políticas**.

Busca por uma Política

A ideia geral é seguir um algoritmo de busca em que, dada uma política inicial, altera essa política até que nenhuma outra melhora possa ser feita.

Um possível algoritmo é o **Hill Climbing** que parte de uma solução inicial $s0$, testa todas as soluções vizinhas e substitui a solução atual pelo melhor vizinho, caso tenha um melhor desempenho.

A busca para no momento que não existir um vizinho melhor que a solução inicial.

```python
def hillClimbing(mdp, s0, goals, nTrials):
    S, A, R, P, gamma = mdp

    piCurrent = { s : np.random.choice(A[s]) for s in S }
    Ucurrent  = evaluate(mdp, piCurrent, nTrials)
    improved  = True
```

```
while improved:
    improved          = False
    bestNeigh, Uneigh = genNeighbors(mdp, piCurrent, s0,
                                          goals, nTrials)

    if Uneigh[s0] > Ucurrent[s0]:
        improved  = True
        Ucurrent  = deepcopy(Uneigh)
        piCurrent = deepcopy(bestNeigh)
return piCurrent
```

Conclusão

Os algoritmos de Aprendizado por Reforço devem ser aplicados quando não temos todas as informações do problema de busca e queremos uma estimativa da função de utilidade dos estados ou da política ótima.

Conclusão

Esses algoritmos podem ser de:

- **Aprendizado Passivo:** em que fixamos a informação de política para estimar a utilidade.
- **Aprendizado Ativo:** alternamos entre estimar a política e a função utilidade.
- **Aprendizado de Modelo:** quando utilizamos um modelo de regressão ou classificação em uma amostra de jogadas.
- **Aprendizado por Busca:** em que buscamos por uma política ótima utilizando uma estimativa da utilidade como função-objetivo.