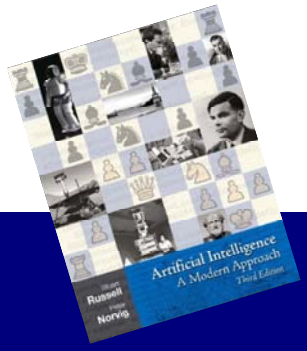


Umělá inteligence II

12

Roman Barták, KTIML

roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Motivace

- Uvažujme agenta, který se učí hrát šachy.
- Při **učení s učitelem** je potřeba pro každou pozici doporučit tah.
  - takto úplná informace je málokdy dostupná
- Při **učení bez učitele** je možné se naučit model vlastních tahů a reakce na oponentovy tahy.
  - bez **zpětné vazby** ale agent nemá žádné podklady pro rozhodnutí, který tah je dobrý a který ne

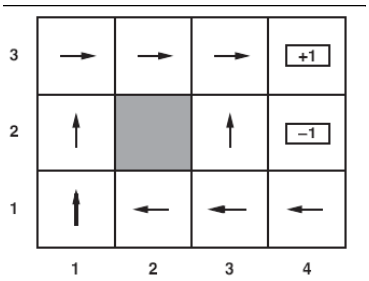
Zpětná vazba

- Typem **zpětné vazby** je ocenění výkonu
  - u her ho získáme na konci, podle toho, kdo vyhrál
  - u řady problémů je ocenění výkonu dostupné po každém tahu
- Ocenění výkonu je **součástí vstupu** jako další vjem, je ale potřeba ho od jiných senzorických vstupů jasně rozlišovat
  - pocit hladu a bolesti je negativní ocenění
  - najezení a potěšení je pozitivní ocenění

Zpětnovazební učení

- **Zpětnovazební učení** (reinforced learning) je ztělesněním jádra umělé inteligence
  - agent je umístěn do prostředí a musí se v něm naučit úspěšně chovat
  - u řady komplexních domén je to jediný efektivní způsob, jak navrhnout agentovu funkci
- Budeme uvažovat tři typy agentů
  - **agent se žádostmi** (utility-based) se učí funkci užitku pro stavy
    - potřebuje mít model prostředí, aby věděl kam vedou akce
  - **agent s Q-učením** se učí akce na základě očekávaného užitku
  - **reflexní agent** se učí strategii, jakou akci použít na daný stav
- **Pasivní učení** pracuje s pevnou strategií a učí se užitek stavů/akcí.
- Při **aktivním učení** se agent učí, co dělat
  - zahrnuje jistou formu **prozkoumávání** prostředí

Pasivní učení



- Agentova **strategie** π je předem **dána** ($\pi(s)$ je akce, kterou agent provede ve stavu s).
- Cílem je **ohodnotit kvalitu** strategie - zjistit očekávaný užitek
  $U^{\pi}(s) = E[\Sigma_{t=0,\ldots,\infty} \gamma^t.R(s_t)]$
- Agent **nezná přechodový model** $P(s`|s,a)$ ani **oceňovací funkci** $R(s)$.
- Základní postup:
  - agent zkouší procházet prostředí použitím strategie π
  - v každém stavu získá formou vjemu jeho ocenění

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

Přímý model

pro pasivní učení

- Pro stavy v každém vzorku vypočteme **ocenění cesty do cíle** (reward-to-go)
  - pro stav (1,1) máme v prvním vzorku ocenění cesty 0.72
  - pro stav (1,2) máme v první cestě dva vzorky 0.76 a 0.84
- Stav se může vyskytovat ve více vzorcích – uděláme průměr ocenění cest do cíle.
- Jedná se tak vlastně o učení s učitelem (vstup = stav, výstup = užitek)
- **Základní problém**
  - stavy oceňujeme jako kdyby byly nezávislé
  - mezi stavy je ale vztah daný Bellmanovými rovnicemi
    $U^{\pi}(s) = R(s) + \gamma \Sigma_{s`} P(s`|s, \pi(s)) U^{\pi}(s`)$
  - prohledáváme zbytečně velký prostor hypotéz (i hypotézy, které nesplňují Bellmanovy rovnice), pomalejší konvergence

ADP
pro pasivní učení

- **Adaptivní dynamické programování** bere v úvahu Bellmanovy rovnice.
- Budeme se učit
  - **přechodový model** P(s`|s, π(s))
    - podle frekvencí daného přechodu, např. P((2,3)|(1,3), Right) = 2/3
  - **ocenění stavů** R(s)
    - součást vjemu
- **Užitek** dopočteme z Bellmanových rovnic např. iterací hodnot/strategií.

ADP
algoritmus

```
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'] ← r'; R[s'] ← 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 | s, a) ← N_s'|sa[t, s, a] / N_sa[s, a]
  U ← POLICY-EVALUATION(π, U, mdp)
  if s'.TERMINAL? then s, a ← null else s, a ← s', π[s']
  return a
```

Temporální diference

pro pasivní učení

- Místo učení se přechodové tabulky můžeme **přímo upravovat užitek stavů** tak, aby odpovídal Bellmanovým rovnicím.
- Příklad:
  - uvažujme přechod (1,3) $\Rightarrow$ (2,3)
  - počáteční nastavení
    $U^{\pi}(1,3) = 0.84$ a $U^{\pi}(2,3) = 0.92$
  - měl by platit vztah (pokud se vždy užije stejný přechod a $\gamma = 1$)
    $U^{\pi}(1,3) = -0.04 + U^{\pi}(2,3)$
  - mělo by tedy platit
    $U^{\pi}(1,3) = 0.88$
  - současný odhad $U^{\pi}(1,3)$ bychom měli zvětšit
- Obecně použijeme následující update:
  $U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha.(R(s) + \gamma.U^{\pi}(s') - U^{\pi}(s))$
- Popsaný postup se nazývá **temporální diference**.

Temporální diference

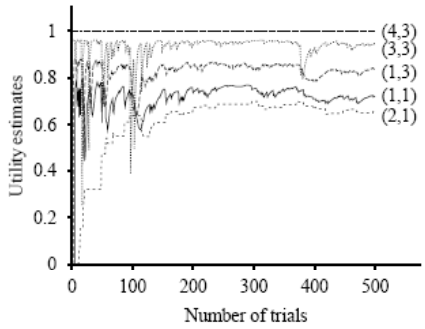
algoritmus

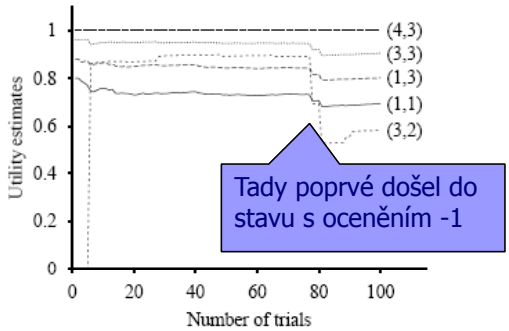
```
function PASSIVE-TD-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
              U, a table of utilities, initially empty
              N_s, a table of frequencies for states, initially zero
              s, a, r, the previous state, action, and reward, initially null

  if s' is new then U[s'] ← r'
  if s is not null then
     increment N_s[s]
     U[s] ← U[s] + α(N_s[s])(r + γ U[s'] − U[s])
  if s'.TERMINAL? then s, a, r ← null else s, a, r ← s', π[s'], r'
  return a
```

ADP vs. TD

- ADP i TD dělají **lokální změny** tak, aby užitek stavu odpovídal užitku okolních stavů
- **Temporální diference**
  - nepotřebuje přechodový model
  - pro update používá pozorovaného následníka
  - v kroku dělá jedinou změnu
- **Adaptivní dynamické programování**
  - pro update používá všechny následníky
  - změnu propaguje, aby byla udržena konzistence





Aktivní učení

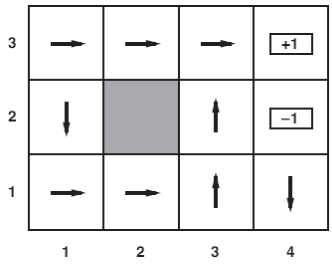
- Při pasivním učení je dána strategie určující chování agenta.
- **Aktivní agent** se sám rozhoduje, jaké akce provede (učí se strategii).
- Zkusíme rozšířit pasivního ADP agenta (potřebuje model přechodů)
  - užitek je definován optimální strategií dle Bellmanových rovnic

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

  - můžeme přímo použít ve funkci výpočtu užitku v ADP
- Co bude agent dělat v jednom kroku?
  - na základě vypočteného užitku U zvolí nejlepší akci
  - akci doporučenou optimální strategií provede
  - Nebo je něco lepšího?

Hladový agent

- Strategie, kterou našel aktivní ADP agent
- Tato strategie **není optimální**! Co se stalo?



- Agent našel cestu přes (2,1), (3,1), (3,2), (3,3) vedoucí k cíli s oceněním +1.
- Při dalších iteracích se této cesty (strategie) držel.
- Nenašel proto jinou, lepší cestu přes (1,2), (1,3), (2,3), (3,3).
- Hovoříme o **hladovém** (greedy) **agentovi**.

Hladový agent

vlastnosti

- Jak je možné, že **výběr optimální akce nedává optimální strategii**?
  - akce je vybrána podle naučeného prostředí ne podle skutečného prostředí
  - akce přispívá nejen do funkce užitku, ale také do zdokonalení modelu světa (tento aspekt ADP agent ignoroval)
  - zlepšením modelu můžeme v budoucnu získat větší užitek než ten, který se v současnosti jeví jako nejlepší
- Agent potřebuje do jisté míry dělat **průzkum prostředí**.

Průzkum prostředí

- **Jaký poměr má agent volit mezi průzkumem prostředí a jeho využitím?**
  - **čistý průzkum** zlepšuje model, ale model se nikdy nevyužije
  - **čisté využití** vede k sub-optimální strategii
- **Základní princip**
  - zpočátku je lepší více prozkoumávat prostředí s vírou v nalezení lepších cest
  - s větším porozuměním prostředí stačí menší průzkum
- **Problém n-rukého bandity**
  - herní automat s n-pákami (nebo n automatů)
  - Jakou páku preferovat?
    - páku, kde už jsem někdy vyhrál (ale málo) nebo raději páku, kterou jsem ještě nezkoušel?

Strategie průzkumu

- s poměrem 1/t volíme akci náhodně, jinak volíme akci dle hladové strategie
  - konverguje k optimální strategii, ale pomalu
- lepší je **preferovat málo vyzkoušené akce** a **vyhnout se akcím**, u kterých věříme, že mají **malý užitek**
  - neprozkoumaným akcím zvýšíme ocenění
  - při iteraci hodnot použijeme následující pravidlo

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

    - N(s,a) je počet, kolikrát byla akce a použita na stav s
    - $U^+(s)$ je optimistický odhad hodnoty užitku
    - f(u,n) je **funkce průzkumu** klesající v u a rostoucí v n
      - např. $f(u,n) = R^+$ pokud $n<N_e$, jinak u (R^+ je optimistický odhad největšího možného užitku)
  - použití U^+ je důležité, protože vysoké hodnoty se propagují z neprozkoumaných oblastí
    - agent preferuje nejen neznámé akce ale také akce vedoucí do neprozkoumaných oblastí

Q-učení

- Jak by vypadal **aktivní TD agent**?
  - použije stejné učení se přechodového modelu jako ADP agent
  - pravidlo pro update funkce užitku zůstane stejné
    $U(s) \leftarrow U(s) + \alpha.(R(s) + \gamma.U(s') - U(s))$
- Alternativním přístupem je **Q-učení**
  - Q(s,a) – hodnota provedení akce a ve stavu s
  - alternativní způsob uložení funkce užitku
    - $U(s) = \max_a Q(s,a)$
  - nepotřebujeme přechodový model P(s`|s, a)
    - **bez-modelová metoda**
  - Q-funkce v rovnovážném stavu musí splňovat vztah
    - $Q(s,a) = R(s) + \gamma \Sigma_{s'} P(s'|s, a) \max_{a'} Q(s',a')$
    - tady je ale použitý model P(s`|s, a)
  - při Q-učení aplikujeme pravidlo pro TD-učení na Q-funkci
    - $Q(s,a) \leftarrow Q(s,a) + \alpha.(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$
    - použije se když agent přejde ze stavu s do stavu s` akcí a

Q-učení
algoritmus

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

SARSA

State-Action-Reward-State-Action

- blízký příbuzný Q-učení s následujícím updatovacím pravidlem

  $Q(s,a) \leftarrow Q(s,a) + \alpha.(R(s) + \gamma.Q(s`,a`) - Q(s,a))$
- použije se na celou pětici s,a,r,s`,a`, tj. pro update se čeká na další akci
- SARSA vs. Q-učení
  - pro hladového agenta jsou Q-učení a SARSA totožné (vybere vždy akci a` maximalizující Q(s`,a`))
  - při průzkumu se výrazně liší
    - Q-učení se nestará o aktuální strategii a agent funguje dobře i s náhodnou strategií
    - SARSA je realističtější, protože se učí Q-funkci na základě toho, co se skutečně stalo, ne co by se mohlo stát
      - naučí se i reakci na další agenty

Srovnání

- Q-učení a SARSA najdou optimální strategii, ale pomaleji než ADP agent
  - lokální změny nejsou propagovány dále a není tak v každém kroku zajištěna konzistence Q-hodnot
- **Je lepší agent s modelem (ADP) nebo bez modelu (Q-učení, SARSA)?**
  - výzkum v UI tradičně tíhne spíše ke **znalostním technikám**, tj. při reprezentaci agentovy funkce používat reprezentaci nějakých aspektů prostředí
  - Q-učení ukazuje, že lze dělat agentové funkce i bez modelu prostředí