Aprendizado por Reforço

Fabrício Olivetti de França

Universidade Federal do ABC

Tópicos

1. Aprendizado por Reforço
2. Q-Learning
3. SARSA
4. Outras ideias

UFABC

Aprendizado por Reforço

Problemas de decisão sequencial

Imagine o seguinte ambiente:

3				$+1$
2		$\#$		-1
1	S			
	1	2	3	4

com S representando onde o agente inicia, $\#$ um obstáculo e $+1, -1$ os estado finais.

UFABC

Problemas de decisão sequencial

Dado um conjunto de ações $A = \{\text{direita}, \text{esquerda}, \text{cima}, \text{baixo}\}$, o objetivo é determinar a sequência de ações que maximiza a recompensa.

Assumindo um ambiente **totalmente observável** e **determinístico**, uma solução seria: $\{C, C, R, R, R\}$.

Problemas de decisão sequencial

Porém, em muitas situações o modelo é **estocástico** e o agente pode não obedecer exatamente o comando.

Imagine que o agente do exemplo faz a ação com probabilidade 0.8 e, com probabilidades de 0.1 ele move nas direções que formam 90° em relação a direção desejada.

Problemas de decisão sequencial

Nessa situação, a solução proposta anteriormente atinge a recompensa $+1$ com probabilidade $0.8^5 = 0.32768$.

Note que essa solução tem também a chance de atingir essa recompensa pelo outro lado com probabilidade $0.1^4 \cdot 0.8 = 0.32776$.

Problemas de decisão sequencial

Dessa forma temos $P(s' \mid s, a)$ como sendo a probabilidade de atingir o estado s' dado que o agente está no estado s e tentou fazer a ação a.

Para sermos capaz de avaliar soluções para esse tipo de problema devemos definir uma função utilitária de recompensa $R(s)$, que pode ser positiva ou negativa, e indica a qualidade de estar no estado s.

Problemas de decisão sequencial

Para nosso exemplo, vamos assumir $R(s) = -0.04$ para qualquer estado, exceto os finais em que o valor é igual a recompensa final.

A recompensa de uma sequencia de ações é a soma de todas as recompensas obtidas. Ou seja, se o agente leva 10 ações antes de chegar até o estado final $+1$, a recompensa total será $-0.04 \cdot 10 + 1 = 0.6$.

Problemas de decisão sequencial

A representação de uma solução para esse problema é através de uma função denominada **política**:

$\pi : s \in S \rightarrow a \in A$,

tal que $\pi(s)$ retorna a ação que deve ser executada pelo agente no estado s.

Problemas de decisão sequencial

Uma política ótima π^* é aquela que maximiza a maior recompensada total esperada.

UFABC

Recompensa

A recompensa pode ser **aditiva**:

$$U([s_0, s_1, s_2, ...]) = R(s_0) + R(s_1) + R(s_2) + ...$$

ou descontada por um fator γ:

$$U([s_0, s_1, s_2, ...]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...,$$

com o fator de desconto $0 \leq \gamma \leq 1$. Um desconto de 0 só leva em conta o estado final, já um desconto igual a 1 é a recompensa aditiva.

Política Ótima

Dada a recompensa total esperada para uma política π:

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

a política ótima pode ser encontrada como:

$\pi_s^* =_s U^{\pi}(s)$,

ou

$\pi^* =_{a \in A} \sum_{s'} P(s' \mid s, a) U(s')$.

UFABC

Equação de Bellman

Partindo da política ótima chegamos na equação de Bellman para o valor de um estado:

$$U(s) = R(s) + \gamma max_a \sum_{s'} P(s' \mid s, a) U(s')$$

UFABC

Equação de Bellman

No nosso exemplo, partindo do estado $(1,1)$ temos:

$$
\begin{aligned}
(1,1) = -0.04 + \gamma max[\\
0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), \\
0.9U(1,1) + 0.1U(1,2), \\
0.9U(1,1) + 0.1U(2,1), \\
0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)]
\end{aligned}
$$

com a sequência *cima, esquerda, baixo, direita*, o que nos indica que *cima* é a melhor ação.

Algoritmo Value-Iteration

Algoritmo que atualiza os valores de U utilizando a própria equação de Bellman até convergência.

---

procedure VALUEITERATION

$U, U' = 0$

$\delta = 0$

while $\delta < (1 - \gamma)\gamma$ **do**

for $s \in S$ **do**

$U'(s) = R(s) + \gamma max_a \sum_{s'} P(s' \mid s, a) U(s')$.

$\delta = max(\delta, abs(U'(s) - U(s)))$

return U

Algoritmo Policy-Iteration

Nesse algoritmo fazemos o procedimento inverso, iniciamos com uma política π, avaliamos e atualizamos.

---

procedure POLICYITERATION

$changed = True$

while *changed* **do**

$U = eval(\pi)$

$changed = False$

for $s \in S$ **do**

if $\max_a \sum_{s'} P(s' \mid s, a)U(s') > \sum_{s'} P(s' \mid s, \pi(s))U(s')$

then

$\pi(s) =_a \sum_{s'} P(s' \mid s, a)U(s')$

$changed = True$

return π

UFABC

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!*.

Aprendizado por Reforço

Vamos assumir:

- 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.

Aprendizado por Reforço

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 $Q(s, a)$ que retorna um valor esperado de executar uma ação no estado atual.
- Aprender uma função que retorna uma ação dado um estado.

Na aula de hoje aprenderemos o segundo.

Q-Learning

Q-Learning

- Aprender $Q(s, a)$ que retorna um valor esperado ao executar uma certa ação a no estado s.
- Tal função é relacionada a função utilidade de tal forma que $U(s) = \max_a Q(s, a)$.
- Não precisamos estimar $P(s' \mid s, a)$ e, portanto, Q-Learning é um algoritmo *model free*.

Q-Learning

Da mesma forma que o algoritmo *Value-Iteration*, atualizamos os valores de $Q(s, a)$ por:

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

com α sendo a taxa de aprendizado.

UFABC

Algoritmo Q-Learning

procedure QLearning

  while $s! = final$ **do**

    $a =_a Q(s, a)$

    $r = reward(s, a)$

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

  return Q, a

UFABC

SARSA: State-Action-Reward-State-Action

Um problema do Q-Learning é que ele não leva em conta as consequências do ato 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.

UFABC

Outras ideias

Q-Learning x Mario

O algoritmo de Q-Learning apresenta alguns problemas quando executado no jogo Super Mario World:

- A recompensa inerente do jogo ocorre em ocasiões muito especificas: quando mata um inimigo, quando o Mario morre, quando coleta moedas, etc. Não existe uma função de recompensa bem formada.
- Existem muitas combinações possíveis de estado e ações, o algoritmo Q-Learning converge rapidamente para um ótimo local, muitas vezes ruim.
- Uma ação feita em um estado s pode refletir negativamente apenas alguns estados depois. O algoritmo leva em conta apenas o atual e o imediatamente seguinte.

Recompensa

A recompensa foi definida como:

$$R = 0.3 \log r + 0.5\text{direita} - 2.0\text{gameOver} + 0.1(1 - \text{gameOver},$$

com r sendo qualquer pontuação que o jogador ganhou ao executar a ação, *direita* indica se o jogador andou para a direita sem perder o jogo, e *gameOver* é 1 se ele perdeu o jogo e 0 caso contrário.

Essa recompensa estimula que o jogador siga em frente e não morra.

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 $Q(s, a)$.

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

Além disso, $\alpha = \alpha_0/t$ com t sendo o número de episódios jogados até então.

Efeito colateral

Para aliviar o problema de ações com efeito negativos apenas após algumas transições de estado, os valores de $Q(s, a)$ são atualizados ao final de um episódio com:

$$Q(s, a) = Q(s, a) + \alpha(0.1 \log(\Delta x) - Q(s, a),$$

para todo (s, a) desse episódio e com Δx sendo o total da distância percorrida pelo Mario.

Próxima aula

Aprenderemos como usar o conhecimento de Aprendizado Supervisionado para estimar a função $U(s)$ de tal forma que possamos encontrar a política ótima com esse valor aproximado.

UFABC