Uczenie się ze wzmocnieniem

W wielu dziedzinach trudno jest sformułować precyzyjne funkcje oceny, pozwalające agentowi ocenić skuteczność, lub poprawność jego akcji, z wyjątkiem gdy osiągnie on stan docelowy. Zakładamy, że w stanie docelowym agent z założenia zawsze otrzymuje obiektywną ocenę swoich działań, zwaną **nagrodą** lub **wzmocnieniem**. Wygodnie jest sformułować zadanie agenta w taki sposób, aby musiał on sam nauczyć się tak działać, aby maksymalizować tę nagrodę (wzmocnienie). Nazywamy to zagadnieniem **uczenia się ze wzmocnieniem** (*reinforcement learning, RL*).

Jest to zadanie trudne. W ogólnym przypadku agent może nie mieć pełnej informacji o swoim środowisku, jak również precyzyjnego (albo żadnego) opisu swoich działań i ich skutków. Jego sytuację można rozumieć jako jedno ze sformułowań pełnego zadania sztucznej inteligencji. Agent zostaje umieszczony w środowisku, którego nie zna, i musi się nauczyć skutecznie w nim działać, aby maksymalizować pewne kryterium, dostępne mu w postaci wzmocnień.

Będziemy rozważali probabilistyczny model skutków akcji agenta. Mówiąc dokładniej, będziemy zakładali, że podstawowe zagadnienie jest dyskretnym procesem Markowa (MDP), jednak agent nie zna jego parametrów.

Pasywne uczenie się ze wzmocnieniem

Na początek rozważymy **uczenie się pasywne**, gdzie zakładamy, że polityka agenta $\pi(s)$ jest z góry ustalona. Agent nie podejmuje żadnych decyzji, musi robić to co dyktuje mu polityka, a wyniki jego akcji są probabilistyczne. Jednak może obserwować co się dzieje, czyli wie do jakich stanów dociera i jakie otrzymuje w nich nagrody. Pamiętajmy jednak, że nagrody otrzymywane w stanach nieterminalnych nie są dla agenta istotnym kryterium — liczy się tylko suma nagród otrzymanych na drodze do stanu terminalnego, zwana wzmocnieniem.

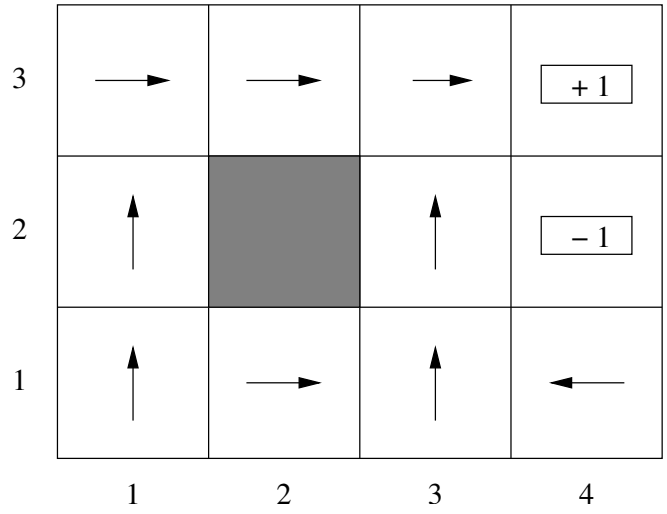
Zadaniem agenta jest nauczenie się wartości użyteczności stanów $U^\pi(s)$, obliczanych zgodnie z równaniem:

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

W rozpatrywanym tu przykładowym zagadnieniu 4x3 będziemy przyjmować $\gamma = 1$.

Przebiegi uczące

Przypomnijmy rozważany wcześniej przykład agenta w świecie 4×3:



3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Agent wykonuje **przebiegi uczące** (ang. *trials*) w których wykonuje akcje zgodne z posiadaną polityką, aż do osiągnięcia stanu terminalnego. W każdym kroku otrzymuje percept wskazujący zarówno bieżący stan, jak i nagrodę. Przykładowe przebiegi:

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

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

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

Obliczanie użyteczności metodą bezpośrednią

Celem działania agenta jest obliczenie użyteczności stanów $U^\pi(s)$ związanych z posiadaną polityką $\pi(s)$. Użyteczności stanów zdefiniowane są jako wartości oczekiwane sumy nagród (dyskontowanych) otrzymanych przez agenta startującego z danego stanu, i poruszającego się zgodnie ze swoją polityką:

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

Agent może szacować tę wartość obliczając tzw. **nagrodę pozostałą** (*reward-to-go*) dla każdego stanu odwiedzonego w danym przebiegu. Na koniec przebiegu agent oblicza nagrodę pozostałą w stanie końcowym jako nagrodę otrzymaną w tym stanie. Następnie, cofając się w przebiegu, oblicza nagrody pozostałe dla wcześniejszych stanów jako sumy nagród otrzymanych na końcowym odcinku przebiegu.[1]

Na przykład, dla przebiegu:

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

otrzymujemy $R_{tg}(4,3) = 1, R_{tg}(3,3) = 0.96, R_{tg}(2,3) = 0.92, R_{tg}(1,3) = 0.88,$
$R_{tg}(1,2) = 0.84, R_{tg}(1,3) = 0.80,$ $R_{tg}(1,2) = 0.76, R_{tg}(1,1) = 0.72$

[1] W przypadku dyskontowania $\gamma \neq 1.0$ należy oczywiście obliczać sumy nagród odpowiednio dyskontowanych.

Posiadając wiele próbek (przebiegów) agent może przez proste uśrednianie określić kolejne przybliżenia wartości oczekiwanej użyteczności stanów, które w nieskończoności zbiegają się do właściwych wartości oczekiwanych.

To podejście jest poprawne, lecz niezbyt efektywne — wymaga dużej liczby przebiegów. Przedstawiona metoda określania użyteczności, stosując proste uśrednianie użyteczności stanów, pomija ważną własność procesów Markowa, tzn., że użyteczności stanów są związane z użytecznościami stanów sąsiednich.

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

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

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

Na przykład, w drugim przebiegu z powyższego przykładu algorytm określa użyteczność stanu (3,2) jako nagrodę pozostałą wyłącznie na podstawie tego przebiegu, ignorując fakt, że kolejnym stanem w tym przebiegu jest stan (3,3), który ma znaną już, istotnie wyższą użyteczność. Równanie Bellmana wiąże ze sobą użyteczności następujących po sobie stanów, lecz to podejście nie potrafi tego wykorzystać.

Adaptacyjne programowanie dynamiczne

Adaptacyjnym programowaniem dynamicznym (ADP) nazywamy proces podobny do programowania dynamicznego w połączeniu z uczeniem się modelu środowiska, czyli funkcji przejść stanów, i funkcji nagrody. Polega ono na zliczaniu przejść od pary stan-akcja do następnej akcji. Przebiegi uczące dostarczają nam serii uczącej takich przejść. Agent może określać ich prawdopodobieństwa jako ich częstotliwości występujące w przebiegach.

Na przykład, w podanych wcześniej przebiegach, w stanie (1,3) trzy razy wykonana została akcja $\boxed{\rightarrow}$ (Right), po czym dwa razy wynikowym stanem był (2,3). Zatem agent powinien określić $P((2,3)|(1,3), \textit{Right}) = \frac{2}{3}$.

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

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

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

Po wykonaniu każdego kroku agent aktualizuje użyteczności stanów rozwiązując równanie Bellmana (uproszczone) jednym z właściwych algorytmów. Równanie jest uproszczone, ponieważ znamy tylko rozkłady skutków akcji należących do polityki, i nie możemy obliczać najlepszej akcji w każdym stanie. Ponieważ chcemy obliczyć U^π to bierzemy właśnie te akcje.

Adaptacyjne programowanie dynamiczne — algorytm

function PASSIVE-ADP-AGENT($percept$) **returns** an action

inputs: $percept$, a percept indicating the current state s' and reward signal r'

persistent: π, a fixed policy

mdp, an MDP with model P, rewards R, discount γ

U, a table of utilities, initially empty

N_{sa}, a table of frequencies for state–action pairs, initially zero

$N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero

s, a, the previous state and action, initially null

if s' is new **then** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$

if s is not null **then**

increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$

for each t such that $N_{s'|sa}[t, s, a]$ is nonzero **do**

$P(t \mid s, a) \leftarrow N_{s'|sa}[t, s, a] \;/\; N_{sa}[s, a]$

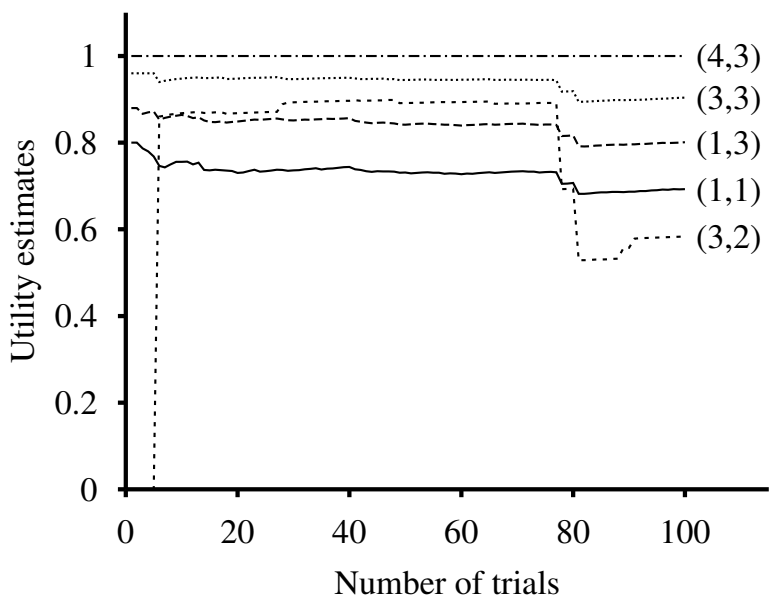
$U \leftarrow$ POLICY-EVALUATION(π, U, mdp)

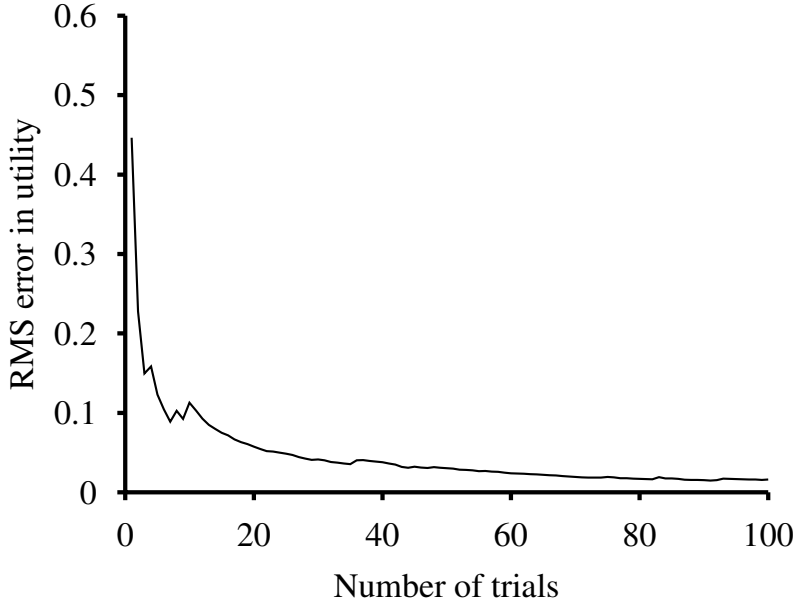
if s'.TERMINAL? **then** $s, a \leftarrow$ null **else** $s, a \leftarrow s', \pi[s']$

return a

Adaptacyjne programowanie dynamiczne — efektywność

Algorytm ADP aktualizuje wartości użyteczności najlepiej jak to jest możliwe, i stanowi w tym względzie standard do porównań dla innych algorytmów. Jednak procedura obliczania użyteczności przez rozwiązywanie układu równań (liniowych) może być niewykonalna dla wielu zmiennych (np. 10^{50} równań z 10^{50} niewiadomymi dla gry backgammon).





Powyższe wykresy ilustrują zbieżność dla przykładowego uczenia w środowisku 4×3. Należy dodać, że w tym przykładzie przebieg kończący się w „złym" stanie terminalnym pojawia się po raz pierwszy w przebiegu 78-ym, co skutkuje skokową aktualizacją niektórych użyteczności.

Metoda różnic czasowych

Zamiast każdorazowo rozwiązywać pełen układ równań ze względu na wartości użyteczności, można aktualizować te wartości aktualnie obserwowanymi wartościami wzmocnień. Tak funkcjonujący algorytm nazywa się metodą **różnic czasowych TD** (*temporal difference learning*):

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

W tym przypadku aktualizujemy użyteczność poprawką obliczaną na podstawie jednego zaobserwowanego przejścia stanów, a nie wartości oczekiwanej wszystkich przejść. Dlatego też poprawkę — różnicę pomiędzy użytecznością ruchu a użytecznością stanu — bierzemy zredukowaną współczynnikiem $\alpha < 1$. Powoduje to wprowadzanie małych poprawek po każdym ruchu. Jednocześnie poprawka zmierza do zera gdy użyteczność stanu zrównuje się z dyskontowaną użytecznością ruchu.

Zauważmy, że ta metoda nie wymaga posiadania modelu środowiska $P(s'|s, a)$, ani sama go nie oblicza.

Metoda różnic czasowych — algorytm

function PASSIVE-TD-AGENT($percept$) **returns** an action

inputs: $percept$, a percept indicating the current state s' and reward signal r'

persistent: π, a fixed policy

U, a table of utilities, initially empty

N_s, a table of frequencies for states, initially zero

s, a, r, the previous state, action, and reward, initially null

if s' is new **then** $U[s'] \leftarrow r'$

if s is not null **then**

increment $N_s[s]$

$U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma\, U[s'] - U[s])$

if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$

return a

Zbieżność metody różnic czasowych

Istnieje związek i podobieństwo pomiędzy algorytmami ADP i TD. O ile ten drugi dokonuje tylko lokalnych zmian w wartościach użyteczności, to ich średnie wartości zbiegają się do tych samych wartości co dla algorytmu ADP.

W przypadku uczenia wieloma przykładami przejść, częstotliwości występowania stanów zgadzają się z rozkładem prawdopodobieństw ich występowania i można wykazać, że wartości użyteczności będą się zbiegać do poprawnych wyników.

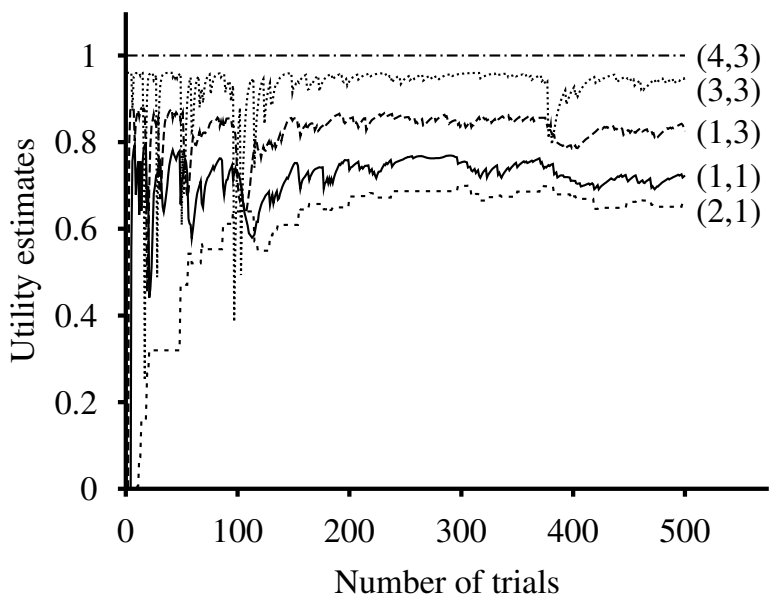
W tym celu parametr uczenia się α powinien zmniejszać się wraz ze zwiększaniem się liczby przetworzonych przebiegów. Dokładniej, wartości tego parametru powinny spełniać zależność:

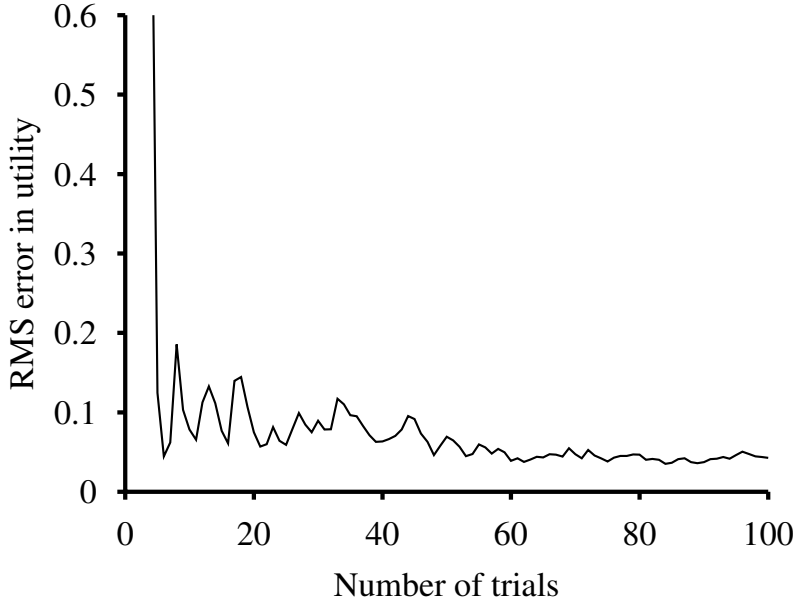
$$\sum_{n=1}^{\infty} \alpha(n) = \infty$$

oraz jednocześnie:

$$\sum_{n=1}^{\infty} \alpha^2(n) < \infty$$

Zbieżność przykładowego procesu uczenia dla środowiska 4×3:





Aktywne uczenie się ze wzmocnieniem

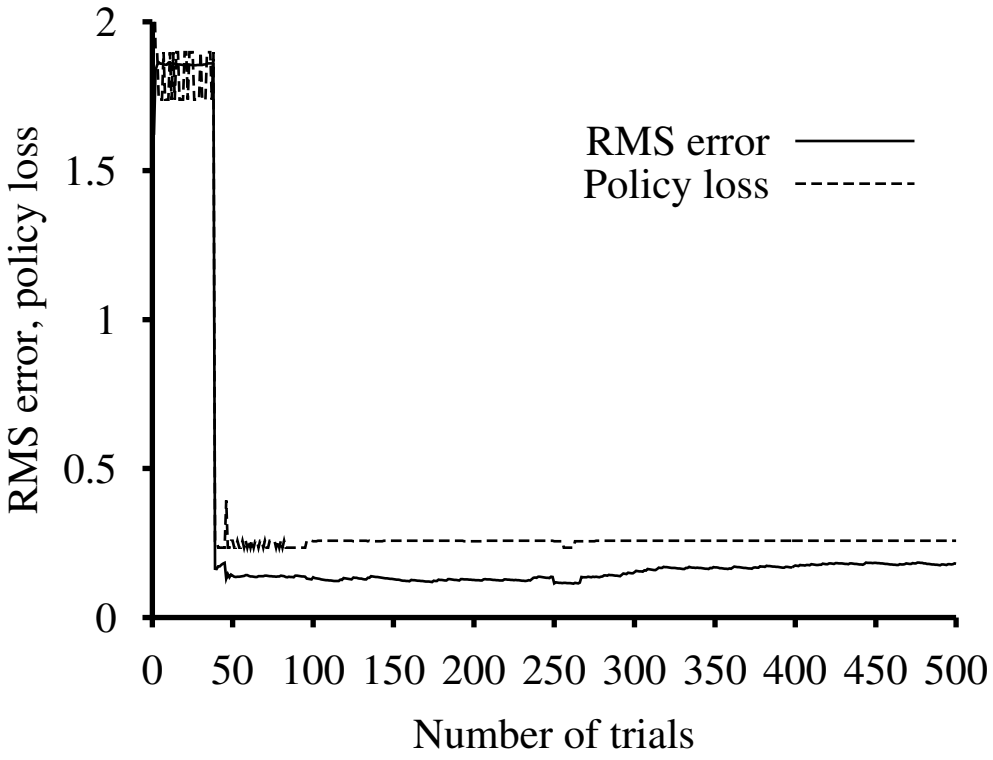
Co powinien zrobić agent, który nie posiada ustalonej polityki, albo który chciałby określić politykę optymalną?

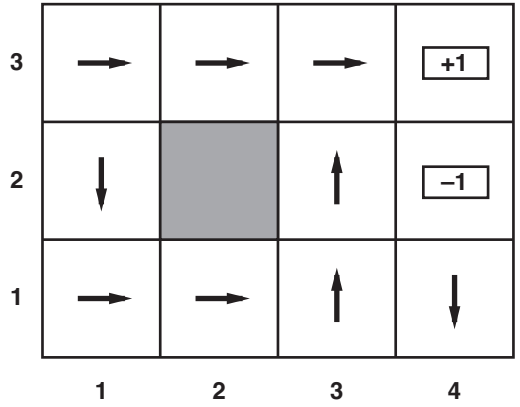
Najpierw powinien wyznaczyć kompletny model przejść dla wszystkich akcji. Przedstawiony wyżej algorytm ADP daje tę możliwość. Następnie należy wyznaczyć politykę optymalną, spełniającą poniższe równanie Bellmana, jak w zwykłym problemie decyzyjnym Markowa:

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

Agent może to zrobić algorytmem iteracji wartości lub iteracji polityki. Następnie, mając wyznaczoną optymalną politykę dla danego środowiska, może spokojnie przejść do jej realizacji.

Ale czy powinien tak zrobić?





Wykres po lewej pokazuje wynik uczenia się w pewnym eksperymencie. Agent znalazł bezpośrednią drogę do rozwiązania [+1] w przebiegu nr 39, lecz była to droga gorsza, wzdłuż stanów: (2,1), (3,1), (3,2), (3,3). Zdeterminowała jednak przyjętą przez agenta politykę optymalną po prawej. Okazuje się, że jest to sytuacja typowa, agent z rzadka tylko znajduje optymalną politykę preferującą drogą górną: (1,2), (1,3), (2,3), (3,3).

Eksploracja

Niestety, jeśli agent nie nauczy się poprawnego modelu środowiska w swoich początkowych przebiegach, to będzie następnie generował przebiegi zgodnie z polityką optymalną dla pewnego modelu, która może nie być globalnie optymalna dla danego środowiska.

Pojawia się tu kompromis pomiędzy **eksploatacją** posiadanej wiedzy a **eksploracją** środowiska. Agent nie może zbyt szybko zadowolić się wyuczonym modelem środowiska, i obliczoną dla niego optymalną strategią. Powinien próbować różnych możliwości.

Co więcej, musi wielokrotnie próbować wszystkich akcji we wszystkich stanach, jeśli chce uniknąć możliwości, że przypadkowa pechowa seria uniemożliwi mu odkrycie jakiegoś szczególnie dobrego ruchu. Jednak w końcu musi również zacząć poruszać się zgodnie z polityką optymalną, aby dostroić ją do specyficznych dla niej ścieżek.

Polityka eksploracji

Aby połączyć skuteczną eksplorację świata z eksploatacją posiadanej wiedzy agent powinien posiadać **politykę eksploracji**. Ma ona zagwarantować, że agent będzie on w stanie poznać wszystkie swoje możliwe akcje w stopniu wystarczającym do obliczenia swojej globalnie optymalnej polityki dla danego środowiska.

Prostą polityką eksploracji mogłoby być wykonywanie przypadkowych akcji we wszystkich stanach, z pewnym ustalonym prawdopodobieństwem, a w pozostałych przypadkach wykonywanie akcji uważanych za optymalne.

Jest to podejście poprawne, lecz wolno zbieżne. Lepiej byłoby preferować eksplorację niezbyt dobrze jeszcze poznanych par stan-akcja, jednocześnie unikając eksploracji par znanych już jako niezbyt korzystne.

Funkcja eksploracji

Sensowną politykę eksploracji można zbudować wprowadzając optymistyczne oszacowania użyteczności $U^+(s)$:

$$U^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} P(s'|s,a)U^+(s'), N(a,s)\right)$$

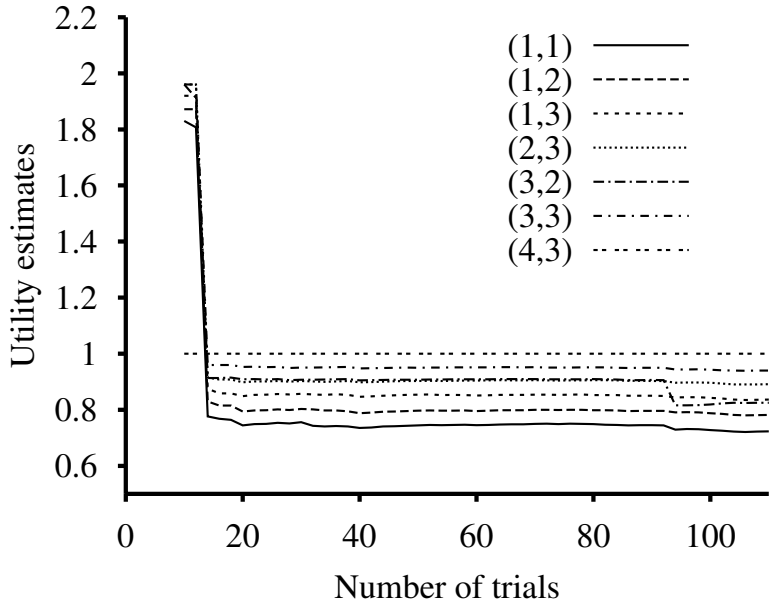
gdzie $N(a,s)$ jest liczbą wcześniej dokonanych wyborów akcji a w stanie s, natomiast $f(u,n)$ jest **funkcją eksploracji**, wyważającą preferencje dla zysku (dużych wartości u) i ciekawości (małych wartości n).

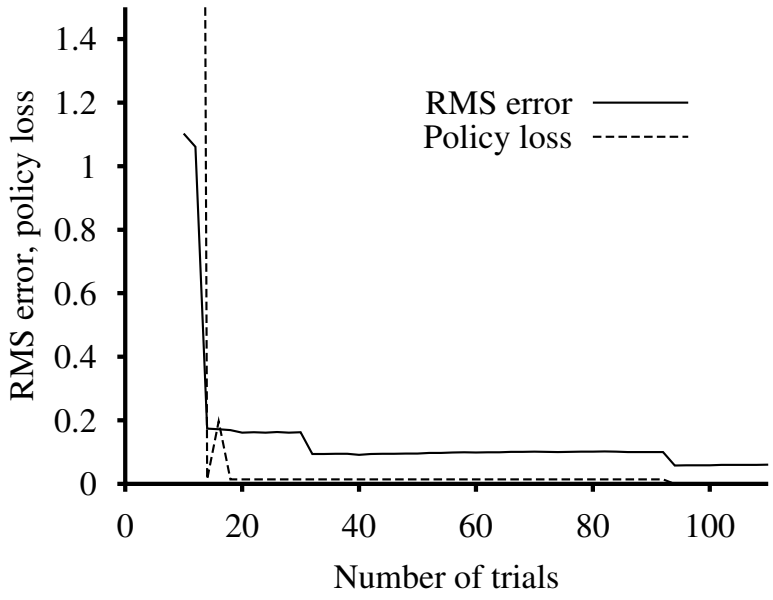
Oczywiście funkcja eksploracji f powinna być rosnąca ze względu na u i malejąca ze względu na n. Prostym przykładem funkcji f może być:

$$f(u,n) = \begin{cases} R^+ & \text{jeśli } n < N_e \\ u & \text{w przeciwnym wypadku} \end{cases}$$

gdzie R^+ oznacza optymistyczne oszacowanie najlepszej nagrody możliwej do otrzymania w którymkolwiek ze stanów, a N_e jest minimalną liczbą prób każdej pary stan-akcja, jaką agent będzie się starał wykonać.

Fakt, że we wzorze na aktualizację U^+ po prawej stronie występuje również U^+ jest istotny. Ponieważ stany i akcje wokół stanu początkowego będą wykonywane wiele razy, gdyby agent zastosował nieoptymistyczne obliczanie użyteczności, mógłby zacząć unikać tych stanów, i w konsekwencji zniechęcić się do wypuszczania się „dalej". Użycie wartości U^+ oznacza, że optymistyczne wartości generowane dla nowo eksplorowanych regionów będą propagowane wstecz, dzięki czemu akcje prowadzące do nieznanych jeszcze regionów będą szacowane wysoko, i tym samym preferowane.





Na lewym wykresie widać przebieg uczenia się agenta z eksploracją. Polityka bliska optymalnej została osiągnięta po 18 przebiegach. Zauważmy, że wartości użyteczności zbiegają się wolniej (*RMS error*) niż zostaje wyznaczona optymalna polityka (*policy loss*).

Aktywne uczenie się różnic czasowych

Metodę różnic czasowych można również zastosować do uczenia się aktywnego. Agent może nie posiadać ustalonej polityki, i nadal obliczać użyteczności stanów wykorzystując ten sam wzór co w przypadku pasywnym:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

Dzięki obliczanym użytecznościom agent może wyznaczać właściwe akcje w każdym stanie korzystając z użyteczności stanów sąsiednich. Można wykazać, że aktywny agent TD osiągnie te same wynikowe wartości użyteczności co aktywny agent ADP.

Metoda Q-learning

Alternatywną do poprzedniego wzoru metodą uczenia się różnic czasowych jest metoda **Q-learning**, która zamiast użyteczności uczy się reprezentacji akcja-wartość w postaci funkcji $Q(s, a)$. Ta funkcja wyraża wartość wykonania akcji a w stanie s, i jest związana z użytecznościami stanów wzorem:

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

Docelowe wartości Q spełniają równanie:

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

Powyższy wzór mógłby być wykorzystywany w procesie iteracyjnym jako wzór do aktualizacji wartości Q. Wymagałoby to jednak jednoczesnego uczenia się wartości Q i modelu w postaci funkcji P, która występuje we wzorze.

Q-learning — aktualizacja metodą różnic czasowych

Możliwa jest również aktualizacja lokalna funkcji Q będąca wariantem metody różnic czasowych i wyrażona poniższym wzorem aktualizacyjnym, obliczanym ilekroć akcja a jest wykonywana w stanie s prowadząc do stanu wynikowego s':

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

Algorytm Q-learning z metodą różnic czasowych zbiega się do rozwiązania znacznie wolniej niż algorytm ADP, ponieważ nie wymusza obliczenia pełnej spójności modelu (którego nie tworzy).

Pełny algorytm Q-learning z eksploracją

function Q-LEARNING-AGENT($percept$) **returns** an action
  inputs: $percept$, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return a

W ogólności aktywny agent uczący się metodą Q-learning wymaga zastosowania eksploracji tak samo jak w przypadku metody ADP. Stąd w algorytmie występuje funkcja eksploracji f i tablica częstości występowania akcji N. Przy zastosowaniu prostszej polityki eksploracji (np. wykonywanie określonej proporcji ruchów losowych) tablica N może nie być potrzebna.

SARSA — *State-Action-Reward-State-Action*

Istnieje pewien wariant algorytmu Q-learning z aktualizacją metodą różnic czasowych zwany SARSA (*State-Action-Reward-State-Action*):

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

W SARSA aktualizacja bierze pod uwagę pięć czynników: s, a, r, s', a'. O ile algorytm Q-learning aktualizuje na podstawie najlepszej akcji wybranej dla stanu osiągniętego przez akcję a, SARSA bierze pod uwagę to jaka akcja została w rzeczywistości wybrana. Zatem np. dla zachłannego agenta realizującego wyłącznie eksploatację te dwie metody byłyby identyczne.

Jednak w przypadku uczenia się z eksploracją różnica jest istotna. Metoda Q-learning jest metodą uczenia się **poza polityką** (*off-policy*), obliczającą najlepsze możliwe wartości Q, niezależnie od tego gdzie prowadzi nas realizowana polityka. Natomiast SARSA jest metodą **w polityce** (*on-policy*), odpowiednią dla agenta poruszającego się zgodnie z posiadaną polityką.

Q-learning jest bardziej elastycznym algorytmem, ponieważ pozwala agentowi uczyć się właściwego zachowania się nawet jeśli wykonuje on aktualnie politykę niezgodną z wyuczanymi wzorcami. Natomiast SARSA jest bardziej realistyczna, ponieważ na przykład, gdyby agent nie mógł w 100% kontrolować swojej polityki, to lepiej mu uczyć się wzorców zgodnych z tym co rzeczywiście będzie się z nim działo, zamiast uczyć się zgodnie z najlepszymi dla agenta wzorcami.

Zarówno Q-learning jak i SARSA są w stanie nauczyć się optymalnej polityki dla przykładowego środowiska 4x3, jednak wolniej niż ADP (w sensie liczby iteracji). Wynika to z faktu, że lokalne poprawki nie wymuszają spójności całej funkcji Q.

Porównując te metody można spojrzeć szerzej i zadać sobie pytanie, czy lepszym podejściem jest uczenie się modelu środowiska i funkcji użyteczności, czy bezpośrednie wyznaczanie odwzorowania stanów do akcji bez oglądania się na model środowiska.

Jest to w rzeczywistości jedno z fundamentalnych pytań jak budować sztuczną inteligencję. Przez wiele lat początkowego rozwoju tej dziedziny wiedzy dominował paradygmat **systemów opartych na wiedzy** (*knowledge-based*), postulujących konieczność budowy modeli deklaratywnych. Fakt, że powstają metody bezmodelowe takie jak Q-learning sugeruje, że być może było to niepotrzebne. Jednak dla niektórych bardziej złożonych zagadnień podejście z modelem sprawdza się lepiej, zatem kwestia pozostaje nierozstrzygnięta.

Uogólnianie w uczeniu się ze wzmocnieniem

Omówione powyżej algorytmy uczenia się ze wzmocnieniem zakładają jawną reprezentację funkcji $U(s)$ lub $Q(s)$ taką jak np. reprezentacja tablicowa. Może to być praktyczne tylko do pewnej wielkości zagadnienia.

Na przykład, dla zagadnień o bardzo dużej liczbie stanów (np. $\gg 10^{20}$ dla gier takich jak szachy lub backgammon), trudno wyobrazić sobie wykonanie wystarczającej liczby przebiegów uczących aby odwiedzić każdy stan wiele razy. Konieczne jest zastosowanie jakiejś metody generalizacji (uogólniania), która pozwoliłaby generować skuteczną politykę na podstawie małej części przebadanej przestrzeni stanów.

Aproksymacja funkcji

Jedną z takich metod jest **aproksymacja funkcji**, polegająca na zapisie badanej funkcji (np. U) w postaci nietablicowej, np. wyrażeniu jej jakąś formułą skończoną. Podobnie jak w konstrukcji funkcji heurystycznych, można zastosować liniową kombinację jakichś cech stanu (zwanych również atrybutami stanu):

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + ... + \theta_n f_n(s)$$

Algorytm uczenia się ze wzmocnieniem uczyłby się wektora współczynników $\theta =< \theta_1, \theta_2, ..., \theta_n >$ tak by funkcja oceny $\hat{U}_\theta$ przybliżała możliwie dobrze rzeczywistą funkcję użyteczności stanów.

Podejście to nazywa się aproksymacją funkcji, ponieważ nie ma pewności, że rzeczywistą funkcję oceny da się wyrazić tego typu formułą. Jakkolwiek wydaje się wątpliwe by np. optymalną politykę dla gry w szachy dało się wyrazić funkcją z kilkunastoma współczynnikami, to jest zupełnie możliwe by osiągnąć w ten sposób dobry poziom gry.

Istotą podejścia jest jednak nie przybliżenie mniejszą liczbą współczynników funkcji,

która w rzeczywistości być może wymaga ich wielokrotnie więcej, ale uogólnianie, czyli generowanie polityki dla wszystkich stanów na podstawie analizy małej ich części.

Np. w eksperymentach przeprowadzonych z grą backgammon, udało się nauczyć gracza poziomu gry porównywalnego z ludzkimi na podstawie prób analizujących jeden na 10^{12} stanów.

Oczywiście, sukces uczenia się ze wzmocnieniem w takich przypadkach zależy od trafnego wybrania funkcji aproksymującej. Jeśli żadna kombinacja wybranych cech nie może dać dobrej strategii gry, to żadna metoda uczenia jej nie wygeneruje. Z kolei, wybranie bardzo rozbudowanej funkcji z dużą liczbą cech i współczynników zwiększa szanse na sukces, ale kosztem wolniejszej zbieżności i zarazem wolniejszego procesu uczenia.

Korekta parametrów funkcji

Aby umożliwić uczenie się na bieżąco (*on-line learning*) niezbędna jest jakaś metoda korekty parametrów na podstawie wartości wzmocnień otrzymywanych po każdym przebiegu (albo po każdym kroku).

Na przykład, jeśli $u_j(s)$ jest wartością pozostałej nagrody dla stanu s w j-tym przebiegu uczącym, to błąd aproksymacji funkcji użyteczności można obliczać jako:

$$E_j = \frac{(\hat{U}_\theta(s) - u_j(s))^2}{2}$$

Dynamika zmiany tego błędu ze względu na parametr θ_i jest określona jako $\partial E_j / \partial \theta_i$, zatem aby skorygować ten parametr w kierunku zmniejszenia błędu, właściwą formułą na poprawkę jest:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha(u_j(s) - \hat{U}_\theta(s)) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

Powyższy wzór zwany jest **regułą Widrow'a-Hoff'a** albo **regułą delta**.

Przykład

Na przykład, dla środowiska 4x3 funkcja użyteczności stanów mogłaby być aproksymowana liniową kombinacją współrzędnych:

$$\hat{U}_\theta(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$

Poprawki zgodne z regułą delta będą teraz dane przez:

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha(u_j(s) - \hat{U}_\theta(s)) \\
\theta_1 &\leftarrow \theta_1 + \alpha(u_j(s) - \hat{U}_\theta(s))x \\
\theta_2 &\leftarrow \theta_2 + \alpha(u_j(s) - \hat{U}_\theta(s))y
\end{aligned}$$

Przyjmując przykładowo $\theta =< \theta_0, \theta_1, \theta_2 >=< 0.5, 0.2, 0.1 >$ otrzymujemy początkowe przybliżenie $\hat{U}_\theta(1,1) = 0.8$. Jeśli po wykonaniu przebiegu uczącego obliczylibyśmy np. $u_j(1,1) = 0.72$ to wszystkie współczynniki θ_0, θ_1, θ_2 zostałyby obniżone o 0.08α, co zmniejszyłoby błąd dla stanu (1,1). Oczywiście, w ten sposób zmieniłaby się cała funkcja $\hat{U}_\theta(s)$, co jest istotą uogólniania.

Zastosowanie różnic czasowych

Można również realizować poprawki metodą różnic czasowych.

$$\theta_i \leftarrow \theta_i + \alpha[R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

$$\theta_i \leftarrow \theta_i + \alpha[R(s) + \gamma \max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)] \frac{\partial \hat{Q}_\theta(s, a)}{\partial \theta_i}$$

Materiały

David Silver: Kurs uczenia ze wzmocnieniem (2015, University College London):
`http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`