József Attila Tudományegyetem
Természettudományi Kar

A megerősítéses tanulás elvén működő tanulórendszer elemzése

Diplomamunka

Fodor István Balázs
Programtervező–matematikus hallgató

Témavezető: dr. habil. Lőrincz András
Tudományos főmunkatárs

Belső konzulens: Kocsor András

Szeged, 1999.

Tartalomjegyzék

Bevezetés **ii**

Jelölések **iii**

1. A megerősítéses tanulás **1**

1.1. A feladat 1

1.1.1. Ügynök-környezet kapcsolat, politika definíciója 1

1.1.2. Célok és jutalmak 2

1.1.3. Hozam 2

1.1.4. Markov tulajdonság, Markov döntési folyamat 3

1.1.5. Bellman egyenlet 5

1.1.6. Optimális politika, Optimális értékelő függvény, Bellman optimalitási egyenlet 6

1.2. Alapvető megoldási módszerek 10

1.2.1. Dinamikus programozás 10

1.2.2. Időbeli-differencia módszere 18

1.2.3. Az emlékeztető nyomok módszere 21

1.3. Függvény approximátorok 32

1.3.1. Az értékelő függvény becslése függvény-approximátorokkal 32

1.3.2. Gradiens keresési eljárás 33

1.3.3. Lineáris eljárás 35

1.4. Bibliográfiai és történeti megjegyzések 38

2. Ritka reprezentáció **40**

3. Khepera robot szimulátor **43**
3.1. A világ leírása 44
3.2. A robot leírása 44
3.2.1. A motor modellje 45
3.2.2. A szenzorok modellje 45
3.2.3. A robot vezérlése 46

4. A megerősítéses tanítás módszer alkalmazása **47**
4.1. A környezet modellje 47
4.1.1. Az állapot reprezentációja 47
4.1.2. A jutalom függvény 48
4.2. Az ügynök modellje 48
4.2.1. Állapotot értékelő függvény közelítése 48
4.2.2. Pályatervezése 50
4.3. A szimuláció felépítése 52

5. Eredmények **53**
5.1. Ritka reprezentáció kialakítása 53
5.2. Az értékelő függvény tanulása 57
5.3. A szimuláció eredményei 58
5.4. Konklúzió 60

Ábrák jegyzéke **62**

Táblázatok jegyzéke **63**

Irodalomjegyzék **64**

Bevezetés

A megerősítéses tanulás és a mesterséges neuronhálózatok összekapcsolásának egyik leghíresebb és legtöbbet emlegetett példája Gerald Tesauro backgammon programja, a TD-Gammon. A TD-Gammon egy öntanuló program, amely a minimális előzetesen belekódolt ismeretek ellenére rendkívül jó szintet ért el csak azáltal, hogy játszmák százezreit játszotta saját magával és közben ezekből a játszmákból tanult. A tanuló-algoritmusa a TD(λ) algoritmus volt és egy többrétegű perceptront használt nemlineáris függvényapproximátorként az állapotot értékelő függvény közelítésére.

Tesauro a TD-Gammon első verziója után továbblépett, az újabb verziókba már előzetes backgammon ismereteket is vitt (bizonyos számolható jellemzőket belekódolt a hálózat bemenetébe), ezen felül 2-3 lépés mélységű heurisztikus keresést alkalmazott. Az eredmény: a TD-Gammon 3.0 a világbajnokokkal is felveszi a versenyt. Ez a program olyan, eddig ismeretlen megnyitásokat fedezett fel, amiket a legjobb játékosok is átvenni kényszerültek.

E diplomamunka célja egy hasonló konstrukció (TD(0) tanuló algoritmus függvény approximátorral) vizsgálata, a megoldandó probléma teljesen más jellegű. A célunk a következő: adott egy robot, s hozzá olyan vezérlő rendszert akarunk építeni, amely a robotot adott, labirintushoz hasonló pályán a fal mellett haladásra késztet. A különbseg magába a konstrukcióban a következő : a függvényapproximátor szerepét a ritka reprezentáció tölti be.

A dolgozat első fejezetében áttekintjük a megerősítéses tanulás alapfogalmait és algoritmusait, a második fejezeben a függvényapproximátorként alkalmazott sparse reprezentációt, a harmadik fejezetben a Khepera robot szimulátort.

Ezen ismeretekre alapozva a negyedik fejezetben bemutatjuk az általunk alkalmazott modellet illetve az ötdik fejezetben a futtatások eredményeit. S végül megpróbálunk következtetéseket levonni a tesztek alapján és vázolunk néhány általunk érdekesnek gondolt kutatási irányt.

Jelölések

t	diszkrét idő
T	az epizód utolsó periódusa
s_t	állapot a t-edik időpillanatban
a_t	akció a t-edik időpillanatban
r_t	jutalom a a t-edik időpillanatban (s_t, a_{t-1}, s_{t-1} függvénye)
R_t	a t-edik időpillanatbeli (az összegyűjtött diszkontált) hozam
$R_t^{(n)}$	n-lépés hozam
R_t^{λ}	λ-hozam
π	politika (a döntéseket meghatározó szabályok)
$\pi(s)$	akció választás az s állapot és determinisztikus π politika esetén
$\pi(s,a)$	az a akció választási valószínűsége az s állapot és sztohasztikus π politika esetén
$\mathcal{S}$	a nemterminális állapotok halmaza
$\mathcal{S}^+$	az összes állapot halmaza (a terminális állapotokkal együtt)
$\mathcal{A}(s)$	az összes lehetséges s állapotbeli akciók halmaza
$\mathcal{P}_{ss'}^a$	s' állapotba kerülés valószínűsége s állapot és a akció esetén
$\mathcal{R}_{ss'}^a$	a várható jutalom s-ből s' állapotba jutáskor a akció mellett
$V^\pi(s)$	az s állapot értéke π politika esetén (várható hozam)
$V^*(s)$	az s állapot értéke optimális politika esetén
V, V_t	V^π vagy V^* közelítése
$\mathcal{Q}^\pi(s,a)$	az a akció értéke s állapot és π politika esetén
$\mathcal{Q}^*(s,a)$	az a akció értéke s állapot és optimális politika esetén
$\mathcal{Q}$, $\mathcal{Q}_t$	$\mathcal{Q}^\pi$ vagy $\mathcal{Q}^*$ közelítése

$\vec{\theta}_t$	V_t-hez vagy Q_t-hez tartozó paramétervektor
$\vec{\phi}_s$	s állapotot reprezentáló vektor
δ_t	átmeneti–differencia hibaértéke a t-edik időpillanatban
$e_t(s)$	"emlékeztető nyom" (eligiblility trace) s állapotban t-edik idő pillanatban
$e_t(s,a)$	emlékezteteő nyom az állapot–akció párra
γ	diszkontálási paraméter
ϵ	véletlen akció választási valószínűség ϵ-mohó politika esetén
α, β	lépésköz paraméter
λ	emlékeztető nyom paramétere

1. fejezet

A megerősítéses tanulás

1.1. A feladat

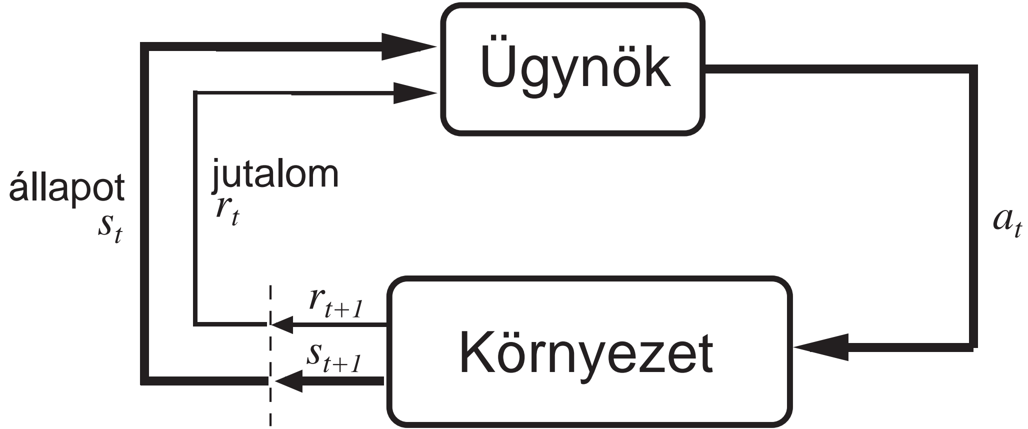
A természetes tanulás folyamatán gondolkozva az első dolog ami eszünkbe juthat az, hogy tudásunk legfőbb forrása a *környezet*ünkkel való kapcsolat. Ekkor valójában nincs világosan megfogalmazható tanító, – aki megmondja, hogy mit kell tenni, – azonban a közvetlen szenzoros kapcsolat alapján megtanulhatóak az ok-okozati viszonyok és az, hogy hogyan érhetjük el a céljainkat. A következőkben a kapcsolatok alapján történő tanulás egy számítógépes modelljét mutatjuk be. Ahelyett, hogy közvetlenül megvalósítanánk az állati, illetve az emberi tanulási folyamatot, megvizsgáljuk az idealizált tanulási helyzeteket és kiértékeljük a különböző tanulási módszerek hatékonyságát. Ezt az eljárást – amely egy célokra összpontosító, kapcsolatok alapján tanuló módszer – hívják *megerősítéses tanulás*nak *(Reinforcement Learning)*.

1.1.1. Ügynök-környezet kapcsolat, politika definíciója

A megerősítéses tanulást megvalósító modell két alapvető részből épül fel: egy tanuló, döntéshozó blokkból, az úgynevezett *ügynök*ből *(Agent)*, illetve a vele kapcsolatba lévő *környezet*ből *(Environment)*. Ez a kapcsolat folytonos: az ügynök választ egy akciót, a környezet válaszol ezekre az akciókra és megadja az új helyzetet az ügynöknek. Ezenfelül a környezet ad egy speciális számértéket, az úgynevezett *jutalmat (Reward)*, amelyet az ügynök maximalizálni próbál.

Részletezve: az ügynök és a környezet kapcsolatban van minden egyes diszkrét t $(= 0, 1, 2, 3, \dots)$ időpillanatban, az ügynök megkapja a környezet aktuális *állapot*át *(State)* s_t-t, ahol $s_t \in \mathcal{S}$ és $\mathcal{S}$ a lehetséges állapotok halmaza, és választ egy a_t akciót, ahol $a_t \in \mathcal{A}(s_t)$ és $\mathcal{A}(s_t)$ az s_t állapot esetén választható akciók halmaza. Egy lépéssel később, az ügynök

saját akciójának következményeként kap egy r_{t+1} numerikus értékű jutalmat, ahol $r_{t+1} \in \mathcal{R}$ ($\mathcal{R}$ a várható jutalmak halmaza), illetve a környezet az s_{t+1} állapotba kerül.



1.1. ábra. Az ügynök-környezet kapcsolat.

Az ágens minden egyes lépésnél egy leképezés alapján cselekszik. és az összes lehetséges akció választási valószínűsége között. Ezt a leképezést hívják az ügynök *politiká*jának *(Policy)* és jelölik π_t-vel, ahol $\pi(a, s)$ $s_t = s$ esetén $a = a_t$ választásának a valószínűségét adja meg. A megerősítéses tanulási módszerek azt határozzák meg, hogy a tapasztalatok alapján az ügynök hogyan változtatja politikáját.

1.1.2. Célok és jutalmak

A megerősítéses tanulásnál az ügynök szándéka vagy *célja (Goal)* formalizálva van egy speciális *jutalom* jel *(Reward)* formájában, amelyet az ügynök kap a környezettől. Minden egyes időközben a jutalom egy egyszerű r_t számérték, $r_t \in \mathcal{R}$. Az ügynök célja tulajonképpen az, hogy maximalizálja a várható kumulált jutalom mennyiségét, ami nem a pillanatnyi jutalomra értendő, hanem a hosszú futási idő alatti összegzett jutalomra. Az, hogy a jutalmat használjuk a célok formalizálására az első ránézésre erős megkötésnek tűnik, azonban a gyakorlatban bizonyítottan rugalmas.

1.1.3. Hozam

Mint ahogy már említetük az ügynök célja az, hogy maximalizálja a hosszú idő alatti várható jutalom mértékét. Tulajdonképpen a várható *hozam (Return)* (R_t) maximális értékét keressük. Jelölje r_{t+1}, r_{t+2}, $r_{t+3}, \ldots$ a t időpillanat utáni jutalmakat, ekkor a hozam a legegyszerűbb formában:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T \,, \tag{1.1}$$

ahol T az utolsó lépés ideje.
A várható hozam ilyen formájú megfogalmazása azokban az esetekben alkalmazható jól, ahol valamilyen természetes jelölése van az utolsó lépésnek, amikor az ügynök-környezet kapcsolat természetes módon széttörhető sorozatok halmazává, amelyeket *epizód*oknak *(Episode)* neveznek. Mindegyik epizód egy speciális állapotban az úgynevezett *terminális állapot*ban fejeződik be. Az olyan folyamatokat, amelyeket természetes módon alsorozatokra bontható *epizodikus folyamat*oknak nevezük.

Sok esetben az ügynök-környezet kapcsolat nem bontható természetes módon epizódokra, de minden határon túl folyamatosak; ezeket *folytatható folyamat*oknak nevezzük. Ebben az esteben (1.1) pontban megfogalmazott hozam definíció nem megfelelő, hisz $T = \infty$, és így hozam maximális értéke könnyen végtelen lehet. Ekkor egy másfajta hozamszámítási módot alkalmaznak, az úgynevezett diszkontálási eljárást. A diszkontált hozam a következőképpen néz ki :

$$R_t = r_{t+1} + \gamma\, r_{t+2} + \gamma\, r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,, \tag{1.2}$$

ahol γ $(0 \leq \gamma < 1)$ a diszkontálási paraméter.
Ha $\gamma < 1$, akkor (1.2) formulának véges értéke van r_k-k megadása esetén. Ha $\gamma = 0$, akkor az ügynök "rövidlátó", azaz csak a pillanyatni várható jutalom értékét akarja optimalizálni. Ha $\gamma \approx 1$ $(\gamma < 1)$ a jövőbeli jutalmak sokkal jobban érvényesülnek a hozam számításánál, azaz a rendszer "távolabb látó" lesz.

A két különböző esetet összefoglalhatjuk egy egységes hozamfüggvény formájában:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \,. \tag{1.3}$$

Abban az esetben, ha T véges és $\gamma = 1$, akkor az epizódikus, ha $T = \infty$ és $0 \leq \gamma < 1$, akkor a folytatható folyamatra vonatkozó hozamfüggvényt kapjuk.

1.1.4. Markov tulajdonság, Markov döntési folyamat

A megerősítéses tanulás módszer szerkezeti felépítésében az ügynök döntései a környezet által adott jelnek, az *állapot*nak *(State)* a függvénye. Természetesen ez az állapotjel magába

foglalja a pillanatnyi érzékelés eredményét, például a szenzorok által mért értékeket, de ezenkívül mást is tartalmazhat. Felépülhet igen bonyolult módon az érzékelések sorozata alapján, de lehet olyan állapotjel definíció amelynél a jelenlegi állapot leírja a rendszert, anélkül, hogy az előzőeket figyelembe venné. Azt mondjuk, hogy, az ilyen rendszereknek – ahol a rendszer állapotának leírásakor csak a jelenlegi állapotot kell figyelembe venni (nem függ az előző állapottól) – *Markov-tulajdonságá*a *(Markov property)* van.

A következőkben formalizáljuk ezt a tulajdonságot. Jelenleg a t-edik időpillanatban vagyunk, és a $t+1$-edik időpillantba lépünk. Feltesszük, hogy véges számú a környezet állapotainak halmaza ($\mathcal{S}$), a jutalmak halmaza ($\mathcal{R}$) és az akciók halmaza ($\mathcal{A}$). Ez lehetővé teszi, hogy összegekkel és valószínűségekkel (Pr) dolgozzunk integrálok és valószínűségi sűrűségek helyett. Először nézzük az általános esetet:

$$Pr\{\, s_{t+1} = s',\, r_{t+1} = r \mid s_t,\, a_t,\, r_t,\, s_{t-1},\, a_{t-1},\, \ldots,\, r_1,\, s_0,\, a_0 \,\}\,, \tag{1.4}$$

bármely $s'(\in \mathcal{S})$, $r(\in \mathcal{R})$ és az összes múltbeli $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$ sorozat esetén. Ha a rendszer rendelkezik a Markov-tulajdonsággal, akkor a környezeti dinamika a következőképpen néz ki:

$$Pr\{s_{t+1} = s',\, r_{t+1} = r \mid s_t\,, a_t\,\}\,, \tag{1.5}$$

bármely s', r, s_t, a_t esetén.

Ekkor elegendő az egylépéses dinamika ahhoz, hogy megmondjuk a következő állapot valószínűségés és a várható jutalom értékét. Iterálva ezt az egyenletet megmutatható, hogy megkaphatjuk a jövőbeli állapotokat és várható jutalmakat a jelenlegi tudásunkból (jelenlegi állapotjel) és valószínűleg megadható a teljes eseménysor az aktuális időponttól. Azt a megerősítéses tanulási folyamatot, amely a Markov-tulajdonságot kielégíti, Markov döntési folyamatnak nevezik *(Markov Decision Process, MDP)*. Ha az állapot- és akciótér véges, akkor véges Markov döntési folyamatról *(finite Markov Decision Process, finite MDP)* beszélünk. A véges Markov döntési folyamat definiálva van az állapot és akció halmazzal és az egylépéses környezeti dinamikával. Az s' állapotba kerülés valószínűsége s állapot és a akció esetén:

$$\mathcal{P}^{a}_{ss'} = Pr\{\, s_{t+1} = s' \mid s_t = s,\, a_t = a \,\}. \tag{1.6}$$

Ezt a mennyiséget *átmeneti valószínűség*nek hívják. A várható jutalom s-ből s' állapotba jutáskor a akció mellett:

$$\mathcal{R}^a_{ss'} = E\{\, r_{t+1} = s' \mid s_t = s,\ a_t = a,\ s_{t+1} = s' \,\}. \tag{1.7}$$

Ezek a mennyiségek, $\mathcal{P}^a_{ss'}$ és $\mathcal{R}^a_{ss'}$ teljesen meghatározzák a véges Markov döntési folyamat dinamikáját.

Azokra a rendszerekre, problémákra alkalmazható bizonyítottan hatásosan a megerősítéses tanulási módszer, amelyekre teljesülnek a következők:

- Markov tulajdonság
- teljesen észlelt rendszer (az állapottér egészét ismerjük)
- véges állapottér

1.1.5. Értékelő függvény, Bellman egyenlet

Majdnem mindegyik megerősítéses tanulási algoritmus az *értékelő függvényen (value function)* alapul, amely nem más mint az állapotok (vagy állapot-akció pár) függvénye amely azt közelíti, hogy az adott állapot "milyen jó" (vagy az adott állapotban egy akció "milyen jó".) Az értékfüggvény jelentését pontosabban a későbbiekben fogalmazzuk meg.

A π politika definíciója (lásd 1.1.1 rész) a következő: bármely $s (\in \mathcal{S})$ állapot és bármely $a\,(\in \mathcal{A}(s))$ akció esetén $\pi(s,a)$ annak a valószínűségét adja meg, hogy s állapot esetén az a akciót választja az ügynök. Az s állapot értéke (jelölése: $V^\pi(s)$) π politika alatt nem más, mint a várható hozam nagysága az s állapotból kiindulva és a π politikát követve. A Markov döntési folyamat esetén formálisan is definiálhatjuk $V^\pi(s)$-t:

$$V^\pi(s) \;=\; E_\pi\{\, R_t \mid s_t = s \,\} \;=\; E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\}, \tag{1.8}$$

ahol E_π-vel jelöltük a várható értéket ha az ügynök a π politikát követi, t pedig az aktuális időpont. Érdemes megjegyezni, hogy a terminális állapot értéke – ha van ilyen – mindig zéró. A V^π függvényt a π politikához tartozó *állapotot értékelő függvény*nek nevezik. Hasonló módon definiálhatjuk az a akció értékét s állapot és π politika követése esetén. Jelölje $\mathcal{Q}^\pi(s,a)$ az a akció értékét s állapotban, feltéve, hogy a π poltikát követjük. Ekkor:

$$\mathcal{Q}^{\pi}(s,a) = E_{\pi}\left\{R_t \,|s_t = s,\, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \,|s_t = s,\, a_t = a\right\}, \tag{1.9}$$

$\mathcal{Q}^{\pi}$-t a π politikához tartozó *akciót értékelő függvény*nek nevezzük.

Az értékelő függvény alapvető tulajdonsága, hogy kielégít egy partikuláris rekurzív kapcsolatot. Minden π politikára és minden s állapotra, az s és a rákövetkező állapot értéke között a következő konzisztens kapcsolat áll fenn:

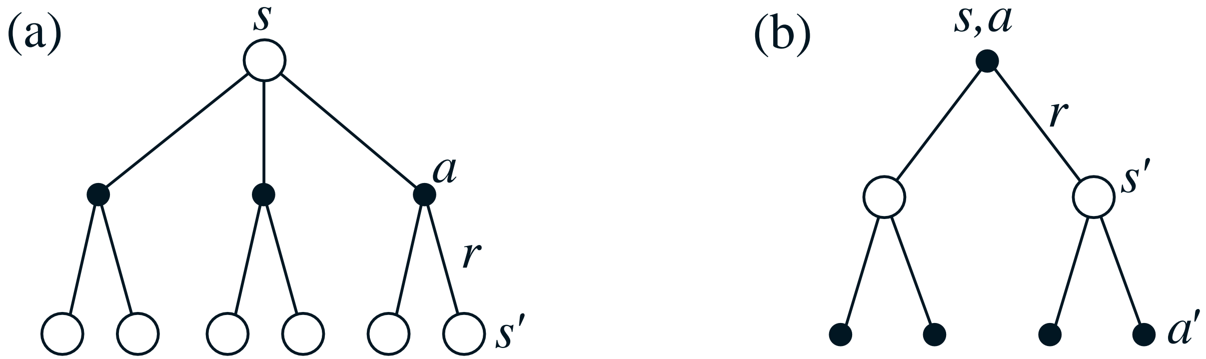
$$\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{ R_t \mid s_t = s \right\} \\
&= E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\} \\
&= E_{\pi}\left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s \right\} \\
&= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma\, E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_{t+1} = s' \right\} \right] \\
&= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma\, V^{\pi}(s') \right],
\end{aligned} \tag{1.10}$$

ahol a természetesen az $\mathcal{A}(s)$ halmaz eleme és a következő állapot pedig $\mathcal{S}$ (vagy $\mathcal{S}^+$, ha a folyamat epizódikus) halmazbeli elem. A (1.10) egyenlet a V^{π} *Bellman egyenlete*[1] *(Bellman equation)*. Megmutatható, hogy a Bellman egyenletnek egyedüli megoldása (azaz fixpontja) a V^{π} állapotot értékelő függvény. A 1.2 ábra a rekurzív kapcsolat szemléltetését szolgálja, ahol a jelölés a következő : üres körök az állapotok, teli körök pedig az állapot-akció párok.

1.1.6. Optimális politika, Optimális értékelő függvény, Bellman optimalitási egyenlet

A véges Markov döntési folyamatokra pontosan definiálni tudjuk az optimális politikát. Az értékelő függvény a következő parciális rendezést definálja a politikák között: egy π politika jobb vagy ugyanolyan jó mint egy π' politika, ha a π politikát követve minden egyes állapotban a várható hozam nagyobb vagy egyenlő, mint π' politika esetén. Ez jelölésekkel: $\pi \geq \pi'$, ha minden $s (\in \mathcal{S})$ állapotra $V^{\pi}(s) \geq V^{\pi'}(s)$. Mindig van legalább egy olyan politika, amely nagyobb, vagy egyenlő az összes többi politikánál. Ez az *optimális*

[1] Hasonló módon felírható $\mathcal{Q}^{\pi}$ Bellman egyenlete.



1.2. ábra. A V^π-t (a), és a $\mathcal{Q}^\pi$-t (b) meghatározó felösszegzési gráf.

politika (optimal policy). Az optimális politika, vagy politikák jelölése: π^*. Az optimális politikák segítségével megadható az *optimális állapotot értékelő függvény* (V^*) definíciója:

$$V^*(s) = \max_\pi V^\pi(s), \tag{1.11}$$

az összes $s \in \mathcal{S}$ állapotra.

Az optimális politikák segítségével az előzőkhez hasonlóan definiálható az *optimális akciót értékelő függvény*, $\mathcal{Q}^*$:

$$\mathcal{Q}^*(s,a) = \max_\pi \mathcal{Q}^\pi(s,a), \tag{1.12}$$

az összes $s \in \mathcal{S}$ állapotra, és az összes $a \in \mathcal{A}$ akcióra. $\mathcal{Q}^*$ felírható V^* segítségével is:

$$\mathcal{Q}^*(s,a) = E\left\{r_{t+1} + \gamma V^*(s_{t+1})\,|s_t = s, a_t = a\right\}. \tag{1.13}$$

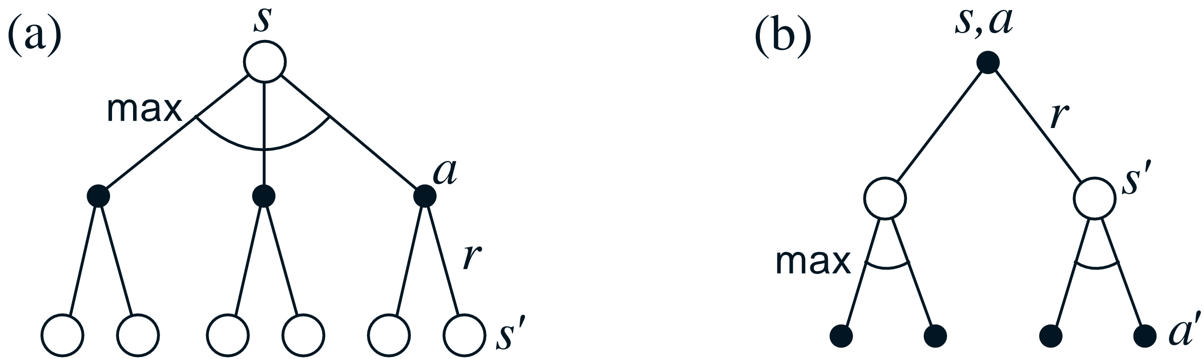
Mivel V^* értékelő függvény, ezért ki kell elégítenie a Bellman egyenlet által kirótt feltételt. Az optimális értékelő függvényhez tartozó Bellman egyenletet *Bellman optimalitási egyenlet*nek nevezik. A Bellman optimalitási egyenlet valójában az állapot értékét adja meg optimális politika esetén, amely egyenlő az adott állapotban a legjobb akció választásával kapott várható hozammal:

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} \mathcal{Q}^{\pi^*}(s,a) \\
&= \max_a E_{\pi^*} \{ R_t \,|s_t = s, a_t = a \} \\
&= \max_a E_{\pi^*} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s, a_t = a \right\} \\
&= \max_a E_{\pi^*} \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,|s_t = s, a_t = a \right\} \\
&= \max_a E_{\pi} \{ r_{t+1} + \gamma V^*(s') \,|s_t = s, a_t = a \} \qquad (1.14) \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s') \right]. \qquad (1.15)
\end{aligned}
$$

Az utolsó két egyenlet a V^*-hoz tartozó Bellman optimalitási egyenlet két formája. A Bellman optimalitási egyenlet $\mathcal{Q}^*$-ra :

$$
\begin{aligned}
\mathcal{Q}^*(s,a) &= E \left\{ r_{t+1} + \gamma \max_{a'} \mathcal{Q}^*(s_{t+1}, a') \,|s_t = s, a_t = a \right\} \qquad (1.16) \\
&= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} \mathcal{Q}^*(s', a') \right]. \qquad (1.17)
\end{aligned}
$$

A 1.3 ábra grafikusan szemlélteti a (1.15) alatt megadott Bellman optimalitási egyenletet.



1.3. ábra. A V^*-t (a), és az $\mathcal{Q}^*$-t (b) meghatározó felösszegzési gráf.

A véges Markov döntési folyamatoknál a Bellman optimalitási egyenletnek (1.15) a politikától független egyedülálló megoldása van. A Bellman optimalitási egyenlet valójában

minden egyes állapotra N állapot esetén egy N egyenletből álló N ismeretlenes egyenlet rendszer. Ha a rendszer dinamikája ismert ($\mathcal{P}^a_{ss'}$ és $\mathcal{R}^a_{ss'}$ adott) elvben megoldható lenne a V^*-hoz tartozó egyenletrendszer valamilyen nemlineáris egyenletrendszert megoldó módszer segítségével.[2]

V^* ismeretében az optimális politika könnyen meghatározható. Minden s állapotban a lehetséges akciók közül a legjobb a Bellman optimalitási egyenletből megadható. Az összes olyan politika, amelynél az optimális akciókhoz nem nulla valószínűség tartozik, az optimális. Az olyan politikát – ahol egylépéses keresés után a legnagyobb értékű akciót választjuk – mohónak nevezzük. A mohó politika hosszabb távra nézve választja ki a legjobb akciót (V^* definíciójából adódik) annak ellenére, hogy csak egy egylépéses keresés történik.

$\mathcal{Q}^*$ ismeretében a legjobb akciót szintén könnyű meghatározni – mivel rendelkezésünkre áll az összes állapot-akció érték ($\mathcal{Q}^*(s, a)$) –, bármely s állapotban a legjobb akció a táblázatból direkt módon kiolvasható (nem szükséges keresés). A $\mathcal{Q}^*$ megadja a várható optimális hozamot az állapot-akció értékpár lokális, közvetlen értékeként.

[2]Hasonló módon felírható illetve megoldható $\mathcal{Q}^*$-hoz tartozó egyenletrendszer.

1.2. Alapvető megoldási módszerek

1.2.1. Dinamikus programozás

A *dimikus programozás* (DP) elnevezés olyan algoritmusok gyüjteményére utal, amelyek az optimális politikák meghatározását szolgálják, és megadják a környezet – mint Markov döntési folyamat – tökéletes modelljét. A klasszikus DP algoritmus (melyet ismertetünk) korlátozottan használt, mivel csak közelítése a tökéletes modellnek, illetve nagy a számítási igénye.

Feltételezzük, hogy a környezet egy véges Markov döntési folyamat, az állapot- és az akció tér – $\mathcal{S}$ és minden s állapotra $\mathcal{A}(s)$ – véges, és a rendszer dinamikája az átmeneti valószínűséggel ($\mathcal{P}^a_{ss'}$), és a közvetlen várható jutalommal ($\mathcal{R}^a_{ss'}$) adott. A DP alkalmazható folytonos állapot- és akciótér esetén, de pontos megoldás csak néhány esetben lehetséges.

Az DP alapötlete –, mint általában a megerősítéses tanulás esetén – az, hogy az értékelő függvényt használjuk a jó politikát megkereső eljárás meghatározására. Azt már láttuk, hogy az optimális politika könnyen meghatározható a V^* vagy a $\mathcal{Q}^*$ optimális értékelő függvények segítségével, amelyek kielégíti a Bellman optimalitási egyenletet (1.15) (1.17). Látni fogjuk, hogy a DP algoritmusok megkaphatók a Bellman egyenletekből, ugyanis átírhatók – az értékelő függvény közelítését javító – felülírási szabályokká.

Politika kiértékelése

Először megadjuk, hogy hogyan számítható ki a V^π állapotot értékelő függvény π politika esetén. A DP nyelvezetben ezt az eljárást *politika kiértékelésnek (policy evalution)* nevezik. Az eljárást tekinthetjük egy predikciós problémának, ahol minden $s \in \mathcal{S}$ esetén,

$$
\begin{aligned}
V^\pi(s) &= E_\pi\left\{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \,|\, s_t = s \right\} \\
&= E_\pi\{ r_{t+1} + \gamma V^\pi(s_{t+1} \,|\, s_t = s\} && (1.18) \\
&= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \right], && (1.19)
\end{aligned}
$$

ahol $\pi(s,a)$ az a állapot választásának valószínűsége s állapotban π politikát követeve. A V^π létezése és egyedülállósága biztosított, ha $\gamma < 1$, vagy ha az összes állapotból véges sok lépés alatt terminális állapotba jutahatunk (természetesen π politikát követve).

Ha a környezeti dinamika adott, akkor (1.19) egyenlet az egy $|\mathcal{S}|$ egyenletből álló $|\mathcal{S}|$ ismeretlenes egyenletrendszernek felel meg. Az egyenletrendszert iteratív módszer segítségével oldjuk meg, amely az értékelő függvény közelítéseinek sorozatát (V_0, V_1, V_2, ...) adja meg. Az állapotok kezdeti értékbecslése (V_0) tetszőleges lehet (kivéve a terminális állapotokot, amelyek értéke mindig 0), és az értékelőfüggvény minden rákövetkező becslése megkapható a Bellman egyenletet (1.10) felülírási szabályként alkalmazva:

$$\begin{aligned} V_{k+1}(s) &= E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s\} \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi_k(s')] , \end{aligned} \qquad (1.20)$$

minden $s \in \mathcal{S}$ állapotra. $V_k = V^\pi$ fixpontja a felülírási szabálynak. Megmutatható, hogy a $\{V_k\}$ sorozat V^π-hez konvergál, ha $k \to \infty$, és a V^π létezése (az előző feltételek alapján) biztosított. Ezt az eljárást iteratív politika-kiértékelésnek nevezik.

```
Input: π policy to be evaluated
Initialize: V(s) = 0∀s ∈ S⁺
Repeat
   Δ ← 0
   For each s ∈ S
      v ← V(s)
      V(s) ← Σ_a π(s,a) Σ_s' P^a_ss' [R^a_ss' + γV^π(s')]
      Δ ← max(Δ, |v − V(s)|)
Until Δ < θ (small positive number)
Output: V ≈ V^π
```

1.1. táblázat. Iteratív poltika-kiértékelés.

Ahhoz, hogy megkapjuk az értékelő függvény következő közelítését (V_{k+1}), az iteratív politika-kiértékelés során minden egyes s állapotra ugyanazt az eljárást kell végrehajtanunk: s régi értékét kicseréljük az új értékre, amelyet az s utáni állapot régi értékéből, a várható pillanatnyi jutalomból, ill. az összes átmeneti valószínűségből számítunk ki. Ezt az eljárást *teljes felösszegzési gráfnak (full backup)* nevezik, ugyanis az állapot új értékének számításakor figyelembe vesszük az összes lehetséges rákövetkező állapotot. A 1.1 táblázat részletesen megadja az iteratív politika-kiértékelés algoritmusát.
A megállási feltétel a következő: a $\max_{s\in\mathcal{S}} |V_{k+1} - V_k|$ különbség kisebb egy pozitív, közel nulla nagyságú θ számnál.

Politika javítása

Tegyük fel, hogy meghatároztuk egy tetszőleges π politikához tartozó V^π értékelő függvényt. Tudni szeretnénk, hogy s állapotban a politika változtatásával – ami s állapotban determinisztikusan válsztott $a \neq \pi(s)$ akció választást jelent – javítunk vagy rontunk az eredeti π politikán. Azért, hogy erre a kérdésre válaszolni tudjunk, ki kell értékelni a megváltoztatott π politikát. Az s állapot értéke, ha megváltoztattuk a politikát $a(\neq \pi(s))$ akció válszatásával és ezután az eredeti π politikát követjük:

$$\begin{aligned} \mathcal{Q}^\pi(s,a) &= E_\pi \{ r_{t+1} + \gamma V_k(s_{t+1}) \,|\, s_t = s,\, a_t = a \} \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi_k(s') \right]. \end{aligned} \tag{1.21}$$

Ha az így kapott $\mathcal{Q}^\pi(s,a)$ nagyobb, mint $V^\pi(s)$, akkor egy jobb politikát kapunk, ha s állapotban a akciót válsztjuk, és utána a π politikát követjük, mintha végig a π politika szerint cselekednénk.

Ez az eredmény az általános *politika javításnak (policy improvement)* egy speciális esete volt. Legyen π és π' két determinisztikus politika, amelyekre minden $s \in \mathcal{S}$ állapotra a következők állnak fent:

$$\mathcal{Q}^\pi(s, \pi'(s)) \geq V^\pi(s). \tag{1.22}$$

Ekkor a π' politka legalább olyan jó vagy jobb, mint a π politika, és a várható hozamokra minden $s \in \mathcal{S}$ állapot esetén a következők teljesülnek:

$$V^{\pi'}(s) \geq V^\pi(s). \tag{1.23}$$

Ha a szigorúan nagyobb reláció fennáll néhány állapotra a (1.22) egyenletnél akkor legalább egy állapotra fennáll a szigorúan nagyobb reláció a (1.23) egyenletnél is. Ezt az eredményt különösen két politika esetén lehet jól alkalmazni, mint az előző bekezdésben látott π és a cserével módosított π' politika esetében, ahol a két politika közötti különbséget csak az s állapotban választott akció jelenti ($\pi'(s) = a \neq \pi(s)$). Magától érthetődik, hogy ha $\mathcal{Q}^\pi(s,a) > V^\pi(s)$, akkor a megváltoztatott π' politika jobb, mint az eredeti π politika.

Ez könnyen bizonyítható kiindulva a (1.22) egyenletből:

$$
\begin{aligned}
V^{\pi}(s) &\leq \mathcal{Q}^{\pi}(s,\pi'(s)) \\
&= E_{\pi'}\{\, r_{t+1} + \gamma\, V^{\pi}(s_{t+1}) \mid s_t = s\} \\
&\leq E_{\pi'}\{\, r_{t+1} + \gamma\, \mathcal{Q}^{\pi}(s_{t+1},\pi'(s_{t+1})) \mid s_t = s\} \\
&= E_{\pi'}\{\, r_{t+1} + \gamma\, E_{\pi'}\{\, r_{t+2} + \gamma\, V^{\pi}(s_{t+2})\} \mid s_t = s\} \\
&= E_{\pi'}\Big\{\, r_{t+1} + r_{t+2} + \gamma^2\, V^{\pi}(s_{t+2}) \mid s_t = s\Big\} \\
&\leq E_{\pi'}\Big\{\, r_{t+1} + r_{t+2} + \gamma^2\, r_{t+3} + \gamma^3 V^{\pi}(s_{t+3}) \mid s_t = s\Big\} \\
&\;\vdots \\
&\leq E_{\pi'}\Big\{\, r_{t+1} + r_{t+2} + \gamma^2\, r_{t+3} + \gamma^3\, r_{t+4} + \ldots \mid s_t = s\Big\} \\
&= V^{\pi'}(s).
\end{aligned}
$$

Ezen eljárás természetes kiterjesztése az, hogy minden állapotban az aktuális politikának megfelelő akcióválasztás helyett, az összes lehetséges akció közül azt válsztjuk, ahol a $Q^{\pi}(s,a)$ értek a lehető legnagyobb. Tulajdonképpen megadunk egy új mohó politikát, π'-t, amely a következő módon formalizálható:

$$
\begin{aligned}
\pi'(s) &= \arg\max_a \mathcal{Q}^{\pi}(s,a) \\
&= \arg\max_a E\{\, r_{t+1} + \gamma\, V^{\pi}(s_{t+1}) \mid s_t = s,\ a_t = a\,\} \qquad (1.24) \\
&= \arg\max_a \sum_{s'} \mathcal{P}^{a}_{ss'}\,[\,\mathcal{R}^{a}_{ss'} + \gamma\, V^{\pi}_{k}(s')\,]\ ,
\end{aligned}
$$

ahol $\arg\max_a$ jelöli a $\mathcal{Q}^{\pi}(s,a)$ kifejezés maximális értékét. A mohó politika megadja a – rövid távon – legjobbnak igérkező akciót V^{π}-re nézve. Azt az eljárást, amely az eredeti politikához tartozó értékelő függvényt mohó módon alkalmazva megadja az (eredetinél jobb) új politikát, *politika javításnak (policy improvement)* nevezik.

π',az új mohó politikánk, legalább olyan jó (de nem rosszabb), mint a régi π politikánk. Ha $V^{\pi} = V^{\pi'}$, akkor minden $s \in \mathcal{S}$ állapotra a (1.10) definíció alapján:

$$
\begin{aligned}
V^{\pi'}(s) &= \max_a E\Big\{\, r_{t+1} + \gamma\, V^{\pi'}(s_{t+1}) \mid s_t = s,\ a_t = a\,\Big\} \\
&= \max_a \sum_{s'} \mathcal{P}^{a}_{ss'}\Big[\,\mathcal{R}^{a}_{ss'} + \gamma\, V^{\pi'}(s')\Big].
\end{aligned}
$$

Ez az egyenlet megegyezik a Bellman optimailtási egyenlettel (1.15), ezért $V^{\pi'}$ az optimális értékelő függvény (V^*), és π és π' is optimális politika.

Sztohasztikus politika esetén a politika javítása hasonló módon történik, a következő egyenlet segítségével:

$$\mathcal{Q}^{\pi}(s,\pi'(s)) = \sum_{a} \pi'(s,a)\,\mathcal{Q}^{\pi}(s,a). \tag{1.25}$$

A bizonyítást nem részletezzük.

Politika iterálása

Adott egy π politikánk, amelyet a V^{π} segítségével javítunk, ekkor kapjuk a π' politikát; utána meghatározzuk $V^{\pi'}$ értékelő függvényt amely segítségével megadunk egy jobb π'' politikát. Így a politika kiértékelés és a javítás a következő sorozatot adja meg:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*,$$

ahol $\xrightarrow{E}$ jelöli a politika kiértékelést, $\xrightarrow{I}$ pedig a politika javítást. Az optimális politika megtalálásának ezen útját *politika-iterácónak (policy iteration)* nevezik. A részletes algoritmust az 1.2 táblázat mutatja.

Érték iteráció

A poltitika-iteráció hátránya a következő: minden egyes lépésben ki kell értékelni a politikát, a politika kiértékelésébe, amely megnyújtja a számítási időt mivel az állapotteret többszörösen át kell vizsgálni. Ha a politika kiértékelés iteratív módon történik, várnunk kell addig, míg az értékelő függvény becslések sorozata pontosan konvergál. Felmerülhet az a kérdés, hogy ki kell-e várnunk a pontos konvergenciát, vagy hamarabb befejezhetjük a politika kiértékelést.

A politika kiértékelés lépésszáma lecsökkenthető anélkül, hogy a politika iteráció konvergenciáját elveszítenénk. Azt a politika kiértéklést (algoritmust), ahol csak egyszer nézzük végig az állapotok halmazát, és utána megállunk *érték iterációnak (value iteration)* nevezzük. Ez egy egyszerű eljárás, amely a politika javítást és a lerövidített politika kiértékelést kombinálja:

1. Initialize:
   $V(s) \in \Re$ és $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$

2. Policy evaluation
   Repeat
     $\Delta \leftarrow 0$
     For each $s \in \mathcal{S}$
       $v \leftarrow V(s)$
       $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   Until $\Delta < \theta$ (small positive number)

3. Policy improvement
   policy-is-stabil $\leftarrow$ true
   For each $s \in \mathcal{S}$
     $b \leftarrow \pi(s)$
     $\pi(s) \leftarrow \max_a \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$
     If $b \neq \pi(s)$, then *policy-is-stabil* $\leftarrow$ false
   If *policy-is-stabil* then stop; else go to 2.

1.2. táblázat. Politika iterálása

$$\begin{aligned} V_{k+1}(s) &= \max_a E\{ r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s,\, a_t = a \} \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')], \end{aligned} \tag{1.26}$$

minden $s \in \mathcal{S}$. Megmutatható, hogy tetszőleges V_0 esetén a $\{ V_k \}$ sorozat a V^*-hoz konvergál V^* létezéséhez szükséges feltételek mellett.

Mint a politika iteráció az érték iteráció is formálisan végtelen lépésszám után konvergál pontosan V^*-hoz, az optimális értékelő függvényhez. Gyakorlatban azonban az algoritmusunk megáll, ha az értékelő függvény változása kicsi. A 1.3 ábra mutatja a teljes érték iteráció algoritmust, az előző megállási feltétellel. Az algoritmus – diszkontált véges Markov folyamatok esetén – az optimális politikát adja meg.

Initialize: V is arbitrarily, except $s \in \mathcal{S}^+$, where $V(s) = 0$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
Until $\Delta < \theta$ (small positive number)

Output: deterministic π policy
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$

1.3. táblázat. Érték iteráció.

Aszinkron dinamikus programozás

A DP módszererek legnagyobb hátránya az, hogy hozzá tartozó algoritmusok a Markov döntési folyamat teljes állapotterén dolgozik – az állapot-halmaz minden elemét végignézi –, így nagy állapottér esetén valószínűleg költséges lesz a futási idő tekintetében.

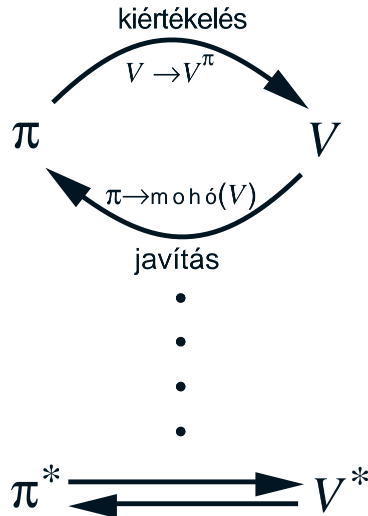
Az *aszinkron (asyncronous)* DP algoritmusok nem tartalmaznak szisztematikus állapothalmaz áttekintést, tulajdonképpen "helyben" iteráló algoritmusok. Egy aszinkron algoritmus tetszőleges sorrendben használja fel az elérhető állapotok értékét, amely nagy rugalmasságot biztosít. A folytonosság biztosítja a konvergenciát, azaz az algoritmus az optimális politika felé konvergál.

Általánosított politika iteráció

A politika iteráció két egyszerre zajló, egymással kapcsolatban lévő folyamatból épül fel: az egyik az adott politikához megadja az értékelő függvényt (politika kiértékelés), a másik, amely mohó módon új politikát hoz létre az értékelő függvény alapján (politika javítás). A politika iterációban ez a két folyamat alternál, az egyik akkor kezdődik, amikor a másik befejeződik, holott ez nem szükséges. A két folyamat egyszerre is végbemehet (mindkettő felülírhatja az értékelő függvényt), és ebben az esetben is az algoritmus az optimális politika, illetve az optimális értékelő függvény felé konvergál.

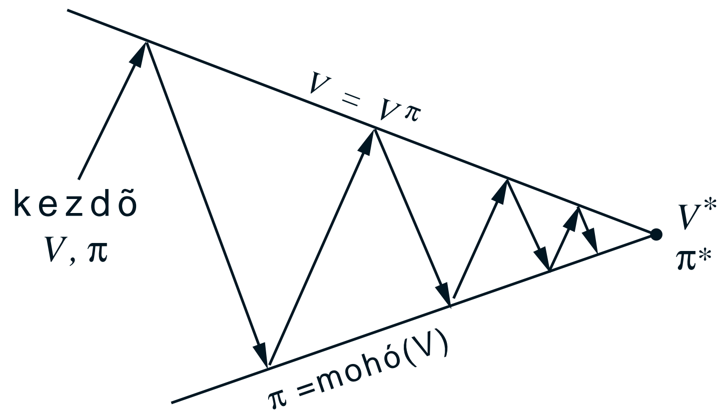
Ezt a módszert *általánosított politika iterációnak (generalized policy iteration, GPI)* nevezik. Az általános ötlet: a politika kiértékelés és javítás folyamata összekapcsolt,

független a folyamatok részleteitől. Majdnem minden megerősítéses tanulási módszer leírható GPI-ként. a 1.4 ábra szemlélteti a GPI-t.



1.4. ábra. Általánosított politika iteráció.

Ennek a módszernek egy másik szemléltető ábrája a 1.5 ábra, ahol a két folyamatot két vonal ábrázolja kétdimenziós térben.



1.5. ábra. Politika kiértékelés és javítás a GPI-nél.

1.2.2. Időbeli-differencia módszere

Ha meg kellene határoznunk, hogy milyen újítást hozott a megerősítéses tanulás az optimalizációs algoritmusok területén, kétségkívül az *időbeli-differenciák (temporal difference, TD)* módszerét jelölnénk meg. A TD módszer direkt módon – a környezeti dinamika ismerete nélkül – tanul a tapasztalatokból. A TD módszer felülírási becslése az előző becsléseken alapul, anélkül, hogy megvárná a végső következtetést. Az eddig megismert módszer, a DP és a TD között a fő különbség a politika kiértékelésében, vagy másképpen a jóslási folyamatban van, amely megadja a V^π értékelő függvényt. A szabályzás probléma megoldására (az optimálais politika megadása) a DP, és a TD egyaránt a GPI variációit használja fel.

TD jóslás

A TD módszer felhasználja a tapasztalatait, annak érdekében hogy megoldja a jóslási problémát. A π politika követése során tapasztalatokat szerez, amelyek segítségével V-t (π-hez tartozó értékelő függvényt) felülírja. Ha a t-edik időpillanatban egy nemterminális s_t állapotban van a rendszer , akkor $V(s_t)$ becslése a következő eseményeken alapul. A TD módszer a következő lépés során már megmondja, hogy hogyan kell megváltoztatni a $V(s_t)$ becslést. A $t+1$-edik lépésben közvetlenül formalizálja a célt, azaz végrehajtja V felülírását felhasználva az észlelt r_{t+1} jutalmat és a $V(s_{t+1})$ korábbi becslést. A legegyszerűbb TD módszer, ismertebb nevén TD(0) a következő:

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]. \tag{1.27}$$

Mivel a TD módszer egy létező becslésen alapul, azt szoktuk mondani, hogy "önmaga becslésein alapuló" *(bootstrapping=cipőkanál)* módszer.

A DP módszer is egy becslés, de nem a várható érték miatt (V), amely a teljes környezeti modellt feltételezi, hanem a $V^\pi(s_{t+1})$ miatt, amely nem ismert, és helyette az aktuális közelítést $V_t(s_{t+1})$-et használják. A TD módszer mindkét szempontból becslés: mintát vesz a várható értékből és a jelenlegi V_t közelítést használja a V^π helyett. Az 1.4 táblázat a TD módszert szemlélteti, a 1.6 ábra pedig a TD algoritmust

Szoktak úgy utalni a TD módszerre, mint egy "példa felösszegzési gráfra" *(sample backup)*, mivel magában foglal egy tetszőleges rákövetkező állapotra (állapot-akció párra) vonatkoztatott felösszegzési gráfot, és a rákövetkező állapot értékét illetve a jutalmat használja fel, hogy visszamenőleg meghatározza az eredeti állapot (állapot-akció pár) értékét. TD példa felösszegzési gráfja különbözik a DP teljes felösszegzési gráfjától mivel

Input: π policy to be evaluated
Initialize: $V(s)$ arbitrarily
Repeat for each episode
  initialize s
  Repeat for each step of episode
    choose a in s using π
    take a, r , s'
    $V(s_t) \leftarrow V(s_t) + \alpha\,[r + \gamma V(s_{t+1} - V(s_t))]$
    $s \leftarrow s'$
  until s is terminal

1.4. táblázat. TD(0) algoritmus V^π becslésére.

1.6. ábra. A TD(0)-hoz tartozó felösszegzési gráf.

az állapot új értéke csak rákövetkező állapoton alapul, nem a teljes lehetséges állapotok halmazán.

A TD jóslás előnyei

A TD módszer a saját becsléseit más becslésekből származtatja. A találgatásait tanulja a találgatásokból, azaz "önfelhúzó" (bootstrap). A TD módszer előnye a DP-vel szemben az, hogy nem szükséges a környezeti modell dinamikájának ismerete: a közvetlen jutalom ($\mathcal{R}^a_{ss'}$) és a átmeneti valószínűség ($\mathcal{P}^a_{ss'}$) sem kell a becslések javításához.

Másik előnye a TD módszernek az, hogy már a következő lépés alatt javítja az V értékelő függvény becslését, így nem kell kivárni egy hosszú epizód végét, nem is beszélve a folytonos folyamatokról.

TD(0) optimalitása

Tegyük fel, hogy adott véges számú tapasztalat, 10 epizód vagy 100 lépés. Ebben az esetben az átlagos közelítési módszer addig ismételgeti a tapasztalatokat, amíg a módszer nem konvergál a válaszhoz. Adott V, az értékelő függvény közelítése, amelyet javítani

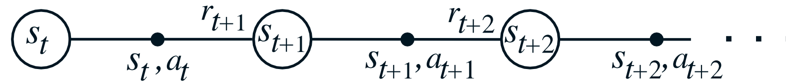
szeretnénk. A (1.27) egyenlet alapján minden időpillanatban meghatározásra kerül a változtatás mértéke, de felülírása csak egyszer (az epizód végén), a változások összegével történik. Ezt a módszert *kötegelt felülírásnak (batch updating)* nevezik.

Kötegelt felülírási folyamat esetén a TD(0) módszer az egyedüli válaszhoz konvergál, a folyamat természetesen függ a lépésköz (α) paramétertől, amelyet a konvergencia érdekében elegendően kis számnak kell választani.

Sarsa: aktív politizálási TD szabályozás

Nézzük meg, hogy TD jóslási módszer hogyan használatos a kontroll probléma esetén. Követjük a minta általános politika iteráció (GPI) módszerét, de a jóslási vagy politika kiértékelési folyamatot a TD módszer alapján végezzük. A közelítéseknek két fő osztálya van: az aktív politizálás és a inaktív politizálás. Nézzük az aktív politizálási módszert részletesen.

Az első lépés az, hogy az akció értékelő függvényt tanuljuk meg, nem az állapotot értékelő függvényt, amely hasonló módon történik, mint V^π közelítése.



1.7. ábra. Az állapotok és az állapot-akció párok alternáló sora.

Most formalizálni fogjuk állapot-akció értékpárból állapot akció értékpárba való átmenet során hogyan tanulja az állapot-akció értékpár értékét. Tételek biztosítják az állapotot értékelő függvény konvergenciáját TD(0) módszer esetén, alkalmazzuk ugyanezt az akció értékekre:

$$\mathcal{Q}(s_t, a_t) \leftarrow \mathcal{Q}(s_t, a_t) + \alpha \left[r_{t+1} + \gamma\, \mathcal{Q}(s_{t+1}, a_{t+1}) \,-\, \mathcal{Q}(s_t, a_t) \right]. \tag{1.28}$$

A felülírás minden egyes nemterminális s_t állapot esetén végrehajtódik. Ha s_{t+1} terminális állapot, akkor $\mathcal{Q}(s_{t+1}, a_{t+1})$ érték nulla. A felülírási szabály az $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ sorozat mindegyik elemét felhasználja, hogy megadja az állapot-akció értékpárból a következőbe történő átmenetet. Ez az ötelemű sorozat kiadja a Sarsa[3] elnevezést.

[3]state-action-reward-state-action

Mint a legtöbb aktív politizálási módszernél, folyamatosan közelítjük Q^{π} értékét, és vele egyidőben π – Q^{π}-re nézve mohó módon – változik. Az általános Sarsa szabályozó algoritmust 11. ábra mutatja be. A Sarsa algoritmus 1 valószínűséggel konvergál az optimális politikához és az állapot-akció értékelő függvényhez, amint az összes állapot-akció értékpáron végtelenszer átfutottunk.A politika limeszben a mohó politikához (amely például lehet ϵ-mohó politika, ahol $\epsilon = 1/t$) konvergál.

```
Initialize: Q(s, a) arbitrarily
For each episode)
    initialize s
    choose a in s ϵ-greedy
    Repeat (for each step of the episode)
            execute a take r, s'
            choose a ϵ-greedy in s'
            Q(s, a) ← Q(s, a) + α [r + γ Q(s', a') − Q(s, a))]
            s ← s'; a ← a';
    until s is terminal
```

1.5. táblázat. Sarsa: aktív politizálási TD szabályozás algoritmusa.

1.2.3. Az emlékeztető nyomok módszere

Majdnem minden TD módszer, mint például a Sarsa, konbinálható az *emlékeztető nyomok (eligibility trace)* módszerével, amely általánosabb és hatásosabban tanuló eljárást eredményez. Kétféle módon tekinthetünk az emlékeztető nyomok módszerre:

Az elméletibb jellegű értelmezés, amelyet *előre tekintésnek (forward view)* neveznek, azt mondja, hogy ez a módszer híd a TD és a *Monte Carlo módszer (MC)*[4] között.

A másik szemlélet sokkal gyakorlatiasabb, ez az úgynevezett *visszafele tekintés (backward view)* módszere. Ebből a szemszögből az emlékeztetető nyomok nem mások, mint a bekövetkezett események (amely állapotot, vagy akciót jelenthet) átmeneti feljegyzése. A nyom megjegyzi a memória paramétereket, amelyek össze vannak kapcsolva az eseményel, mint emlékezés az átmenő tanulási változásokon. Mindent összevetve az emlékeztető nyomok módszere segít áthidalni azt a rést, amely az események és a tanulási információk között van.

[4] A Monte Carlo módszer az értéklő függvény és az optimális politika meghatározására szolgáló eljárás. Lásd [1] 5. fejezet. Az ϵ-mohó politika ϵ valószínűséggel sztochasztikus akcióválasztást valósít meg és 1-ϵ valószínűséggel mohó politikát folytat.

n-lépéses TD jóslás

Mi a különbség az MC és a TD között? A MC módszer esetén a felösszegzési gráf értéke a teljes észlelt jutalom-sorozaton alapul, kezdve az aktuális állapottól a befejező, epizódot lezáró terminális állapotig. Az egyszerű TD módszer csak a következő jutalmon és a következő állapot becsült értékén alapul, amely helyettesíti a várható jutalmakat. A köztes módszerek azok, amelyek n-lépésben képzik a felösszegzési gráf értékét. Ezek szintén TD módszerek, mivel a korábbi becsléseket változtatja a későbbi becslések különbségéből n-lépéssel később. Ezt a módszert *n-lépéses TD jóslásnak* nevezik és TD(n)-nel jelölik.

TD (1–lépéses) 2–lépéses 3–lépéses *n*–lépéses Monte Carlo

1.8. ábra. Az n-lépéses TD jóslás felösszegzési gráfja.

Formálisan: s_t az aktuális állapot, amelyre a felülírási értéket szeretnénk meghatározni, $s_t, r_{t+1}, s_{t+1}, r_{t+2}, \ldots, s_T, r_T$ pedig az állapot-jutalom sorozat. A MC módszer esetében ekkor a $V_t(s_t)$, vagy a $V^\pi(s_t)$ felülírás a teljes várható hozamon alapul:

$$R_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + \ldots + \gamma^{T-t-1}\, r_T\,,$$

ahol T a periódust lezáró lépés. Ezt a mennyiséget nevezzük a felösszegzési gráf *célpontjának.* Az egy-lépéses TD módszer célja az első jutalom plusz a diszkontált, becsült értéke a következő állapotnak:

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}).$$

Ennek van értelme, mert a $\gamma V_t(s_{t+1})$ érték helyettesíti a maradék $\gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T$ tagokat. Más szempontból ez lehetőséget ad hasonló módon megfogalmazni a kétlépéses TD módszer célját, amely:

$$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2}),$$

ahol $\gamma^2 V_t(s_{t+2})$ helyettesíti a $\gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots + \gamma^{T-t-1} r_T$ formulát. Általánosan megfogalmazva:

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}).$$

Ezt a mennyiséget "korrigált n-lépéses levágott hozamnak" nevezik, mert a hozamot n-lépés után "levágjuk", és a levágott részt megközelítően korrigáljuk az n-edik állapot becsült értékével. Az elnevezés egy kicsit hosszú, ezért inkább $R_t^{(n)}$-t egyszerűen *n-lépéses hozamnak (n-step return)* nevezik. Természetesen, ha az epizód n lépésnél hamarabb fejeződik be, a levágás az epizód végétől kezdődik, és a teljes hozamot eredményezi. Más szóval: ha $T - t < n$, akkor $R_t^{(n)} = R_t^{(T-t)} = R_t$.

Az n-lépésben képzett felösszegzési gráf az n-lépéses hozamhoz tartozó felösszegzési gráffal van definiálva. Az állapot-érték esetben a $V_t(s_t)$ (becsült értéke $V^\pi(s_t)$-nek a t-edik időpillanatban) módosítása a következő módon definiált:

$$\Delta V_t(s_t) = \alpha \left[R_t^{(n)} - V_t(s_t) \right],$$

ahol α pozitív lépésköz paraméter. Természetesen $s \neq s_t$ állapotok becsült értékének a növelése $\Delta V_t(s) = 0$. Az n-lépéses módszert ezzel az egyenlettel definiáljuk a közvetlen felülírási szabály helyett. Ennek az az oka, hogy két különböző formáját különböztetjük meg a felülírásnak. Az egyik forma az úgynevezett *on-line felülírás (on-line updating)*, a felülírás folyamatosan történik, mikor kiszámítjuk az új változást. Ekkor $V_{t+1}(s) =$

$V_t(s) + \Delta V_t(s)$ minden $s \in \mathcal{S}$ esetén. A másik formát *off-line felülírásnak (off-line updating)* nevezik, ahol a felülírás az epizód végén történik az addig összegyűjtött változások összegével. Ekkor $V_t(s)$ minden s állapot esetén konstans az epizód végéig. Ha s állapot értéke $V(s)$, akkor az epizód végén az új érték $V(s) + \sum_{t=0}^{T-1} \Delta V_t(s)$ lesz.

Az n-lépéses várható hozam V segítségével történő meghatározása biztosítja a V^π V-nél jobb közelítését a legrosszabb esetben is. Ez azt jelenti, hogy a legnagyobb hiba az új közelítésnél kisebb, vagy egyenlő, mint γ^n-szer a legnagyobb hiba V értékelő függvény esetén:

$$\max_s \left| E_\pi \left\{ R_t^{(n)} \mid s_t = s \right\} - V^\pi(s) \right| \;\leq\; \gamma^n \max_s |V(s) - V^\pi(s)| \,. \tag{1.29}$$

Ezt az n-lépéses hozam *hiba csökkentési tulajdonságnak* nevezik. A tulajdonság következtében formálisan megmutatható, hogy az on-line és az off-line TD jóslás helyes módon működik a közelítesi feltételek mellett.

A TD(λ), mint előretekintő módszer

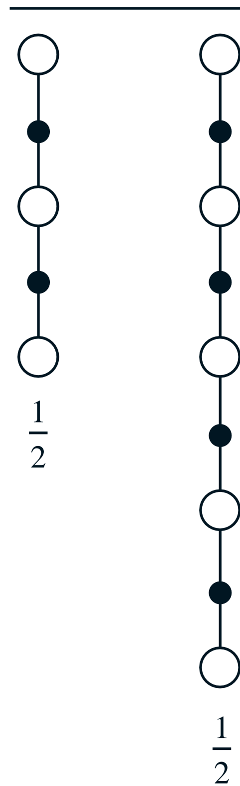
A felösszegzési gráf értékének meghatározása nemcsak n-lépés hozammal, hanem n-lépéses hozamok átlagának segítségével is történhet. Vegyük a következő példát: két- és négylépéses hozamok átlagával határozzuk meg a felösszegzési gráf értékeket. A hasonló módon előállított értékeket *komplex felösszegzési gráf értéknek* nevezzük (lásd 1.9 ábra).

A TD(λ) algoritmus sajátos formája az n-lépéses felülírás átlagának. Ez az átlag tartalmazza az összes n-lépéses felülírást, mindegyik λ^{n-1}-gyel arányos módon súlyozott (1.11 ábra). A normalizáló faktor $1 - \lambda$ biztosítja, hogy a súlyok összege 1 legyen. Az kapott hozamot *λ-hozamnak (λ return)* nevezik, amely a következő formában definiálunk:

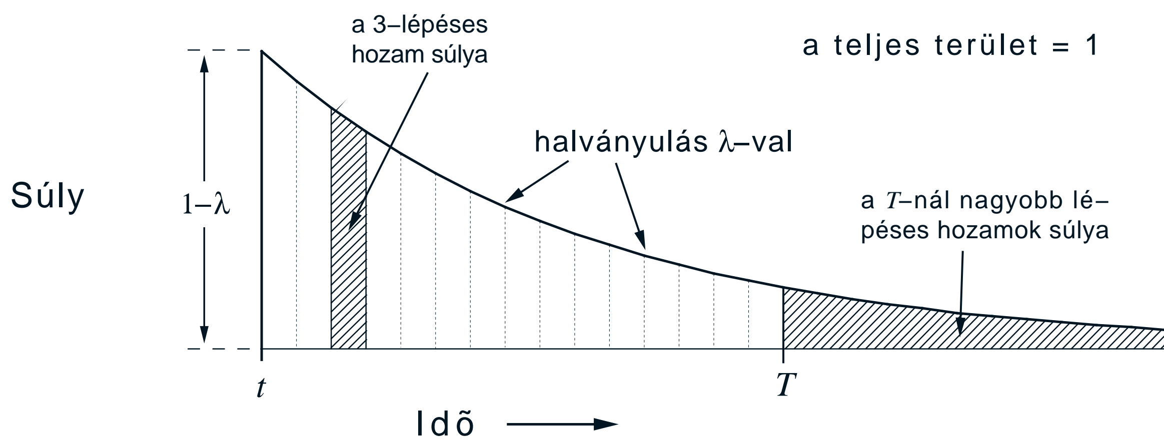
$$R_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}.$$

A súlyok minden egyes hozzáadott lépés során λ-val felejtődnek el. Akkor, amikor a rendszer terminális állapotba kerül (epizód vége), akkor az összes n-lépes hozam egyenlő lesz R_t-vel. Ezt felhasználva a λ-hozam definíció átfogalmazható:

$$R_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \gamma^{T-t-1} R_t. \tag{1.30}$$

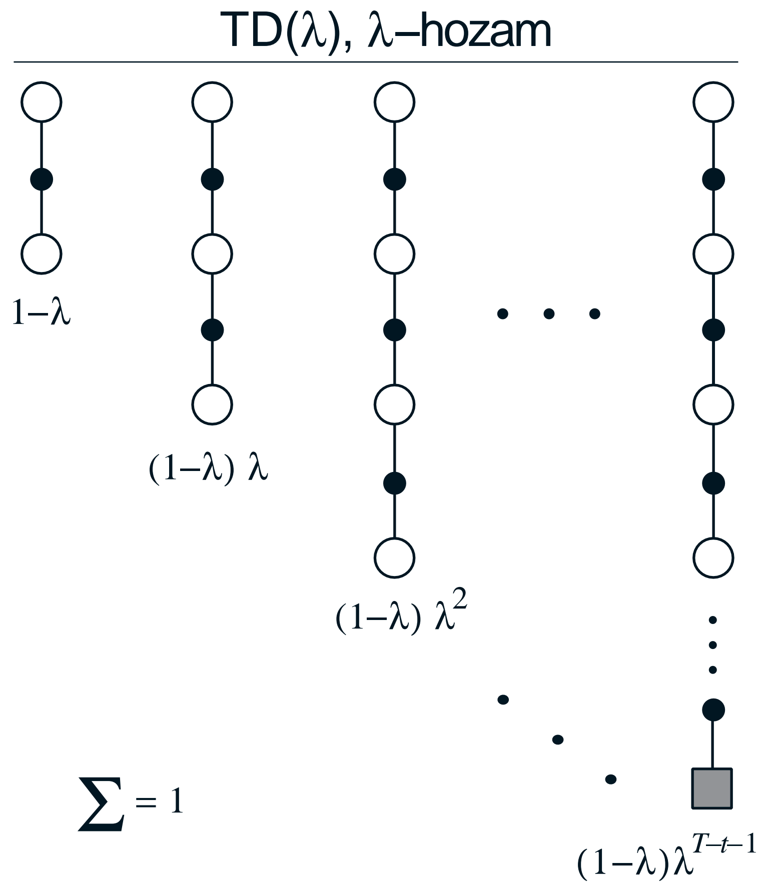


1.9. ábra. Komplex felösszegzési gráf



1.10. ábra. λ-súlyozás

Megvizsgálhatjuk a λ paraméter értékét: ha a $\lambda = 1$, akkor a λ hozam megegyezi az MC hozammal, ha $\lambda = 0$, akkor a λ-hozam megegyezik a az egylépéses TD, azaz a TD(0) módszernél definiált hozammal.

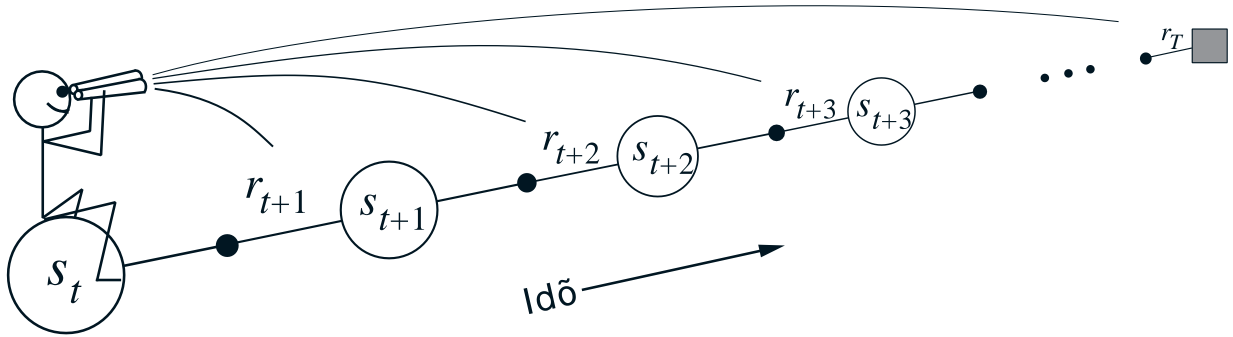


1.11. ábra. A TD(λ) módszer felösszegzési gráfja

Definiáljuk úgy a λ-*hozam algoritmust*, mint egy olyan algoritmust, amely a λ-hozam segítségével meghatározza a felülírási gráf értékét. Minden egyes t lépésben a $\Delta V_t(s_t)$ változtatás mértékét megadja az algoritmus a következő formában:

$$\Delta V_t(s_t) = \alpha \left[R_t^\lambda - V_t(s_t) \right]. \quad (1.31)$$

($\Delta V(s_t) = 0$ minden $s \neq \mathcal{S}$ állapotra.) A növekedési egyenlet egyaránt alkalmazható on-line és off-line esetben. A TD(λ) módszer ezt a fajta közelítését hívják elméleti, vagy előretekintő szemléletű tanulási algoritmusnak (lásd 1.12 ábra).



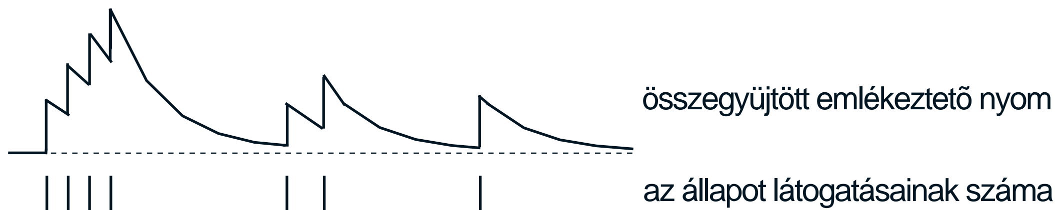
1.12. ábra. A TD(λ), mint elméleti vagy előretekintő módszer

A TD(λ), mint visszatekintő módszer

A gyakorlatiasabb, vagy viszatekintő módszer egyszerűbb felépítésű, és sokkal könnyebben lehet algoritmizálni. Ebben a felépítésben egy memória változót, az úgynevezett *emlékeztető nyomot (eligibility trace)* használunk, amely az összes állapottal össze van kapcsolva. s állapot és t időpillanat esetén az emlékeztető nyomot – $e_t(s)$ ($\in \mathcal{R}^+$)-vel jelöljük – a következőképpen definiáljuk:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) \text{ ha } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 \text{ ha } s = s_t \end{cases} \tag{1.32}$$

minden nemterminális s állapotra, ahol γ a diszkontálási hányados, λ definíciója pedig az előző fejezetben megtalálható. Ezentúl λ-ra, mint nyomfelejtési paraméterre hivatkozunk. Ezt a fajta emlékeztető nyomot összegyűjtési nyomnak nevezik, mert egy állapot minden egyes elérésével a hozzá tartozó nyom felerősödik, és fokozatosan eltűnik, ha az adott állapotot nem érjük el többször.



1.13. ábra. Az összegyűjtési nyom

A nyom megjegyzi, hogy a melyek azok az állapotok, amelyeket mostanában elért, a $\gamma\lambda$

szorzat segítségével. A megerősíteni kívánt eseményeket pillanatról pillanatra összekapcsolja a TD módszer hibájával. Az állapot-érték jóslás esetén a TD módszer hibája:

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)\,. \tag{1.33}$$

A globális TD hibajel segítségével megadhatjuk az értékelő függvény felülírási egyenletét:

$$\Delta V_t(s) = \alpha \delta_t e_t(s)\,, \quad \forall s \in \mathcal{S} \tag{1.34}$$

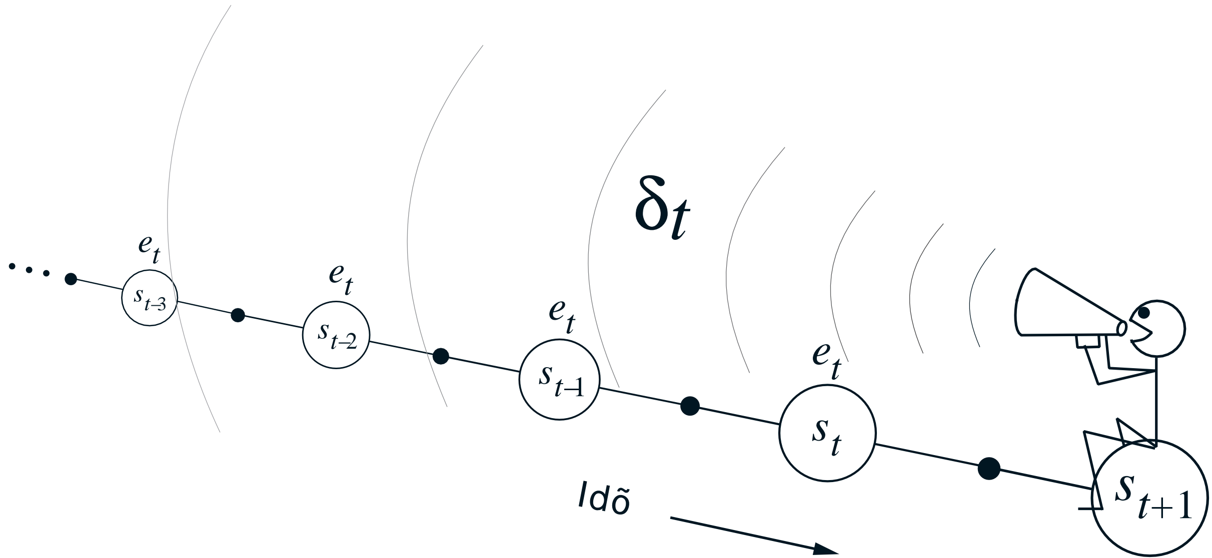
A $\Delta V_t(s)$ értékkel felülírhatjuk a becslésünket minden lépésben (on-line algoritmus) vagy csak az epizód végén a változások összegével (off-line algoritmus). Másrészt az előbbi (1.32-1.34) egyenletek megadják a TD(λ) algoritmus definícióját. A 1.6 táblázat tartalmazza a teljes on-line TD(λ) algoritmust. Minden egyes időpillanatban megnézzük az aktuális TD hibát és megállapítjuk a felülírási értéket az összes olyan állapotra, amelynek nyoma van az adott pillanatban. Ezt szemléletesen a 1.14 ábra mutatja be.

Initialize $V(s)$ arbitrarily and $e(s) = 0 \forall s \in \mathcal{S}$
Repeat for each step in episode
    initialize s
    choose a in s using π
    take r, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For each s
        $V(s) \leftarrow V(s) + \alpha \delta e(s)$
        $e(s) \leftarrow \gamma \lambda e(s)$
    $s \leftarrow s'$
until s is terminal

1.6. táblázat. Az on-line TD(λ) algoritmus.

Vizsgáljuk meg a paraméterek lehetséges értékénél hogyan viselkedik a TD(λ) módszer. Ha $\lambda = 0$, akkor a TD(λ) (1.34) felülírási szabály a TD(0) módszer felülírási szabályára (1.27) egyszerűsödik. λ nagyobb értékeire, de még $\lambda < 1$ több megelőző állapotra emlékezünk, de a távolabbi állapotoknak az aktuális állapotra gyakorolt hatása kisebb lesz, mivel a hozzájuk tartozó emlékezési nyom kisebb. Azt mondjuk, a korábbi állapotok kisebb

súllyal szerepelnek a TD hibában. Ha $\lambda = 1$, akkor a korábbi állapotokhoz tartozó súly γ szorosával csökken lépésenként. Ekkor az algoritmus TD(1) módszerként ismert. Ha még $\gamma = 1$, akkor az emlékeztető nyom úgy viselkedik, mint a nemdiszkontált, epizódikus MC módszer.



1.14. ábra. A TD(λ) módszer mehanikus, vagy visszatekintő szemlélete.

Az előre- és a visszatekintő módszerek ekvivalenciája

Ebben a részben megmutatjuk, hogy az off-line TD(λ)[5] módszer ugyan azt a súlyfelülírást valósítja meg, mint az off-line λ-hozam algoritmus, tehát a visszatekintő és az előretekintő módszerek ekvivalenciáját bizonyítjuk be. Jelölje $\Delta V_t^{\lambda}(s_t)$ a t-edik pillanatbeli $V(s_t)$ felülírást λ-hozam algoritmus (1.31) esetén, és $\Delta V_t^{TD}(s)$ pedig a t-edik időpillanatbeli s érték felülírást a mehanikusan definiált TD(λ) esetén (1.34). A célunk az, hogy megmutassuk a felülírások összege az epizód végén mindkét algoritmus esetében megegyezik:

$$\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) = \sum_{t=0}^{T-1} \Delta V_t^{\lambda}(s_t)\, \mathcal{I}_{ss_t}\,, \quad \forall s \in \mathcal{S}\,, \tag{1.35}$$

ahol $\mathcal{I}_{ss_t}$ azonosító jelző függvény, amely 1 ha $s = s_t$, és minden más esetben 0.

Az első megjegyzés az, hogy az összegyűjtött emlékeztető nyom explicit módon a következőképpen írható:

[5] Az on-line eset bizonyítása hasonlóan történik.

$$e_t(s) = \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathcal{I}_{ss_k} .$$

Eképpen a (1.35) egyenlet bal oldala a következő formában írható le:

$$\begin{aligned}
\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) &= \sum_{t=0}^{T-1} \alpha\delta_t \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathcal{I}_{ss_k} \\
&= \sum_{k=0}^{T-1} \alpha\delta_k \sum_{t=0}^{k} (\gamma\lambda)^{k-t} \mathcal{I}_{ss_t} \\
&= \sum_{k=t}^{T-1} \alpha\delta_k \sum_{t=0}^{T-1} (\gamma\lambda)^{k-t} \mathcal{I}_{ss_t} \\
&= \sum_{t=0}^{T-1} \alpha\mathcal{I}_{ss_t} \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} \delta_k . \qquad (1.36)
\end{aligned}$$

Most nézzük a (1.35) egyenlet jobb oldalát. Alkalmazzuk a λ-hozam algoritmus sajátos felülírását:

$$\begin{aligned}
\frac{1}{\alpha}\Delta V_t^{\lambda}(s_t) &= R_t^{\lambda} - V_t(s_t) \\
&= -V_t(s_t) + (1-\lambda)\lambda^0 \left[r_{t+1} + \gamma V_t(s_t)\right] \\
&\quad + (1-\lambda)\lambda^1 \left[r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})\right] \\
&\quad + (1-\lambda)\lambda^2 \left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \quad + \gamma^3 V_t(s_{t+3})\right] \\
&\qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad\qquad \ddots
\end{aligned}$$

Vizsgáljuk meg az első oszlopot (r_{t+1}) a szögletes zárójelen belül, amely λ hatványaival ill. a normálási tényezővel felszorozott. Tudjuk, hogy az oszlop súlyfaktorainak összege 1. Eképpen kihúzhatjuk az első oszlopot és helyettesíthetjük r_{t+1} súlyozatlan formulával. Ezt az egyszerű fogást alkalmazhatjuk a második oszlopra is a második sortól kezdve, amelynél a súlyok összege: $\gamma\lambda r_{t+2}$. Minden oszlopra elvégezve a következőket kapjuk:

$$\begin{aligned}
\frac{1}{\alpha}\Delta V_t^\lambda(s_t) &= -V_t(s_t) \\
&\quad + (\gamma\lambda)^0 \left[r_{t+1} + \gamma V_t(s_{t+1}) - \gamma\lambda V_t(s_{t+1})\right] \\
&\quad + (\gamma\lambda)^1 \left[r_{t+2} + \gamma V_t(s_{t+2}) - \gamma\lambda V_t(s_{t+2})\right] \\
&\quad + (\gamma\lambda)^2 \left[r_{t+3} + \gamma V_t(s_{t+3}) - \gamma\lambda V_t(s_{t+3})\right] \\
&\quad \vdots \\
&= (\gamma\lambda)^0 \left[r_{t+1} + \gamma V_t(s_{t+1}) - \gamma\lambda V_t(s_t)\right] \\
&\quad + (\gamma\lambda)^1 \left[r_{t+2} + \gamma V_t(s_{t+2}) - \gamma\lambda V_t(s_{t+1})\right] \\
&\quad + (\gamma\lambda)^2 \left[r_{t+3} + \gamma V_t(s_{t+3}) - \gamma\lambda V_t(s_{t+2})\right] \\
&\quad \vdots \\
&\approx \sum_{k=t}^{\infty} (\gamma\lambda)^{k-t}\,\delta_k \\
&\approx \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t}\,\delta_k
\end{aligned}$$

Az előbbi esetben a közelítés pontos az off-line felülírás esetén, amikor V_t megegyezik minden t időpillanatban. Az utolsó lépés pontos (de nem a közelítés), mivel az összes δ_k formulát elhagyjuk az összes terminális állapot után képzelt lépéseknél. Az összes ilyen lépésnél a jutalom értéke zéró így a hozzá tartozó δ érték is nulla. Eképpen megmutattuk, hogy az (1.35) egyenlet jobb oldala a következőképpen írható fel:

$$\sum_{t=0}^{T-1} \Delta V_t^\lambda(s_t)\,\mathcal{I}_{ss_t} = \sum_{t=0}^{T-1} \alpha\,\mathcal{I}_{ss_t} \sum_{k=t}^{T-1} (\gamma\lambda)^{(k-t)}\,\delta_k\,,$$

Ezzel a bizonyítást befejeztük.

1.3. Függvény approximátorok

Korábban az értékelő függvényt véges sok állapot-akció párossal írtuk le. Így az értékelő függvényünket egy egyszerű táblázat segítségével valósítottuk meg. Ez sajnos sok állapot-akció párosra nem megfelelő szerkezet, mert nemcsak a táblázat helyigénye nagy, hanem a teljes feltöltéséhez szükséges idő is. A kulcskérdés az általánosításban rejlik. Hogyan érhetjük el, hogy korlátozott számú kísérlet eredményének általánosításával is jó közelítését kapjuk az értékelő függvényünk egész tartományán?

Ez nehéz probléma, mivel a legtöbb feladatban, ahol megerősítéses tanulást szeretnénk alkalmazni, kevés számú állapot ismeretében kell olyan állapotokra is becslést adni amikkel még sohasem találkoztunk. Ez a helyzet például a folytonos állapot-akciótérnél, vagy a vizuális kép érzékelésénél is. Így jutunk el a példák alapján történő általánosítási eljárásokhoz, aminek jól kidolgozott algoritmusai vannak.

1.3.1. Az értékelő függvény becslésése függvény-approximátorokkal

Az általánosításnak azt a formáját, ahol mintavételezésekből általánosítva valósítjuk meg a függvényünket, *függvényapproximátornak* nevezzük. Ez a módszer a megerősítéses tanulásnak egy alkalmazása.

Az állapotot értékelő függvényünket a t pillanatban V_t-vel jelöljük, és ezesetben nem táblázattal hanem egy $\vec{\theta_t}$ paraméter vektorral adjuk meg, azaz $V_t = V(\vec{\theta_t}(\vec{s_t}))$. V_t-t választhatjuk egy mesterséges neurális hálózat kimenetének is. Ebben az esetben a $\vec{\theta_t}$ vektor a hálózat súlyvektoraiból áll. Tipikusan a paraméterek száma (a paraméter vektor komponenseinek a száma) sokkal kisebb, mint az állapotok száma. Következésképpen ha változtatunk egy paraméter értékén, akkor sok állapot értékét módosítottuk. Minden becslést az értékek mentésével valósítunk meg. Jelöljük $s \mapsto v$-vel az s állapothoz tartozó mentést. Ez az, amivel a következő időpontban és állapotban becsülni fogjuk az értékelő függvényünket. Az érték mentés következőképpen alakul:

DP esetén	$s_t \mapsto E_\pi\{r_{t+1} + \gamma V_t(s_{t+1}) \vert s_t = s\}$,
TD(0) esetén	$s_t \mapsto r_{t+1} + \gamma V_t(s_{t+1})$
és TD(λ) esetén	$s_t \mapsto R_t^\lambda$.

A táblázatos módszer triviális módon adja vissza a kívánt értékelő függvényt. Az s állapothoz tartozó táblabejegyzést közelítem a tőle elvárt v értékhez. A tetszőlegesen

összetett függvényapproximátor tanítása pont ilyen $s \mapsto v$ állapot-érték mentésekből álló tanítási párokkal valósítható meg. Ekkor a megerősítéses tanulás minden módszerét alkalmazhatjuk. A legjobb neurális hálózatok és statisztikus módszerek legtöbbje a megerősítéses tanulással ellentétben statikus és lassú tanulást igényelnek. Ezzel szemben a megerősítéses tanulás képes a környezetével kölcsönhatva ahhoz igazodni. Ehhez olyan eljárások kellenek amelyek nem stacionárius célfüggvényt is kezelni tudnak.

Teljesítmény értékelések a függvény közelítő eljárásokra: a legtöbb felügyelt tanulási módszer az átlagos *hiba-négyzet* (Mean-Square Error MSE) minimalizálására törekszik az állapotok egy bizonyos eloszlásával.

$$MSE(\vec{\theta}_t) = \sum_{s \in S} P(s)[V^\pi(s) - V_t(s)]^2, \tag{1.37}$$

ahol P az állapotok hibájának a súlyeloszlása. Ez azért fontos, mert általában nem lehet a hibát nullára csökkenteni minden állapotban. Az egyenletes hibaeloszlás érdekében a hibák súlyozásának eloszlását tegyük egyenlővé a tanítási állapotok eloszlásával. Ezt az eloszlást *aktív politika eloszlásnak* hívjuk ha egy olyan értékelő függvényt becsülünk ami egy környezettel kölcsönható ügynök politikájához tartozik. Ez egy olyan eloszlást valósít meg, amely segítségével az értékelő függvényt csak azokban az állapotokban tanuljuk, ahol az ügynök-környezet rendszer előfordul.

A hiba minimalizálása egy $\vec{\theta^*}$ vektor megtalálásával egyezik meg, ahol $MSE(\vec{\theta^*}) \leq MSE(\vec{\theta})$ minden $\vec{\theta}$-ra. Ennek elérése egyszerűbb esetekben, mint a lineáris függvény approximátorok, lehetséges, nem lineárisoknál csak egy lokális minimum garantált. De a legjobb becslés nem minden esetben a legoptimálisabb a hiba szempontjából.

1.3.2. Gradiens keresési eljárás

A gradiens keresési eljárás egy a függvény approximátorok terén széleskörben elterjedt eljárás, és nagyon jól illeszkedik a megerősítéses tanulás koncepciójába is. Ebben az esetben a paraméter vektor $\vec{\theta}_t = (\theta_t(1), \theta_t(2), \ldots, \theta_t(n))^T$ és $V_t(s)$ egyenletesen differenciálható függvénye $\vec{\theta}_t(\vec{s_t})$-nek minden $s \in S$-re. A hiba minimalizálát a négyzetes hiba függvény $\vec{\theta}_t$ szerinti gradinens irányába haladva érhetjük el, azaz:

$$\begin{aligned} \vec{\theta}_{t+1} &= \vec{\theta}_t - \frac{1}{2}\alpha \nabla_{\vec{\theta}_t} [V^\pi(s_t) - V_t(s_t)]^2 \\ &= \vec{\theta}_t + \alpha [V^\pi(s_t) - V_t(s_t)] \nabla_{\vec{\theta}_t} V_t(\vec{s_t}) \end{aligned} \tag{1.38}$$

ahol α a lépésköz paramétere és $\nabla_{\vec{\theta}_t} f(\vec{\theta}_t)$ az f függvény $\vec{\theta}_t$ paramétervektor szerinti gradiense. Ezen negatív gradiens irányába haladva a paraméter vektorok terében csökken a négyzetes hiba. Konvergencia csak abban az esetben garantált, ha a lépésköz paraméterünk az idővel nullához tart, ekkor teljesül a standard sztochasztikus feltétel. A hibát természetesen egy lépésben is nullára tudnánk csökkenteni egy adott állapotban, de a többi állapotban ez rontaná a becslésünket. Általában $V^\pi(s_t)$-t nem ismerjük pontosan, annak csak zajos, vagy becsült értéke áll rendelkezésünkre. Ekkor eljárásunkban v_t-t helyettesitünk helyette *(Bootstrapping)*.

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\,[v_t - V_t(s_t)]\nabla_{\vec{\theta}_t} V_t(\vec{s_t})$$

Ha v_t *független becslése (unbiased estimate)*,$V^\pi(s_t)$-nek, azaz $E\{v_t\} = V^\pi(s_t)$, minden t-re, akkor az α lépésköz nullához való konvergenciája garantálja a sztochasztikus közelítési feltételt.

V_t^π helyett a TD hozamát (v_t) vagy valamilyen átlagát írva megkapjuk a jövőbe tekintő TD(λ) gradiens keresési eljárást:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\,[R_t^\lambda - V_t(s_t)]\nabla_{\vec{\theta}_t} V(s_t) \tag{1.39}$$

Sajnos $\lambda < 1$-re R_t^λ nem független becslése a $V^\pi(s_t)$-nek, és nem konvergál egy lokális optimumhoz. Ennek ellenére egészen jó eredményeket lehet az ilyen *bootstrap* eljárásokkal elérni.

A jövőbe tekintő eljárásokkal az a baj, hogy csak a hozam összegyűjtése után (epizód) tudunk értékelni, s így az értékelő függvényünket módosítani. Ezzel szemben a visszatekintő eljárásokkal az epizódok közben is lehetséges a hangolás, mert az információ rendelkezésre áll. A TD(λ) múltba tekintő változata a következőképpen alakul:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\,\delta_t \vec{e}_t \tag{1.40a}$$

$$\delta_t = r_{t+1} + \gamma\, V_t(s_{t+1}) - V_t(s_t), \text{ és} \tag{1.40b}$$

$$\vec{e}_t = \gamma\,\lambda\,\vec{e}_{t-1} + \nabla_{\vec{\theta}_t} V_t(s_t). \tag{1.40c}$$

ahol δ_t a szokásos TD hiba, $\vec{e}_t$ pegig az úgynevezett *emlékeztető nyom (eligibility)* vektor amely a múltbeli állapotok egyre csökkenő súllyal vett lenyomata.

1.3.3. Lineáris eljárás

Az egyik legfontosabb eljárás a lineáris eljárás, amelyben V_t lineáris függvénye a $\vec{\theta_t}$ paraméterverktornak.

$$V_t = \vec{\theta_t}^T \vec{\phi}_{s_t} \tag{1.41}$$

Ebben az esetben a $\nabla_{\vec{\theta_t}} V_t = \vec{\phi}_{s_t}$ tulajdonságvektorral. Így leegyszerűsödött a $\vec{\theta}^*$ keresésére használt egyenletünk is. Valamint a lineáris eljárásnak meg van még az a jó tulajdonsága, hogy csak egyetlen egy optimális paraméter vektor, vagy tartomány létezik. Így garantált az is, hogy konvergencia esetén az a globális optimumhoz fog tartani.

Az előző fejezetben tárgyalt $TD(\lambda)$ eljárás is konvergál abban az esetben, ha a lépésközparamétere csökken a lépésekkel. Ekkor természetesen nem a globális optimumhoz fog kovnergálni, hanem egy olyan $\vec{\theta}_\infty$ paraméterverktorhoz, amelynek a hibája a következő egyenlettel adható meg:

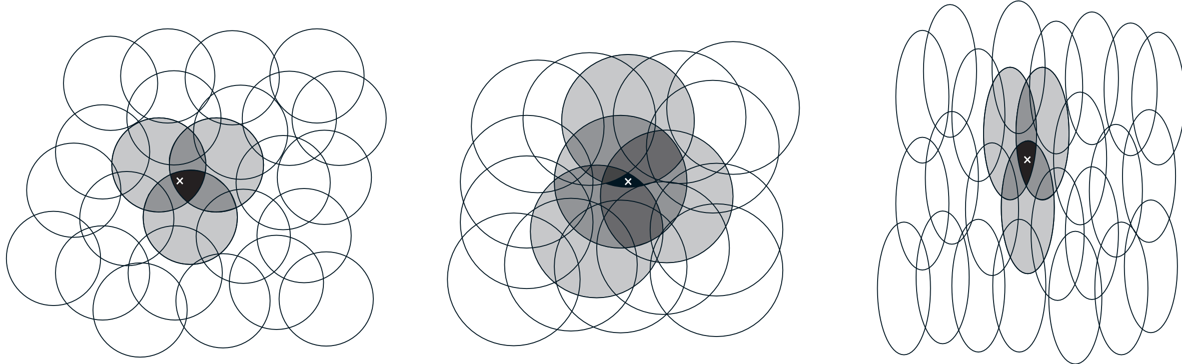
$$MSE(\vec{\theta}_\infty) \leq \frac{1-\gamma^\lambda}{1-\gamma} MSE(\vec{\theta}^*).$$

A lineáris eljárások nemcsak az elmélet egyszerűsége miatt érdekesek, hanem hatékonyságuk miatt is, mind adatmennyiség, mind számítási gyorsaságban is. Nem függ kritikusan attól, hogy hogyan alakítom ki az állapotokból a tulajdonságvektort.

Bináris tulajdonság vektor

Azzal, hogy az állapotokból hogyan állítjuk elő a tulajdonságvektort priori információt viszünk a eljárásba. Így például az állapottér fontosnak tartott tartományait jobban ki lehet emelni, azzal, hogy a tulajdonságvektorban több elem kapcsolódik nagyobb súllyal az adott területhez, míg a kevésbé fontosakhoz, kevesebb.

Minden egyes kör a tulajdonságvektor egy elemének érzékelési területét hivatott reprezentálni. Az *a* esetben a tulajdonságvektor kevés komponense reprezentál egy állapotot. A *b* esetben sok komponens reprezentálja ugyan azt az állapotot. Míg *c* esetben az ábrázolás asszimmetrikus, vízszintesen sokkal nagyobb felbontást eredményez mint függőlegesen. Amennyiben a tulajdonságvektor komponenseit úgy választjuk meg, hogy 1 ha az állapot az adott komonens érzékelési tartományán belül van és 0 ha kívül, az úgynevezett *bináris (coarse)* kódoláshoz jutunk. Ezt a számítógép architetúrája miatt hatékony algoritmusokkal lehet megvalósítani. Ez a hatékonyság azonban információvesztést eredményez



1.15. ábra. a. szűk ábrázolás b. széles ábrázolás c. aszimmetrikus ábrázolás

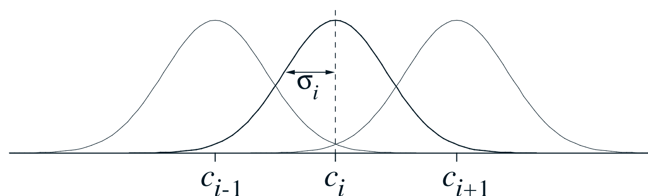
a diszkretizáció erős nemlinearitása miatt, mert csak azt tudjuk megmondani, hogy melyik tartományban van az állapotunk azt nem, hogy az ábran fekétvel jelzett területen belül hol található.

A *bináris* kódolás a számítógépek természetéből fakad, és a valóságot durva nemlinearitásokkal közelíti. A pontosság a tulajdonságvektor nagyságától függ. Ennél természetesebb és pontosabb ábrázolása egy állapotnak ha *bázisfüggvényeket* használunk a *coarse* kódolás buckafüggvényei helyett.

Bázisfüggvények

Legyen a tulajdonságvektor i. komponense:

$$\Phi_i = e^{-\frac{\|s-c_i\|^2}{2\sigma_i^2}} \tag{1.42}$$



1.16. ábra. Bázisfüggvények egy dimenzióban

ahol c_i az i. bázisfüggvény maximális helye, σ_i pedig a félérték szélessége. Természetesen a norma és a távolság metrikáját szabadon lehet megválasztani a feladathoz. A tanulást

a (1.39) és a (1.41) egyenletek határozzák meg. A *bázisfüggvények* előnye a bináris tulajdonságvektorral szemben az, hogy tetszőleges pontossággal tudja az állapotokat leképezni, míg hátránya a nagyobb számolási igény a lebegőpontos műveletek miatt.

1.4. Bibliográfiai és történeti megjegyzések

Az első fejezet Sutton,R.S., és Barto,A.G. (1998) könyvén alapul.

1.1.4. fejezet. Minsky (1967): mélyebb értekezés az állapot leírásról.
Az MDPs elméletével Bertsekas(1995), Ross(1983), White(1969), és Whittle (1982,1983) foglalkoztak.

1.1.5. fejezet. Watkins (1988)$\mathcal{Q}$-tanulási algoritmussal becsüli a$\mathcal{Q}^*$ állapot-akció párt értékelő függvényt, amelyet konzekvensen $\mathcal{Q} - \textit{függvénynek}$ is neveznek.
Amit mi a V^*-hoz tartozó Bellman-egyenletnek nevezünk, először Richard Bellman (1957a) vezette be "alapvető működési egyenlet" néven.

A dinamikus programozás elnevezés Bellmantól (1957a) ered, aki megmutatta, hogy hogyan kell alkalmazni ezt a módszert kúlönféle problémák megoldására. A DP-t szigorúan csak MDPs megoldására használjuk, bár a módszer más típusú problémák esetén is alkalmazható. Kumar és Kanal (1988) egy általánosabb DP nézetet ad meg.

1.2.1. fejezet. A politika javítása és a politika iteráció algoritmusa Bellmantól (1957a) és Howardtól (1960) származik.
Az érték iteráció általunk leírt formája Puterman és Shin (1978) közelítésén alapszik.
Bertsekastól (1987) származik az a rész amely megmutatja, hogy az érték iteráció hogyan találja meg az optimáli politikát.
Az aszinkron DP algoritmusok Bertsekastól (1982,1983) származik, aki osztott DP algoritmusoknak nevezi.

A TD tanulás ötlete az Samuel (1959) és Klopf (1972) munkáiban jelenik meg először. Szintén utal a TD tanulásra Holland (1975,1976) az érték jóslás összefüggésében.

1.2.2. fejezet. A fejezet legtöbb része Suttontól (1988) származik,
beleértve a TD(0) algoritmust és a időbeli differencia módszer
elnevezést.
A TD(0) módszer konvergációját Sutton (1988) és 1 valószínűséggel
Dayan (1992) bizonyította Watkins és Dayan (1992) alapján.
A Sarsa algoritmus első felfedezője Rummery és Niranjan (1994) volt,
akik módosított *Q-tanulásnak* nevezték.
A Sarsa elnevezés Suttontól (1996) származik.

1.2.3. fejezet. Az emlékeztető nyomok előretekintő látásmódja
(az n-lépeés hozam és a λ hozam) Watkinstól (1989) származik.
A fejezetben leírt forma a kissé módosított Jaakkola, Jordan, and
Singh (1996) szemléletén alapul.
Az emlékeztető nyomok módszerének használata Klopf munkáján
alapul (Sutton 1978a, 1978b, 1978c; Barto and Sutto, 1981a, 1981b;
Sutton and Barto, 1981a; Barto, Sutton, and Anderson, 1983;
Sutton, 1984).
A $TD(\lambda)$ algoritmus Suttonnak (1988) köszönhető.
Az előre- és a visszatekintő módszerek ekvivalenciáját Sutton (1988)
bizonyította, az általános esetre pedig Watkins (1989).

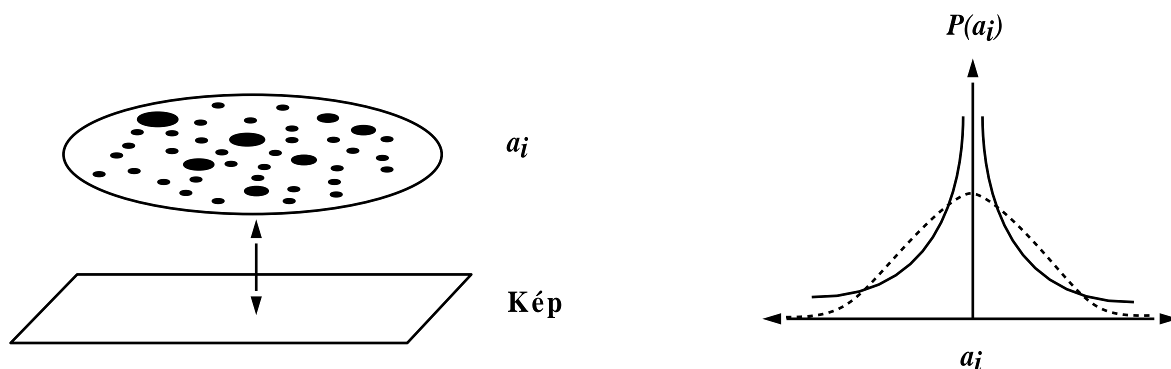
1.3. fejezet. Widrow és Hoff (1960) ismertette a legkisebb hiba négyzet (LMS)
algoritmust.
TD(λ) eljárás lináris gradiens-csökkentési függvény aproximátorral
módszert Sutton (1984, 1988) fedzte fel.
A lineáris függvény approximátor jelenlegi bemutatása Barto (1990)
munkáján alapszik.

2. fejezet

Ritka reprezentáció

A *ritka (sparse)* reprezentációval a látott természetes képek agybeli feldolgozásában találkozhatunk [26]. A természetes képek olyan tulajdonságokkal rendelkeznek, amelyeket nem lehet lineáris párkorrelációkkal leírni. A struktúrájuk lokális, irányított és jellemző sávszélességgel rendelkeznek. A lokális struktúrákat a Fourier komponensek fázis spektrumával lehet jellemezni. Az irányított struktúrák jellemzésére legalább három-pont korreláció szükséges. A sávkorlátolt struktúrák megfogására szintén szükség van a fázis spektrumra. Így a lineáris párkorreláció, ami csak a Fourier komponensek amplitúdóját karakterizálja kevés a természetes képek hatékony karakterizálására.

A képek hatékony kódolása a reprezentáció ritkaságának a maximalizásával lehetséges. Ez azt jelenti, hogy a képek reprezentálásában résztvevő neuronok közzül csak kevés neuron aktív egyszerre, a többi neuron aktivitása elhanyagolható, valamint a reprezentáló neuronok száma jóval meghaladja a reprezentálandó kép képelemeinek a számát. Ez az aktivitáseloszlásban a 0 aktivitás körül egy nagy csúcsot eredményez.



2.1. ábra. a. Ritka reprezentáció b. aktivitás eloszlás

Legyen a képünk karakterizálására szolgáló egyenlet a következő:

$$I(x,y) = \vec{a}^T\vec{\phi}(x,y) \tag{2.1}$$

Célunk az, hogy megtaláljuk azon ϕ-k halmazát amelyek a teljes bemeneti teret kifeszítik és ritka kódolást eredményeznek, azaz az aktivitások eloszlása a 2.1 ábrához hasonlóan nagy csúcsa van a 0 aktivitás közelében. Az ilyen eloszásnak alacsony entrópiája van.

Optimizálási problémára átfogalmazva a következő költségfüggvény minimalizálására fogalmazzuk át a problémát:

$$E(a,\phi) = \sum_{x,y}[I(x,y) - \vec{a}^T\vec{\phi}(x,y)]^2 + \beta\sum_i S\left(\frac{a_i}{\sigma_i}\right), \tag{2.2}$$

ahol $\sigma_i^2 = \langle a_i^2\rangle$. Az első tag a kép ábrázolásának jóságát adja meg, a második egy költség az aktivitásokra, annál kisebb minnél kevesebb neuron aktív. $S(x)$ -re a következő függvények javasoltak: $-e^{-x^2}$, $log(1+x^2)$, $|x|$. Bayes-i értelmezésben az első tag egy logaritmikus valószínűséget, a második tag az együtthatókra feltételezett eloszlás logaritmusa. $S(x)$ választásához más-más eloszlás tarozik: egy ritka formájú, cauchy, exponenciális eloszlásnak megfelelő valószínűséget jelent értelemszerűen.

A tanulás a hibafüggvény minimalizálását (2.2) jelenti egy gradiens keresési eljárás segítségével (1.38). Ekkor a_i minden, a hálózatnak bemutatott képre a minimumig fejlődik E gradiense mentén:

$$\dot{\vec{a}} = \eta[\vec{b} - C\vec{a} - \frac{\beta}{\sigma_j}\beta\sum_i S'\left(\frac{a_i - \mu_i}{\sigma_i}\right)], \tag{2.3}$$

ahol $\vec{b} = \sum_{x,y}\vec{\phi}(x,y)I(x,y)$, $C = \sum_{x,y}\vec{\phi}(x,y)\vec{\phi}^T(x,y)$, η relaxációs konstans. Néhány lépést így végrehajtva ϕ_i-ket az $\langle E\rangle$ gradiense mentén változtatjuk:

$$\Delta\vec{\phi} = \eta_\omega\left\langle\left[I(x_n,y_m) - \hat{I}(x_n,y_m)\right]\vec{a}\right\rangle. \tag{2.4}$$

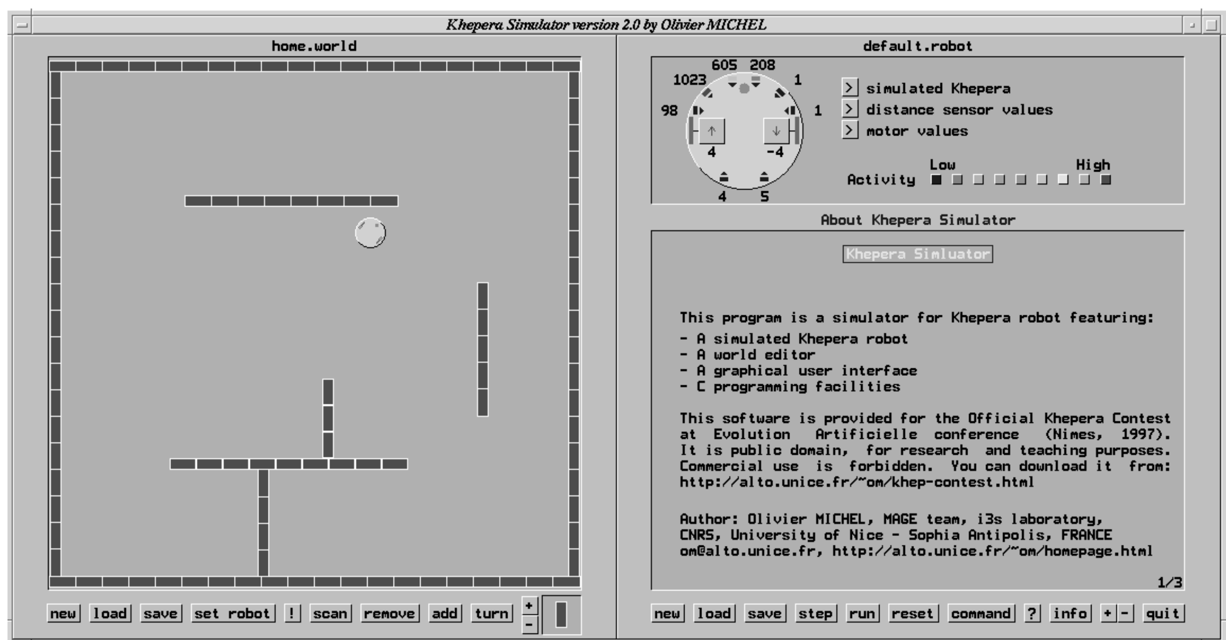
ahol $\hat{I} = \vec{a}\vec{\phi}^T(x_m,y_n)$ a rekonstruált kép, és η_ω a tanulási állandó. Minden bázisfüggvény ($\vec{\phi}$-k) hosszát úgy állítjuk be lépésenként, hogy az együtthatóinak a varianciája egyenlő legyen.

Ennek a rendszernek létezik egy egyszerű hálózati megfogalmazása a következő módon: minden kimeneti érték, a_i egy előrecsatolt bemeneti tagtól, $\vec{b}$, egy ismétlődőtagtól, $\underline{\underline{C}}\vec{a}$ és egy nemlineáris öngátló tagból, S' áll. Ez az öngátló tag az aktivitásokat különböző mértékben nyomja a nulla felé.

Ekkor olyan tulajdonságvektorhoz jutunk ($\vec{a}$) amely önszerveződő módon alakul ki az bemeneti állapotokból, mégha lassan is.

3. fejezet

Khepera robot szimulátor

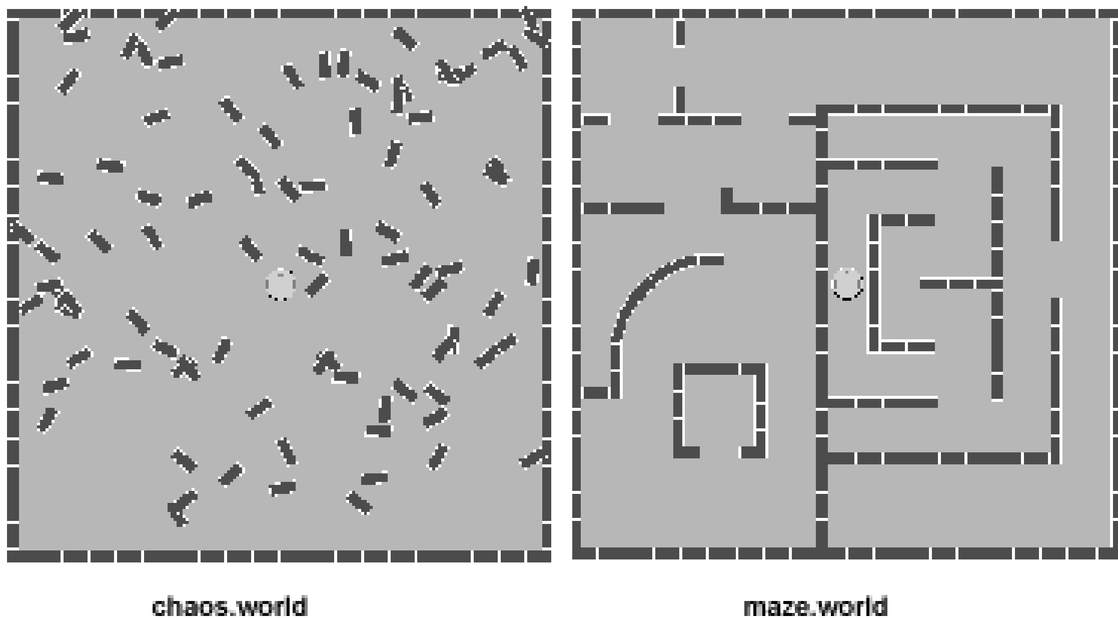


3.1. ábra. A khepera robot szimulator

A megerősítéses tanítási módszer egy alkalmazását a Khepera robot segítségével fogjuk megmutatni. Valódi robot helyett a robotot és a környező világot modellező Khepera szimulátort fogjuk használni [25]. A szimulátor két részből épül fel: a Khepera robot modelljéből, illetve a környező világból. A következő részekben ezek megvalósítását mutatjuk be.

3.1. A világ leírása

Minden egyes világ egy $1m * 1m$-es dimenziójú valós teret reprezentál. Ezek a világok téglákból és lámpákból épül fel. A szimulátor tartalmaz néhány alpvető beépített világot, amelyek közül a feladat megoldása során a 'home' illetve a 'maze' világokat használjuk(lásd: 3.1, 3.2 ábrák).

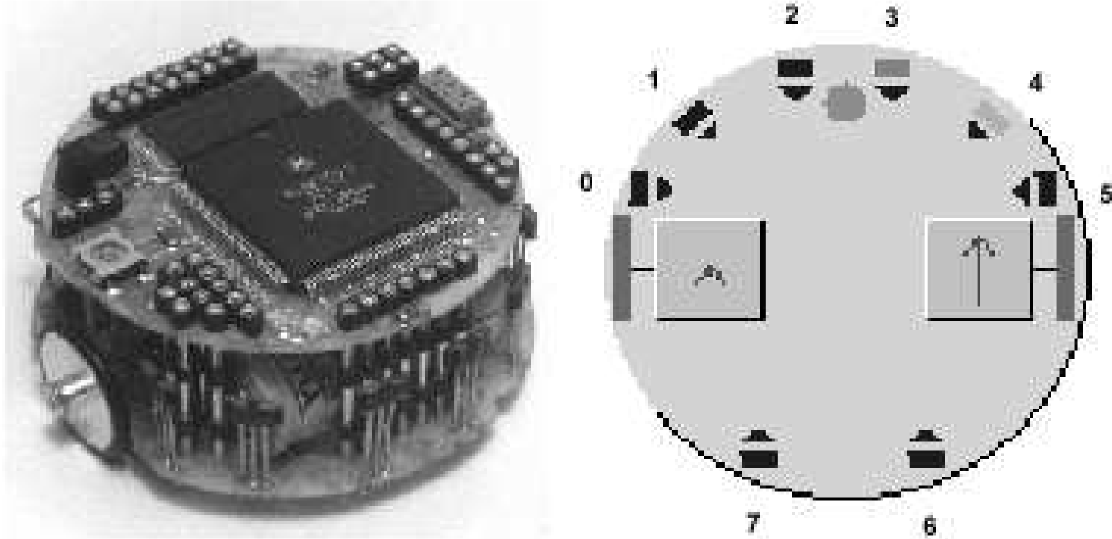


3.2. ábra. Két példa szimulált világ

3.2. A robot leírása

A Khepera egy egyszerű $5\,cm$ átmérőjű kör alakú robot, amely rendelkezik két kerékkel, és nyolc darab infravörös szenzorral. A szenzorok közül hat szenzor a robot elülső felén, egyenletesen elosztva helyezkedik el, a mardék kettő, pedig a robot hátulján található (3.3 ábra). A szenzoroknak kettő s funkciója van: távolságot és fényerőséget érzékelnek. A távolságérzékelés értékei 0 és 1023 közötti lehet, 0 azt jelenti, hogy nincs objektum a közelben, 1023 pedig azt, hogy a szenzorhoz közel van egy objektum (lehet, hogy már hozzá is ér a szenzorhoz). Hasonló módon a fényerő értéke 500 és 50 közötti értékeket vehet fel, 500 jelenti a sötétséget, 50 pedig azt, hogy nagyon közel fényforrás van. A köztes értékek

közelítőleg arányosan mutatja a távolságot, illetve a fényerőséget.



3.3. ábra. A valódi és a szimulált Khepera robot

3.2.1. A motor modellje

A robot mozgatása két párhuzamosan (bal és jobb oldalt) elhejezkedő motor segítségével történik. A szimulált robot motor értékét a felhsználó állítja be a -10,$+10$ tartományban. Ezek az értékek a motor forgási sebességét jelenti. A szimulátor a motorok akatuális értékeihez $\pm 10\,\%$ zajt ad, és az újabb pozíció meghatározása is $\pm 5\,\%$ zaj hozzáadásával történik.

3.2.2. A szenzorok modellje

A távolság meghatározása a következőképpen történik: a szimulált szenzor az előtte lévő területet háromszög formában 15 pontban felderíti. Az így kapott értékek segítségével a szimulátor meghatározza a távolság értéket, amelyhez még $\pm 10\,\%$ zajt ad. A fényerőség értéke, pedig a fényforrás ás a szimulált szenzor közötti távolságból, $\pm 5\,\%$ zaj hozzádása mellett adódik.

3.2.3. A robot vezérlése

A robot vezérlésére egy interfészt biztosít a program, amely segítségével minden egyes időlközben el tudjuk kérni a robot informéciókat, illetve a be tudjk állítani a motorok sebességét (Az egyetlen lehetőség a robot vezérlésére). A szimuláció futhat lépésenkénti vezérléssel, illetve folyamatosan, ahol mi határozzuk meg a futási idő végét.

4. fejezet

A megerősítéses tanítás módszer alkalmazása

Ebben a részben egy konkrét megvalósítását adjuk meg a környezet–ügynök modellnek. A feladat a következő: a Khepera robothoz olyan tanuló vezérlő rendszert akarunk építeni, amely adott világban a robotot a fal mellet haladásra készteti. A tanuló rendszer a megerősítéses tanulási módszert alkalmazza, azaz a jutalmak alapján tanulja meg, hogy mit kell tennie ahhoz, hogy a fal mellett maradjon. A következőkben az első fejezet analógiáját követve építjük fel a modellünket.

4.1. A környezet modellje

A környezeti résznek kettős feladata van: az első az, hogy az ügynök felé biztosítsa a rendszer aktuális állapotát és a pillanatnyi jutalmat, illetve az ügynök által küldött vezérlő jelek alapján változtassa meg a rendszer helyzetét. Az jelenlegi feladatban a környezet a Khepera robot szimulátor, kisebb kiegészítésekkel. Definiáljuk pontosan a rendszer állapotát, illetve a jutalom függvényt.

4.1.1. Az állapot reprezentációja

A rendszer állapotát egy 10 dimenziós $\vec{x}$ vektor formájában adjuk meg, ahol a vektor első nyolc komponense a nyolc távolságszenzor érték, az utolsó két komponenes pedig az aktuális motor értékek. A motorok értékei a következők lehetnek : (3,3); (3,0); (3,-3); (0,3); (0,0); (0,-3); (-3,3); (-3,0); (-3,-3) (Ezekkel az értékekkel vezéreljük a robotunkat). A motorok és a távolságszenzor értékei nagy mértékben eltérnek egymástól, ezért az összes értéket a [0,1] tartományba arányosan levetíjük, így egységes 0 és 1 közötti értékeket adó

10 konponensű állapotvektorhoz jutunk.
Állapotvektor: $\vec{x}$, ahol $\dim(\vec{x}) = 10,\ x_i \in [0,1]$ és $1 \leq i \leq 10$

4.1.2. A jutalom függvény

A jutalom meghatározására az eredeti távolság és motor értékeket használom. A célunk a következő: a robot egyenesen előre haladjon úgy, hogy a jobb vagy bal oldala (vagy mindkettő) a fal mellett legyen. Ennek alpján a jutalomfüggvényt a következőképpen definiáltam, részjutalmak összegeként (Jelölés: B – bal motor értéke; J – jobb motor értéke):

$r_1 = -1.0$, ha a robot ütközik a fallal.
$r_1 = 0.0$, egyébként

$r_2 = 0.03$, ha a robot nulladik vagy az ötödik szenzorának távolságértéke nagyobb mint 700.
$r_2 = 0.0$, egyébként

$r_3 = (B + J - 6.0) \div 6000$, a robot mozgásából eredő jutalom

$r = r_1 + r_2 + r_3$ az összjutalom

A jutalom definíciójából látszik, hogy maximális jutalom akkor van, ha a robot a falat követi a maximális sebességgel (3,3), ekkor a jutalom = 0.03, minimális jutalom pedig, ha a robot maximális sebességgel hátrafelé haladva ütközik (jutalom = -1.002).

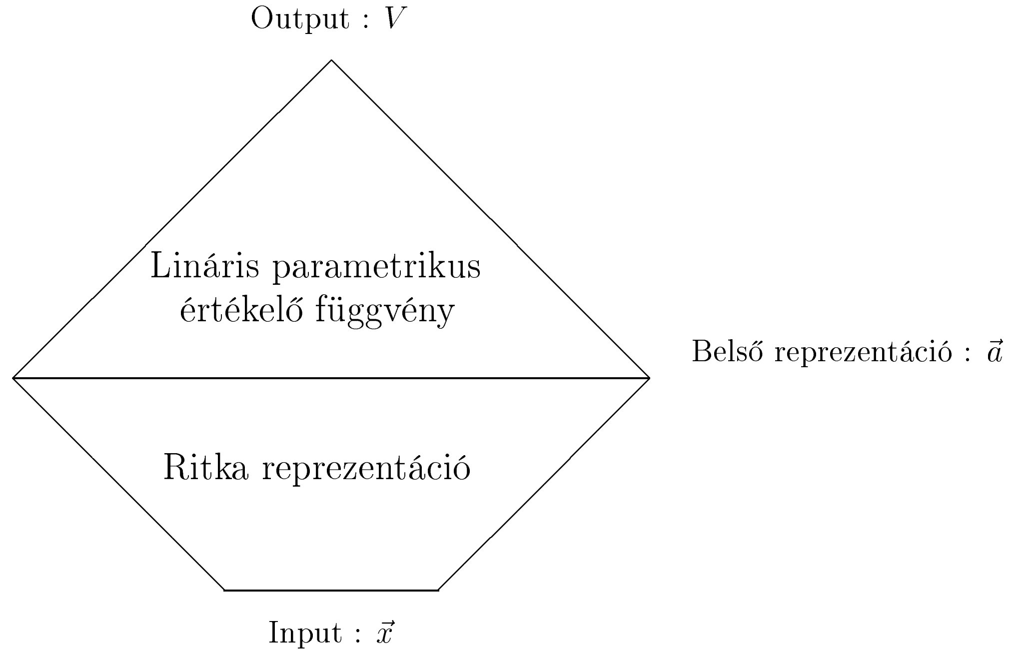
4.2. Az ügynök modellje

Az ügynök modellje két részből épül fel: az állapotot értékelő függvény (V) közelítéséből, és a pályatervező részből.

4.2.1. Állapotot értékelő függvény közelítése

A feladat nehézségét az állapottér folytonossága jelent, emiatt nincs lehetőségünk más esetekben jól működő $\mathcal{Q}(s,a)$ állapot-akciópárt értékelő függvény kialakítására.
A feladat megoldása ezért az állapotot értékelő függvény segítségével történik, mégpedig a lineáris parametrikus függvényapproximátorral kiegészített $TD(0)$ (+ emlékeztető nyomok

módszere) módszerrel, ahol a függvényapproximátor szerepét a ritka reprezentáció tölti be. A ritka reprezentáció, mint függvényapproximátor segítségével a V függvény előállítása lineárisan adódik. A V függvény előállítása:



4.1. ábra. Az állapot értékének kialakítása

Ezen kostrukció alapján a V függvény tanulása nem jelent mást, mint az θ (lásd: 1.3.3 fejezet) paramétervektor kialakítását.
Nézzük az alkalmazott ritka reprezentációra vonatkozó egyenleteket (2.3, 2.4 alpján mátrix-vektor formában):

$$\dot{\vec{a}} = Q^T * (\vec{x} - Q * \vec{a}) - \dot{S}(\frac{\vec{a} - \vec{\mu}}{\vec{\sigma}}) \tag{4.1a}$$

$$\dot{Q} = (\vec{a} - Q * \vec{a}) * \vec{x^T} \tag{4.1b}$$

, ahol $\dot{S} = \frac{x}{1+x^2}$ és $\dim(\vec{x}) = 10$, $\dim(\vec{a}) = 30$. A ritka reprezentáció kialakításának algoritmusa a 4.1. táblázatban látható.

```
Initialize Q, x⃗, a⃗
Repeat
    Get x⃗ (System)
    Repeat
        a⃗+ = α(Q^T ∗ (x⃗ − Q ∗ a⃗) − Ṡ((a⃗−μ⃗)/σ⃗))
    until a⃗ is converged
    Q+ = β((a⃗ − Q ∗ a⃗) ∗ x⃗^T)
until Q is converged
```

4.1. táblázat. Ritka reprezentáció kialakítása

A lineáris parametrikus értékelő függvény ($TD(0)$ módszer+emlékeztető nyomok módszere) egyenletei:

$$\delta_t = r_{t+1} + \gamma\, V_t(a_{t+1}) - V_t(a_t), \text{ és} \tag{4.2a}$$

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\, \delta_t \vec{e}_t \tag{4.2b}$$

$$\vec{e}_t = \gamma\, \lambda\, \vec{e}_{t-1} + \vec{a}_t. \tag{4.2c}$$

, ahol $\vec{a}$ vektor a ritka reprezentáció output vektora, V_t az értékelő függvény, θ ($\dim(\vec{\theta}) = 30$) a paramétervektor, e_t emlékeztető nyom. Az értékelő függvény tanítása epizódok segítségével történik. Egy epizódnak akkor van vége, ha a robot a fallal ütközik, illetve ha a megadott lépésszámot túllépi (4.2. táblázat).
Az adott $\vec{x}_t$állapot értékét a következő módon kapjuk (közvetetten):

$$\text{Ritka reprezentáció} : S\ :\ \vec{x}_t \rightarrow \vec{a}_t \tag{4.3a}$$

$$V(S(\vec{x}_t)) = \vec{\theta}_t^{\,T}\ \vec{a} \tag{4.3b}$$

4.2.2. Pályatervezése

A pályatervezés a következő módon valósul meg (4.3. táblázat). A lehetséges motorértekek halmazának legyen egy szűkített részhalmaza M, melynek elemei a következők: $M = \{(3,3),(3,0),(3,-3),(0,3),(0,0),(0,-3),(-3,3),(-3,0),(-3,-3)\}$. Az M nem más, mint a lehetséges akciók halmaza. Jelenleg a rendszer $\vec{x}_t$ állapotban van. Tekintsük lehetséges vezérlőjelnek M elemeit, s próbáljuk ki, hogy velük milyen új állapotba kerül a rendszer. Az így kapott állapotok közül válasszuk ki ϵ-mohó módon a legértékesebbet (azaz

Initialize $\vec{\theta} = \vec{0}$
Repeat for each episode
    Initialize Khepera position and $\vec{e} = \vec{0}$
    Repeat for each step in episode
        choose action in $\vec{x}$ state using $\epsilon - greedy\, policy$ and V function
        Step ahead with the Khepera Robot, (new position)
        Take r, $\vec{x}_{t+1} \rightarrow \vec{a}_{t+1}$
        Compute the new Value Function and the Eligibility Trace
            $\delta_t = r_{t+1} + \gamma V_t(a_{t+1}) - V_t(a_t)$
            $\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\, \delta_t \vec{e}_t$
            $\vec{e}_t = \gamma\, \lambda\, \vec{e}_{t-1} + \vec{a}_t.$
    until terminal state
until $\vec{\theta}$ is converged

4.2. táblázat. Az értékelő függvény tanulása

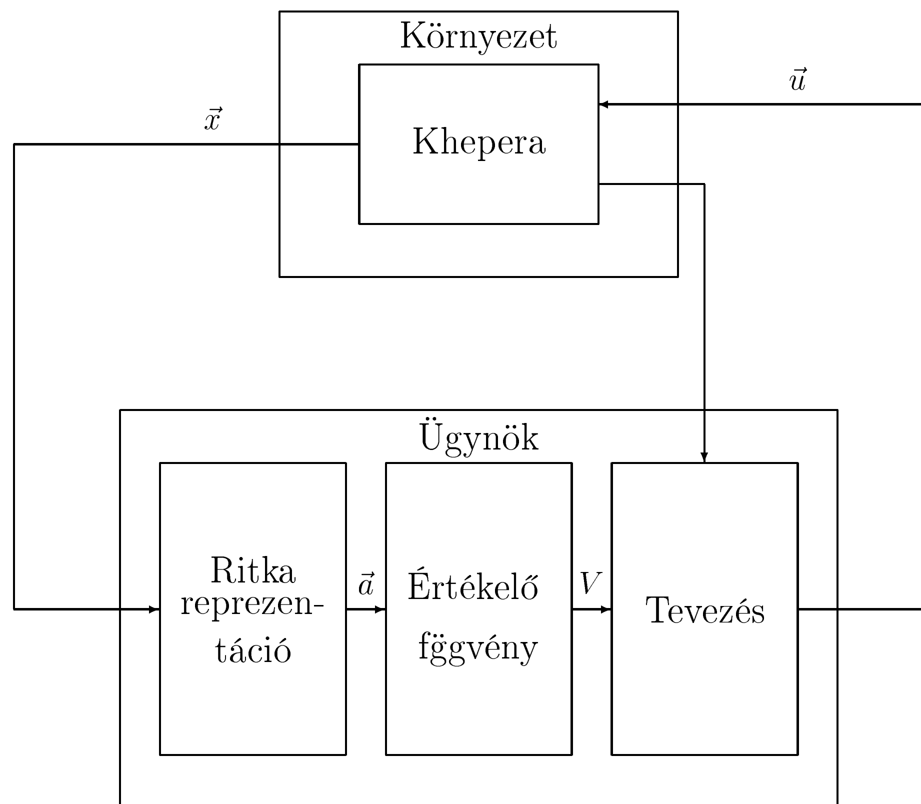
amelyhez tartozó ritka reprezentáció értéke a legnagyobb ($\max V(\vec{x})$)), és az állapot motor-konbinációját tekintsük az aktuális vezérlőjelnek ($\vec{u}$).

$M = \{(3,3),(3,0),(3,-3),(0,3),(0,0),(0,-3),(-3,3),(-3,0),(-3,-3)\}$
Repeat for each $\vec{m} \in M$
    $\vec{m} \rightarrow \vec{u}_{trying}$
    $\vec{x}_{t+1} \rightarrow \vec{a}$
    $\vec{u} = \max V(\vec{a})$ using $\epsilon - greedy\, policy$
$\rightarrow \vec{u}$

4.3. táblázat. A kontroll jel megadása

4.3. A szimuláció felépítése

A program blokkdiagrammja 4.2. ábrán látható.



4.2. ábra. A modell blokkdiagrammja

5. fejezet

Eredmények

A modell implementálása C++ nyelven történt (lásd melléklet). A következőkben a megvalósítás főbb lépéseit és a kapott erdményeket mutatom be.

5.1. Ritka reprezentáció kialakítása

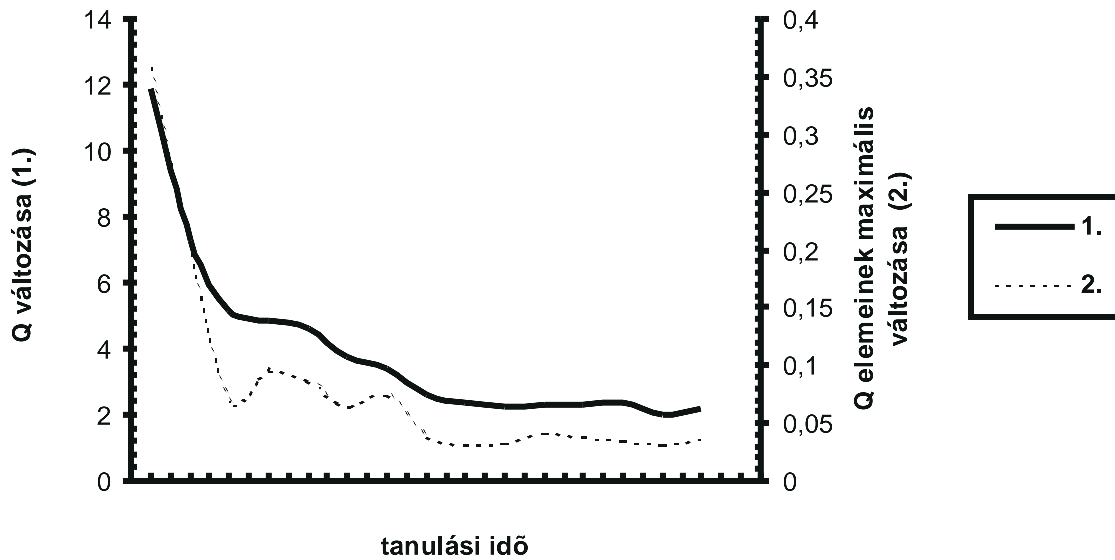
A kialakítás egyenletei a program jelöléseivel :

```
for(unsigned int i=0;i<RelaxNumber;i++)
   a+= DeltaT *( Qtx - QtQ*a  -  Loss * ((a-Mu) & Sigma).Map(SDot));

Q+= ( (AlphaQ  * x) - AlphaQ * Q*a)*a.T();
```

A paraméterek választása:

```
Input1Size= 10
Output1Size= 30
DeltaT= 0.15
Loss= 0.31
AlphaQ= 0.01
Mu= (0, ....,0)
Sigma= (0.04,...,0.04)
RelaxNumber= 2000
```

SDot függvény választása :

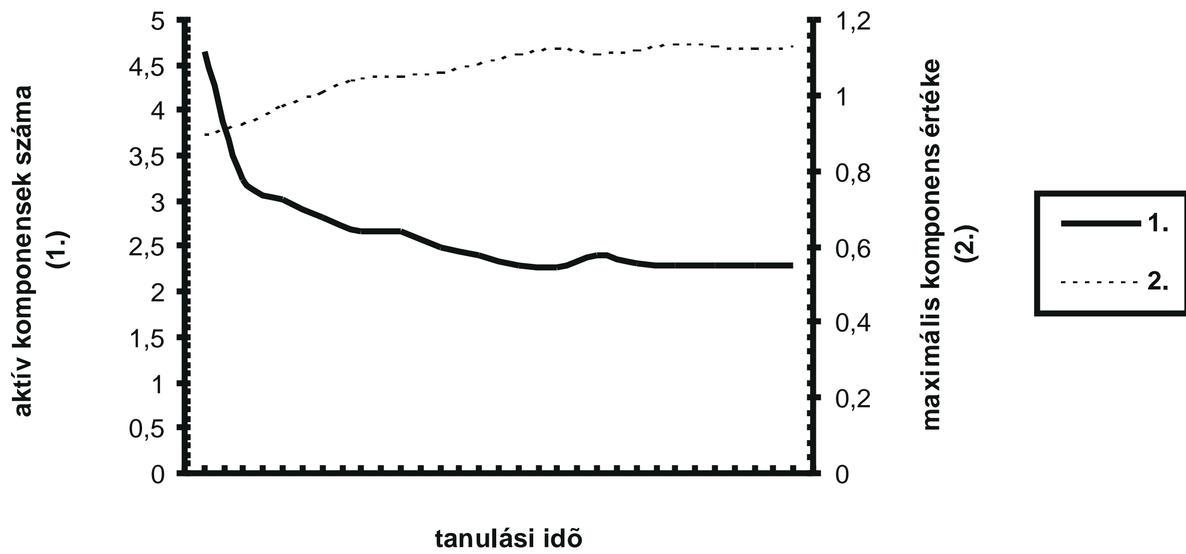


5.1. ábra. A memóriavektorok konvergálása (Q mátrix)

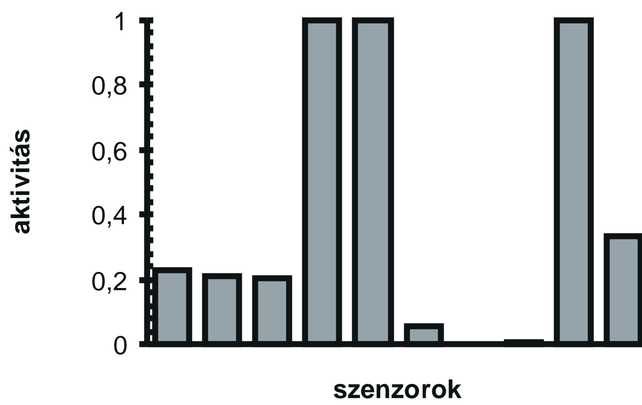
```
inline long double SDot(long double x)
{
   return ((2*x)/ (1+x*x));
};
```

A ritka reprezentáció tanítása 30000 véltlenszerű minta alapján történt. A tanítás alapjául szolgáló mintahalmazt a Khepera robot a 'maze.world' világba való véletlen elhelyezésével kaptam. A tanítás eredményei:

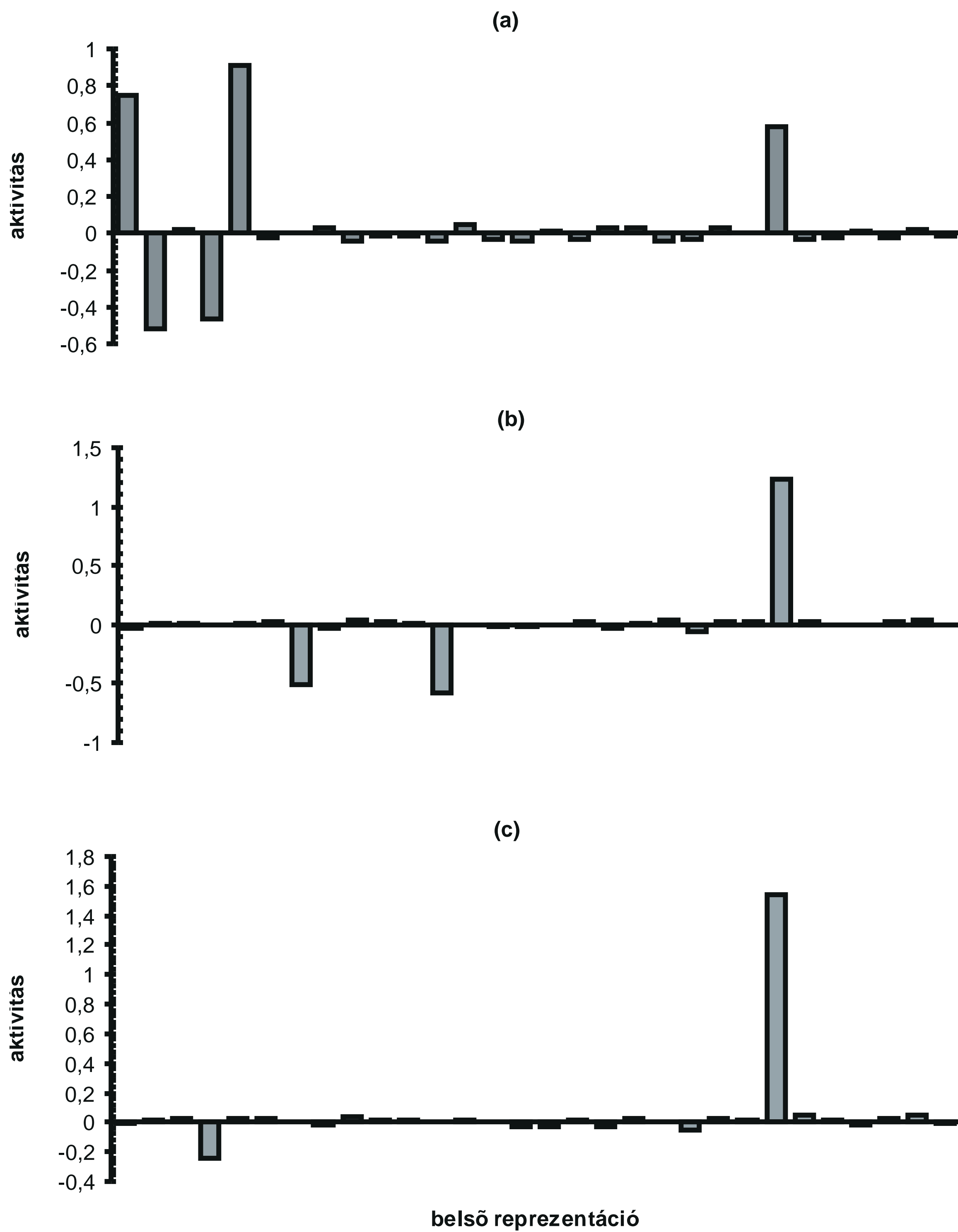
A 5.1. ábrán látható, hogy a Q mátrix változása fokozatosan csökken, a tanulás végén konvergál. A reprezentćió "ritka" tulajdonságát a 5.2 ábra mutatatja, azaz kevés komponens (kb 2.5) aktív, és ezezk értéke nagy (1.2). A továbbiakban egy adott inputra (5.3. ábra) adott válaszokat mutatom meg a tanulás különböző fázisaiban (5.4. ábra). Az 5.4.(a) ábra 1000, a (b) 10000 a (c) 30000 tanulási lépés után adott "ritka" reprezentációt mutatja. Az ábrán látható, hogy kezdetben – az azonos inputra – az output vektornak öt konponense volt aktív, a tanulás végére ez a szám kettőre csökkent. Ami még megfigyelhető, hogy a maximális komponens értéke a kezdeti 0.8 értékről 1.6-ra változik.



5.2. ábra. A "ritka" tulajdonság kialakulása.



5.3. ábra. Minta input vektor



5.4. ábra. A minta input vektor ritka reprezentációja a tanulás folyamat során

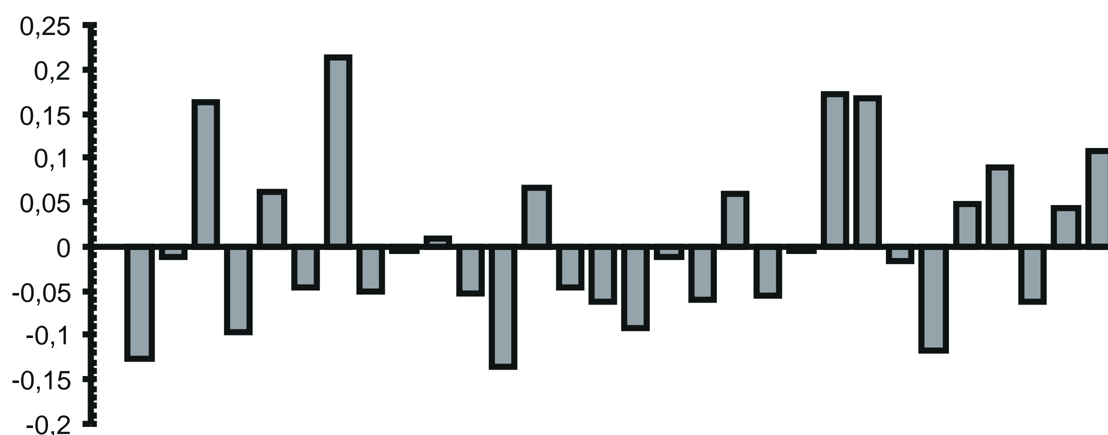
5.2. Az értékelő függvény tanulása

Az értékelő függvény tanítása epizódok alatt történik. Egy epizódnak akkor van vége, ha a robot a fallal ütközik, illetve ha a megadott lépésszámot túllépi. Az epizódok száma 2000, egy epizód hossza 300 lépés volt. A tanítási egyenletek a program jelöléseivel:

```
CurrentValue = (THETA.T()*input1)(0,0);

Delta = input2 - LastValue + (terminate ? 0 : Gamma * CurrentValue );
THETA += Alpha * Delta * E;
E = (Gamma * Lambda) * E + input1;
```

A paraméterek értéke:

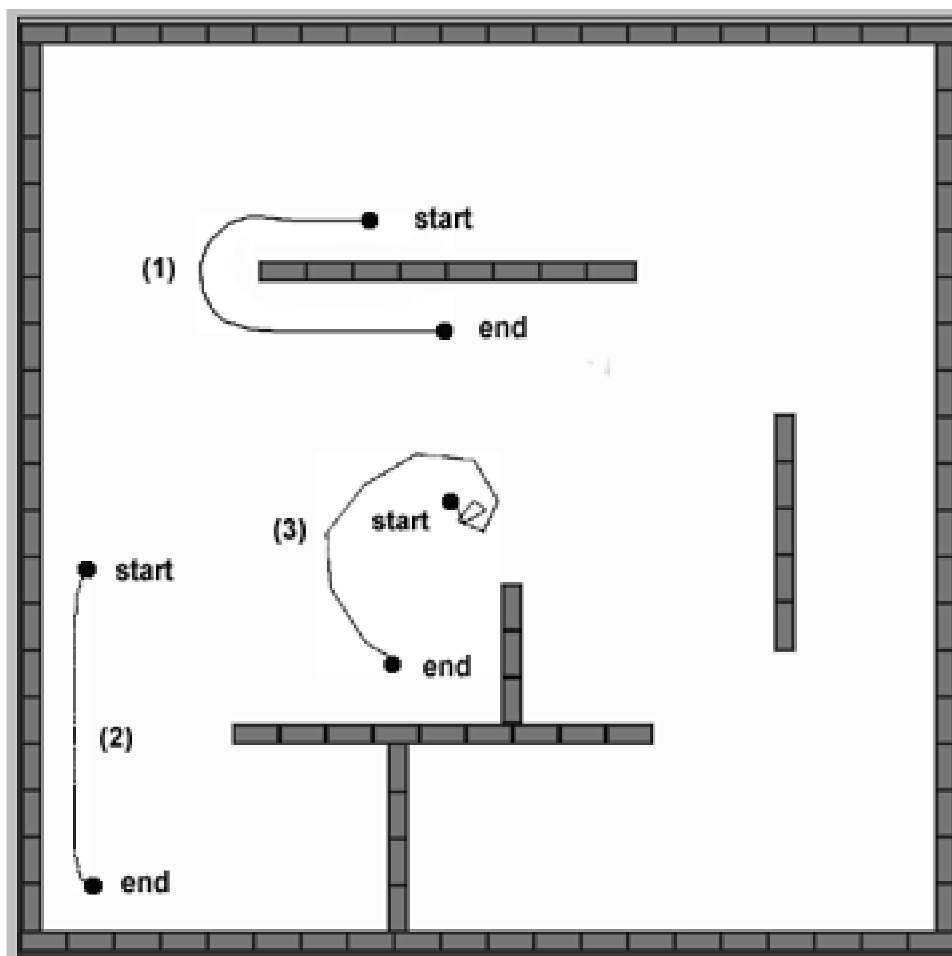
```
Input1Size= 30
Gamma= 0.9
Alpha= 0.01
Lambda= 0.5
```



5.5. ábra. A kialakult Theta vektor

5.3. A szimuláció eredményei

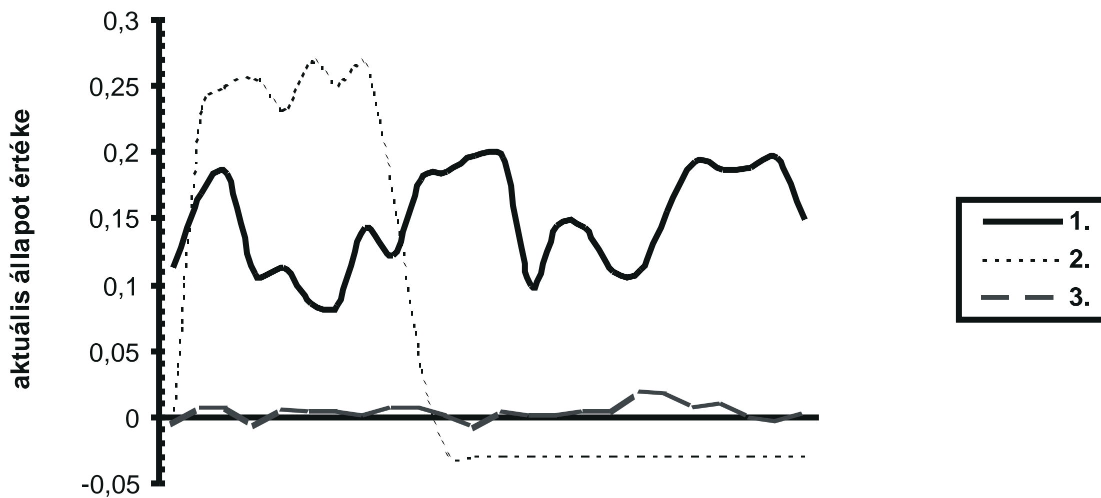
Az ritka reprezentáció és az értékelő függvény kialakítása után nézzük meg, hogy a kapott eredmény ténylegesen jó, azaz a robot falat követve halad. A vizsgálatot a "home.world"-ben végeztem, amely egy vizsonylag egyszerű világ. A robotot három különböző pozícióba helyeztem el: (1) a fal mellé a fallal párhuzamosan, (2) a fal mellé vele szemben, és (3) a faltól távol. A kapott eredményeket a 5.6. ábra, (a robot mozgásának felülnézeti ábrája), illetve a 5.7.ábra (amely a pillanyatni állapot értékeket mutatja) reprezentálja.



5.6. ábra. A Khepera robot mozgása (felülnézet)

A szimulációt (ügynök-környezet kapcsolat) megvalósító programrészlet:

```
void main()
{
   Sparse SP("default");
   LPValueFunction LP("default");
   Khepera KH(worldname.c_str());

   agent1 Agent(&SP,&LP,&KH);
   environment1 Environment(&KH);
   Control u;
   Robot* robot=(KH.GetInformation())->Robot;

   for(episodenumber=0;episodenumber<EPISODS;episodenumber++)
   {
      Agent.Reset();
      Environment.Reset();
      LP.Reset();
      KH.Reset();KH.SetRandomPosition();
      u.Set(0,0);
      terminate=false;

      for(stepnumber=0;stepnumber<STEPS;stepnumber++)
      {
          Environment.Iterate(u);

          if (robot->State==1 || (stepnumber+1)==STEPS) terminate = true;

          Agent.Planning(Environment.GetState(),Environment.GetReward(),
                         terminate);

          u = Agent.GetControlSignal();
      }
   }
}
```



5.7. ábra. Az aktuális állapotok értéke a szimuláció során

5.4. Konklúzió

A feladat megoldása részben sikerült, a robotot fal mellé helyezve követi azt megvalósítva a megerősítéses tanulás módszerét, de az üres térbe téve (a környéken nincs fal), a rendszer nem jól működik. Ennek elsősorban több oka lehet :

- a robot viszonylag kis távolságot érzékel, azaz "rövidlátó"
- a pálya tervezés kezdetleges. A pályatervezés lokális, nem tartalmaz globális információkat.
- a ritka reprezentációnál inforációvesztés
- a jutalom függvény rossz választása

A szimuláció felépítése lehetőséget ad a módosításokra (a tökéletesebb erdményért), azonban ez nem célja a dolgozatnak.
Összegezve: sikerült a megerősítéses tanulási elvén működő tanulórendszert létrehozni, amely a folytonos állapottérben célokat tudott teljesíteni.

Ábrák jegyzéke

1.1. Az ügynök-környezet kapcsolat. 2
1.2. A V^π-t (a), és a Q^π-t (b) meghatározó felösszegzési gráf. 7
1.3. A V^*-t (a), és az Q^*-t (b) meghatározó felösszegzési gráf. 8
1.4. Általánosított politika iteráció. 17
1.5. Politika kiértékelés és javítás a GPI-nél. 17
1.6. A TD(0)-hoz tartozó felösszegzési gráf. 19
1.7. Az állapotok és az állapot-akció párok alternáló sora. 20
1.8. Az n-lépéses TD jóslás felösszegzési gráfja. 22
1.9. Komplex felösszegzési gráf 25
1.10. λ-súlyozás 25
1.11. A TD(λ) módszer felösszegzési gráfja 26
1.12. A TD(λ), mint elméleti vagy előretekintő módszer 27
1.13. Az összegyűjtési nyom 27
1.14. A TD(λ) módszer mehanikus, vagy visszatekintő szemlélete. 29
1.15. a. szűk ábrázolás b. széles ábrázolás c. aszimmetrikus ábrázolás . 36
1.16. Bázisfüggvények egy dimenzióban 36

2.1. a. Ritka reprezentáció b. aktivitás eloszlás 40

3.1. A khepera robot szimulator 43
3.2. Két példa szimulált világ 44
3.3. A valódi és a szimulált Khepera robot 45

4.1. Az állapot értékének kialakítása 49
4.2. A modell blokkdiagrammja 52

5.1. A memóriavektorok konvergálása (Q mátrix) 54
5.2. A "ritka" tulajdonság kialakulása. 55

5.3. Minta input vektor 55
5.4. A minta input vektor ritka reprezentációja a tanulás folyamat során . . . 56
5.5. A kialakult Theta vektor 57
5.6. A Khepera robot mozgása (felülnézet) 58
5.7. Az aktuális állapotok értéke a szimuláció során 60

Táblázatok jegyzéke

1.1. Iteratív poltika-kiértékelés. 11

1.2. Politika iterálása 15

1.3. Érték iteráció. 16

1.4. TD(0) algoritmus V^{π} becslésére. 19

1.5. Sarsa: aktív politizálási TD szabályozás algoritmusa. 21

1.6. Az on-line TD(λ) algoritmus. 28

4.1. Ritka reprezentáció kialakítása 50

4.2. Az értékelő függvény tanulása 51

4.3. A kontroll jel megadása 51

Irodalomjegyzék

[1] Barto,A.G. (1990). Connectionist learning for control: An overview.In. T. Miller, R. S. Sutton, and P. J. Werbos (eds.), *Neural Networks for Control*, pp. 5-58. MIT Press, Cambridge, MA

[2] Barto,A.G. and Sutton,R.S. (1981a). Goal seeking components for adaptive intelligence: An initial assessment. Technical Report AFWL-TR-81-1070.

[3] Barto,A.G. and Sutton,R.S. (1981b). Landmark learning: An illustration of associative search. *Biological Cybernetics*, 42:1-8

[4] Barto,A.G., Sutton,R.S. and Anderson,C.W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics,* 13:835-846

[5] Bellmann, R.E. (1957a). *Dynamic Programming.* Princeton University Press, Princeton

[6] Bertsekas,D.P. (1982). Distributed dynamic programming. *IEEE Transactions on Systems, Man, and Cybernetics,* 27:610-616

[7] Bertsekas,D.P. (1983). Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27:107-120

[8] Bertsekas,D.P. (1987). *Dynamic programming. Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, NJ.

[9] Bertsekas,D.P. (1995). *Dynamic programming and Optimal Control.* Athena Scientific, Belmont, MA.

[10] Bertsekas,D.P., and Tsitsiklis,J.N. (1996). *Neural Dynamic Programming* Athena Scientific, Belmont, MA.

[11] Bradtke,S.J. (1993) Reinforcement learning applied to linear quadratic regulation. In *Advances in Neural Information Processing Systems: Proceedings of the 1992 Conference*, pp. 195-302. Morgan Kaufmann, San Mateo, CA.

[12] Bradtke,S.J. (1994) *Incremental dynamic programming for On-Line Adaptive Optimal Control*. Pf.D. Thesis, University of Massachusetts, Amherts. Appeared as CMPSCI technical Report 94-62.

[13] Bradtke,S.J., and Barto,A.G. (1996) Linear least-squares algorithms for temporal differemce learning. *Machine Learning*, 22:33-57.

[14] Bradtke,S.J., Ydestie,B.E., and Barto,A.G. (1994) Adaptive linear quadratic control using policy iteration. In *Proceedings of the American Control Conference*, pp. 3475-3479. American Automatic Control Council, Evanston, IL.

[15] Dayan, P.(1992). The convergence of TD(λ) for general λ. *Machine Learning*, 8:341-362.

[16] Hinton,G.E.(1984). Distributed representations. Technical Report CMU-CS-84-157. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

[17] Holland,J.H. (1975). *Adaptation in Natural Artificial Systems*. University of Michigan Press, Ann Arbor

[18] Holland,J.H. (1976). Adaptation. *Progress in Theoretical Biology*, vol. 4, pp. 263-293. Academic Press, New York.

[19] Holland,J.H. (1986). Escaping brittleness: The possibility of general-purpose learning algorithms applied to rule-based systems. *Maschine Learning: An Artifical Intelligence Approach*, vol. 2., pp. 593-623. Morgan Kaufmann, San Mateo, CA.

[20] Howard,R. (1960). *Dynamic Programming and Markov Process*. MIT Press, Cambridge, MA.

[21] Jaakkola,T., Jordan,M.I., and Singh,S.P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185-1201.

[22] Klopf, A.H. (1972) Brain function and adaptive systems — A heterostatic theory. Technical report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA.

[23] Kumar, V., and Kanal, L.N. (1988). The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. *Search in Artificial Intelligence*, pp 1-37. Springer-Verlag, Berlin

[24] Minsky,M.L. (1967). *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, NJ

[25] Oliver,M. (1996) Khepera Simulator version 2.0 User manual

[26] Olshausen,B. A., Field,D.J. (1996) *Natural image statistics and efficient coding.* Presented at the Workshop on Information Theory and the Brain , September 4-5, 1995, University of Stirling, Scotland.

[27] Puterman,M.L., and Shin,M.C. (1978). Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24:1127-1137.

[28] Ross,S. (1983). *Introducion to Stochastic Dynamic Programming.* Academic Press, New york.

[29] Rummery, G.A., and Niranjan,M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166. Engineering Depatrment, Cambridge University.

[30] Samuel,A.L. (1959). Some studies in machine learning using the game of checkers. *IBM journal on Research and Development*, 3:211-229.

[31] Schultz,D.G., and Melsa,J.L. (1967). *State Functions and Linear Control Systems.* McGrawHill, new York.

[32] Singh,S.P., and Suton,R.S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123-158

[33] Sutton,R.S. (1978a) Learning theory support for a single channel theory of the brain. Unpublished report.

[34] Sutton,R.S. (1978b) Single channel theory: A neuronal theory of learning. *Brain Theory Newsletter*, 4:72-75. Center for System Neuroscience, University of Massachusetts, Amherts, MA.

[35] Sutton,R.S. (1978c) A unified theory of expectation in classical and instrumental conditioning. Bachelors thesis, Stanford University.

[36] Sutton,R.S. (1984) *temporal Credit Assignment in Reinforcement Learning.* Ph.D. thesis, University of Massachusetts, Amherts.

[37] Sutton,R.S. (1988) Learning to predict by the method of temporal differences. *Machine Learning*, 3:9-44.

[38] Sutton,R.S., and Barto,A.G. (1981a) Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Rewiew*, 88:135-170

[39] Sutton,R.S., and Barto,A.G. (1998) *Reinforcement Learning* MIT Pess, Cambridge, MA

[40] Waltz,M.D., and Fu,K.S. (1965). A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10:390-398.

[41] Watkins,C.J.C.H. (1989). *Learning from Delayed Rewards.* Ph.D. thesis, Cambridge University.

[42] Werbos,P.J. (1990). Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, 3:179-189.

[43] White,D.J. (1969). *Dynamic Programming.* Holden-Day, San Francisco.

[44] Whittle,P. (1982). *Optimization over Time*, vol. 1. Wiley, New York.

[45] Whittle,P. (1983). *Optimization over Time*, vol. 2. Wiley, New York.