Diplomamunka

Dimenzióredukciós mesterséges neuronháló és megerősítéses tanulás elvén működő tanulórendszer elemzése

Nagy Tamás

Témavezető: dr. habil. **Lőrincz András**
tudományos főmunkatárs

Eötvös Loránd Tudományegyetem
Természettudományi Kar, programtervező–matematikus szak

Budapest, 1999. június 21.

Tartalomjegyzék

1. fejezet

Bevezetés

A megerősítéses tanulás és a mesterséges neuronhálózatok összekapcsolásának egyik leghíresebb és legtöbbet emlegetett példája Gerald Tesauro backgammon[1](magyar neve: ostábla) 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 állapot-érték függvény közelítésére.

A backgammon játék szabályai bonyolultak és a játéknak van egy nagyon erős sztochasztikus tulajdonsága is. A 30 báb és a 24 lehetséges pozíció miatt a lehetséges játékállások száma gigantikus: 10^{20} állapot. Ennyi állást különkülön táblázatban tárolni fizikailag is képtelenség. A kockadobás átlagosan 20 lehetséges lépést eredményez, ami gátat emel a játékfa-kiértékelő heurisztikus keresések komolyabb mélységben való alkalmazásának. A játékban nincs igazi szerepe a tervezésnek, a mesteri szinten játszó emberi játékosoknál is a mintafelismerő képesség és az ismert minták száma kiemelkedő.

A TD-Gammon egy standard többrétegű perceptront használt az állapot-érték függvény becslésére. A hálózat bemenete a „nyers” állás volt, rejtett réteg 40, kimeneti réteg egyetlen neuront tartalmazott. A hálózat kimenetén megjelenő értéket úgy értelmezték, mint a játszma adott állásból való megnyerésének a valószínűsége. Ennek megfelelően a közvetlen jutalom a játszma során mindig 0 volt, kivétel ez alól a győztes lépésé, amikor 1.

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

[1]A backgammon játék rövid ismertetője a Függelékben megtalálható

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 a Tesauro által alkalmazott sikeres konstrukció vizsgálata egy eltérő karakterisztikájú környezetben. Szintén egy játékról, az Othello[2]-ról (ismert még pl. Reversi néven is) van szó. Az alapvető különbség a két játék között az, hogy az Othello-ban nincs szerepe a véletlennek. Az Othello további jellemzője, hogy viszonylag jól működnek rajta a különböző játékfa-kiértékelő heurisztikus keresések (bár a lehetséges lépések száma átlagosan 10-12.) Másik lényeges tulajdonsága, hogy egyetlen lépés is jelentősen meg tudja változtatni az állást. Ez a tulajdonság a kiértékelő függvény bonyolultabb reprezentációját igényli.

A dolgozat első fejezetében áttekintjük a megerősítéses tanulás alapfogalmait és algoritmusait. A második fejezeben a függvényapproximátorként alkalmazott többrétegű perceptronról, egy speciális neurális hálózatról adunk rövid ismertetőt. A harmadik fejezetben a teljes információs kétszemélyes játékok játékfájának minimax kiértékeléséről lesz szó.

Ezen ismeretekre alapozva a negyedik fejezetben bemutatjuk az Othello-problémán alkalmazott tanítási módszereket illetve a futtatások eredményeit. Megpróbálunk következtetéseket levonni a tesztek alapján és vázolunk néhány általunk érdekesnek gondolt kutatási irányt.

[2]Az Othello játék ismertetése is megtalálható a Függelékben

1.1. 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 $\mathcal{Q}_t$-hez tartozó paramétervektor
$\vec{\phi}_s$	s állapotot reprezentáló vektor
δ_t	átmeneti–differencia hibaértéke a t-edik időpillanatban
$e_t(s)$	"emlékeztő nyom" (eligiblility trace) s állapotban t-edik idő pillanatban
$e_t(s,a)$	emlékeztető 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
$F\colon \mathbb{R}^{\text{l}} \mapsto (-1,1)^s$	az approximálandó függvény
$\hat{F}\colon \mathbb{R}^{\text{l}} \mapsto (-1,1)^s$	a hálózat által megvalósított leképezés
$p \in \mathbb{N}$	a hálózat bemenő vektorának mérete (*pattern*)
$n^h \in \mathbb{N}$	a hálózat rejtett rétegének a mérete (*hidden layer*)
$s = n^o \in \mathbb{N}$	a hálózat kimenetének a mérete (*output layer*)
$w^h_{ij} \in \mathbb{R}$	a rejtett réteg j-dik neuronjának az i-dik súlya (*weight*), $(i \in \{1,\ldots,p\}, j \in \{1,\ldots,n^h\})$
$w^o_{ij} \in \mathbb{R}$	a kimeneti réteg j-dik neuronjának az i-dik súlya, $(i \in \{1,\ldots,n^h\}, j \in \{1,\ldots,s\})$
y^h_j	a rejtett réteg j-dik neuronjának a kimenete, $j \in \{1,\ldots,n^h\}$
y^o_j	a kimeneti réteg j-dik neuronjának a kimenete, $j \in \{1,\ldots,n^o\}$
$l^h_j := \sum_{i=1}^{p} w^h_{ij}\vec{x}_i$	a rejtett réteg j-dik neuronjának lineáris aktivitása $\vec{x}$ bemenő vektorra
$l^o_j := \sum_{i=1}^{n^h} w^o_{ij} y^h_i$	a kimeneti réteg j-dik neuronjának lineáris aktivitása

2. fejezet

Megerősítéses tanulás

2.1. A feladat

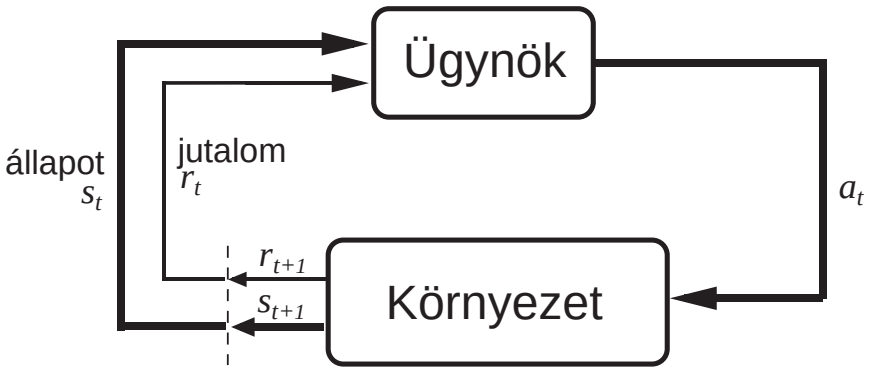
A természetes tanulás folyamatán gondolkozva az első dolog ami eszünkbe juthat az, hogy tudásunk legfőbb forrása a *környezet*ünkkel való kapcsolat. Ekkor valójában nincs világosan megfogalmazható tanító, – aki megmondja, hogy mit kell tenni, – azonban a közvetlen szenzoros kapcsolat alapján megtanulhatóak 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)*.

2.1.1. 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.



2.1. 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 *politikájának (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.

2.1.2. 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.

2.1.3. 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}, ... a t időpillanat utáni jutalmakat, ekkor a hozam a legegyszerűbb formában:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T \ , \tag{2.1}$$

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 (2.1) 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 :

$$R_t = r_{t+1} + \gamma\, r_{t+2} + \gamma\, r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \ , \tag{2.2}$$

ahol γ $(0 \leq \gamma < 1)$ a diszkontálási paraméter.
Ha $\gamma < 1$, akkor (2.2) 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:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \, . \tag{2.3}$$

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.

2.1.4. 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-tulajdonság*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:

$$Pr\{\, s_{t+1} = s',\, r_{t+1} = r \mid s_t,\, a_t,\, r_t,\, s_{t-1},\, a_{t-1},\, \ldots,\, r_1,\, s_0,\, a_0 \,\}, \tag{2.4}$$

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:

$$Pr\{s_{t+1} = s',\, r_{t+1} = r \mid s_t\,, a_t\,\}\,, \tag{2.5}$$

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:

$$\mathcal{P}^a_{ss'} = Pr\{\, s_{t+1} = s' \mid s_t = s,\, a_t = a \,\}. \tag{2.6}$$

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

$$\mathcal{R}^a_{ss'} = E\{ r_{t+1} = s' \mid s_t = s,\ a_t = a,\ s_{t+1} = s' \}. \tag{2.7}$$

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

2.1.5. 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 2.1.1 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:

$$V^\pi(s) \;=\; E_\pi\{ R_t \mid s_t = s \} \;=\; E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\}, \tag{2.8}$$

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:

$$\mathcal{Q}^{\pi}(s,a) = E_{\pi}\{R_t \,|s_t = s,\, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s,\, a_t = a\right\}, \tag{2.9}$$

$\mathcal{Q}^{\pi}$-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:

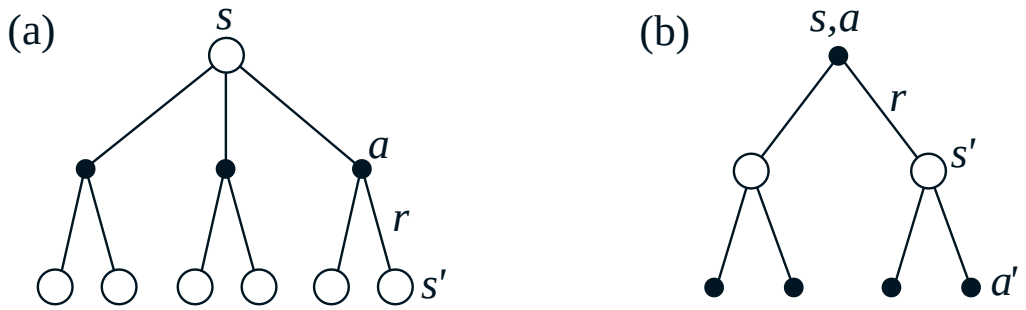
$$\begin{aligned}
V^{\pi}(s) &= E_{\pi}\{\, R_t \mid s_t = s \,\} \\
&= E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\} \\
&= E_{\pi}\left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s \right\} \\
&= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma\, E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_{t+1} = s' \right\} \right] \\
&= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma\, V^{\pi}(s') \right] ,
\end{aligned} \tag{2.10}$$

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 (2.10) 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 2.2 ábra a rekurzív kapcsolat szemléltetését szolgálja, ahol a jelölés a következő : üres körök az állapotok, teli körök pedig az állapot-akció párok.

2.1.6. 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

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



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

$s(\in \mathcal{S})$ 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 politika (optimal policy)*. Az optimális politika, vagy politikák jelölése: π^*. Az optimális politikák segítségével megadható az *optimális állapotot értékelő függvény* (V^*) definíciója:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \tag{2.11}$$

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

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

$$\mathcal{Q}^*(s,a) = \max_{\pi} \mathcal{Q}^{\pi}(s,a), \tag{2.12}$$

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

$$\mathcal{Q}^*(s,a) = E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \,|s_t = s, a_t = a\right\}. \tag{2.13}$$

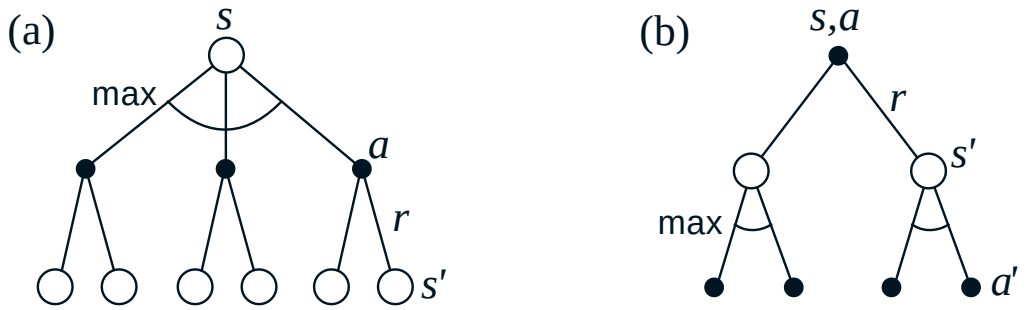
Mivel V^* értékelő függvény, ezért ki kell elégítenie a Bellman egyenlet által kirótt feltételt. Az optimális értékelő függvényhez tartozó Bellman egyenletet *Bellman optimalitási egyenlet*nek nevezik. A Bellman optimalitási egyenlet valójában az állapot értékét adja meg optimális politika esetén, amely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

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} \mathcal{Q}^{\pi^*}(s,a) \\
&= \max_a E_{\pi^*} \{ R_t \,|s_t = s, a_t = a \} \\
&= \max_a E_{\pi^*} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s, a_t = a \right\} \\
&= \max_a E_{\pi^*} \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,|s_t = s, a_t = a \right\} \\
&= \max_a E_{\pi} \{ \, r_{t+1} + \gamma \, V^*(s') \,|s_t = s, a_t = a \} \qquad (2.14) \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \, V^*(s') \right]. \qquad (2.15)
\end{aligned}
$$

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

$$
\begin{aligned}
\mathcal{Q}^*(s,a) &= E \left\{ r_{t+1} + \gamma \max_{a'} \mathcal{Q}^*(s_{t+1}, a') \,|s_t = s, a_t = a \right\} \qquad (2.16) \\
&= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} \mathcal{Q}^*(s', a') \right]. \qquad (2.17)
\end{aligned}
$$

A 2.3 ábra grafikusan szemlélteti a (2.15) alatt megadott Bellman optimalitási egyenletet.



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

A véges Markov döntési folyamatoknál a Bellman optimalitási egyenletnek (2.15) 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.

2.2. Alapvető megoldási módszerek

2.2.1. 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 (2.15) (2.17). 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,

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \,|\, s_t = s \right\} \\ &= E_\pi\left\{ r_{t+1} + \gamma V^\pi(s_{t+1} \,|\, s_t = s\right\} \qquad (2.18) \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \right] , \qquad (2.19) \end{aligned}$$

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 (2.19) 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 (2.10) felülírási szabályként alkalmazva:

$$\begin{aligned} V_{k+1}(s) &= E_\pi \{ r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s \} \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi_k(s')] , \end{aligned} \tag{2.20}$$

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: $V(s) = 0 \forall s \in \mathcal{S}^+$
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)
Output: $V \approx V^\pi$

2.1. 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 2.1 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 választott $a \neq \pi(s)$ 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:

$$\begin{aligned} \mathcal{Q}^\pi(s,a) &= E_\pi\{\, r_{t+1} + \gamma V_k(s_{t+1}) \,|\, s_t = s,\, a_t = a\} \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \,[\, \mathcal{R}^a_{ss'} + \gamma V^\pi_k(s') \,]\,. \end{aligned} \tag{2.21}$$

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:

$$\mathcal{Q}^\pi(s, \pi'(s)) \geq V^\pi(s). \tag{2.22}$$

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:

$$V^{\pi'}(s) \geq V^\pi(s). \tag{2.23}$$

Ha a szigorúan nagyobb reláció fennáll néhány állapotra a (2.22) egyenletnél akkor legalább egy állapotra fennáll a szigorúan nagyobb reláció a (2.23) 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 (2.22) egyenletből:

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

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:

$$
\begin{aligned}
\pi'(s) &= \arg\max_a \mathcal{Q}^{\pi}(s,a) \\
&= \arg\max_a E\{\, r_{t+1} + \gamma\, V^{\pi}(s_{t+1}) \,|\, s_t = s,\, a_t = a\,\} \qquad (2.24) \\
&= \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\, \mathcal{R}^a_{ss'} + \gamma\, V^{\pi}_k(s') \,\right] ,
\end{aligned}
$$

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 (2.10) definíció alapján:

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

Ez az egyenlet megegyezik a Bellman optimailtási egyenlettel (2.15), 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:

$$\mathcal{Q}^{\pi}(s, \pi'(s)) = \sum_{a} \pi'(s,a)\, \mathcal{Q}^{\pi}(s,a). \tag{2.25}$$

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:

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

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 2.2 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 kelle 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:

$$\begin{aligned} V_{k+1}(s) &= \max_a E\{\, r_{t+1} + \gamma\, V_k(s_{t+1}) \,|\, s_t = s,\, a_t = a \,\} \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\, \mathcal{R}^a_{ss'} + \gamma\, V_k(s') \,\right], \end{aligned} \tag{2.26}$$

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 $s \in \mathcal{S}$
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   Until $\Delta < \theta$ (small positive number)

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

2.2. táblázat. Politika iterálása

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 2.3 ábra mutatja a teljes érték iteráció algoritmust, az előző megállási feltétellel. Az algoritmus – diszkontált véges Markov folyamatok esetén – az optimális politikát adja meg.

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 algo-

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

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

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

2.3. táblázat. Érték iteráció.

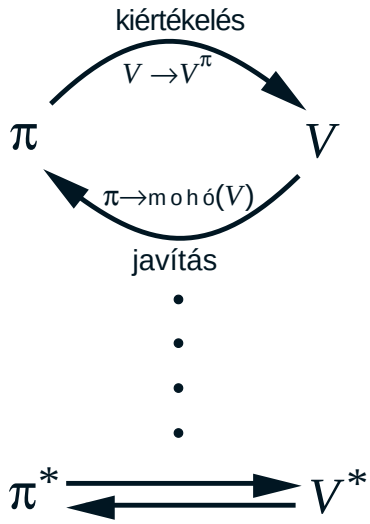
ritmusok. Egy aszinkron algoritmus tetszőleges sorrendben használja fel az elérhető állapotok értékét, amely nagy rugalmasságot biztosít. A folytonosság biztosítja a konvergenciát, azaz az algoritmus az optimális politika felé konvergál.

Általánosított politika iteráció

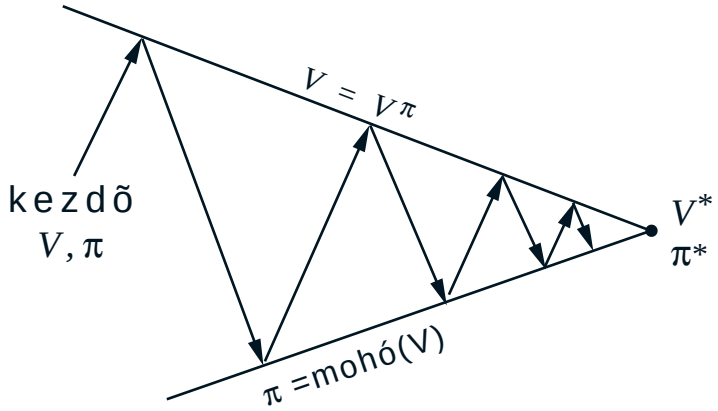
A politika iteráció két egyszerre zajló, egymással kapcsolatban lévő folyamatból épül fel: az egyik az adott politikához megadja az értékelő függvényt (politika kiértékelés), a másik, amely mohó módon új politikát hoz létre az értékelő függvény alapján (politika javítás). A politika iterációban ez a 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 2.4 ábra szemlélteti a GPI-t.

Ennek a módszernek egy másik szemléltető ábrája a 2.5 ábra, ahol a két folyamatot két vonal ábrázolja kétdimenziós térben.



2.4. ábra. Általánosított politika iteráció.



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

2.2.2. 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:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]. \tag{2.27}$$

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 2.4 táblázat a TD módszert szemlélteti, a 2.6 ábra pedig a TD algoritmust

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'
    $V(s_t) \leftarrow V(s_t) + \alpha \left[r + \gamma V(s_{t+1} - V(s_t)) \right]$
    $s \leftarrow s'$
  until s is terminal

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

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

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 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

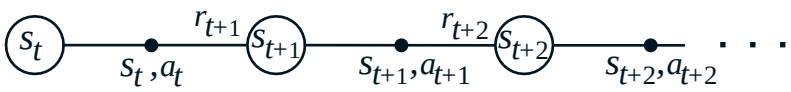
Tegyük fel, hogy adott véges számú tapasztalat, 10 epizód vagy 100 lépés. Ebben az esetben az átlagos közelítési módszer addig ismételgeti a tapasztalatokat, amíg a módszer nem konvergál a válaszhoz. Adott V, az értékelő függvény közelítése, amelyet javítani szeretnénk. A (2.27) egyenlet alapján minden 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.



2.7. 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:

$$\mathcal{Q}(s_t, a_t) \leftarrow \mathcal{Q}(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \, \mathcal{Q}(s_{t+1}, a_{t+1}) \; - \; \mathcal{Q}(s_t, a_t) \right]. \qquad (2.28)$$

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.

Mint a legtöbb aktív politizálási módszernél, folyamatosan közelítjük $\mathcal{Q}^\pi$ értékét, és vele egyidőben π – $\mathcal{Q}^\pi$-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.

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

Initialize: $\mathcal{Q}(s,a)$ 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'
    $\mathcal{Q}(s,a) \leftarrow \mathcal{Q}(s,a) + \alpha\,[r + \gamma\,\mathcal{Q}(s',a') - \mathcal{Q}(s,a))]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

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

2.2.3. 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.

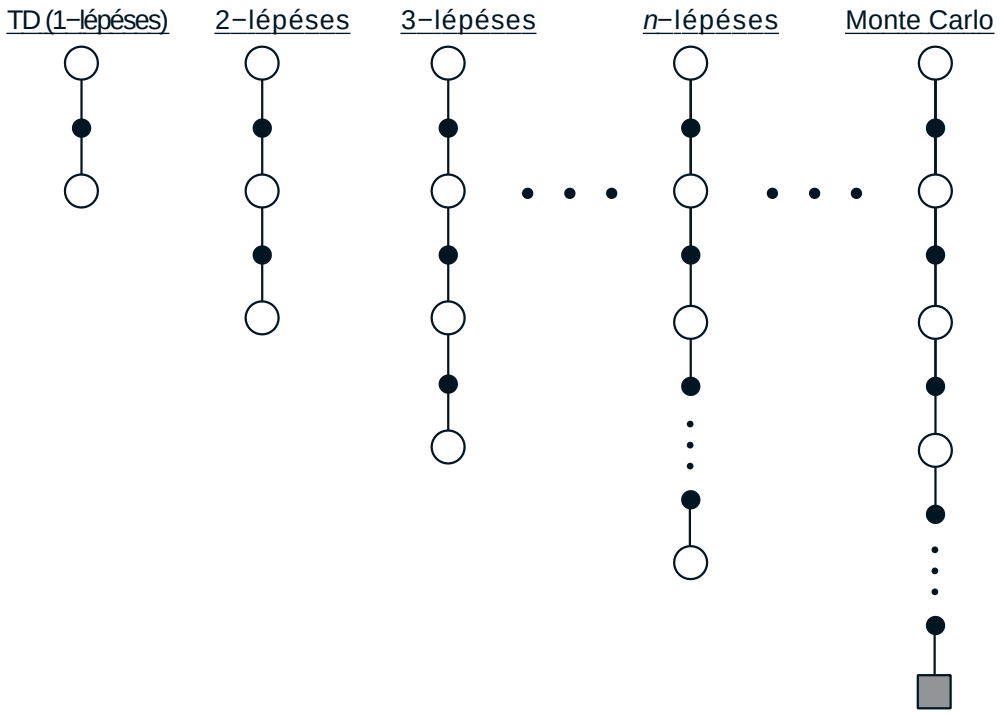
A másik szemlélet sokkal gyakorlatiasabb, ez az úgynevezett *visszafele tekintés (backward view)* módszere. Ebből a szemszögből az emlékeztetető nyomok nem mások, mint a bekövetkezett események (amely állapotot, vagy akciót jelenthet) átmeneti feljegyzése. A nyom megjegyzi a memória paramétereket, amelyek 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.

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,

[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.

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.



2.8. 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.

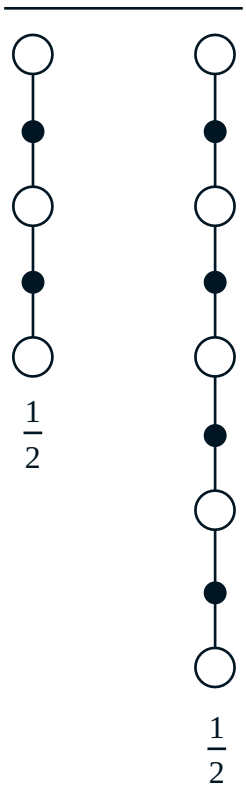
Az n-lépéses várható hozam V segítségével történő meghatározása biztosítja a V^π V-nél jobb közelítését a legrosszabb esetben is. Ez azt jelenti, hogy a legnagyobb hiba az új közelítésnél kisebb, vagy egyenlő, mint γ^n-szer a legnagyobb hiba V értékelő függvény esetén:

$$\max_s \left| E_\pi \left\{ R_t^{(n)} \mid s_t = s \right\} - V^\pi(s) \right| \leq \gamma^n \max_s |V(s) - V^\pi(s)| . \qquad (2.29)$$

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 négy-lépéses hozamok átlagával határozzuk meg a felösszegzési gráf értékeket. A hasonló módon előállított értékeket *komplex felösszegzési gráf értéknek* nevezzük (lásd 2.9 ábra).

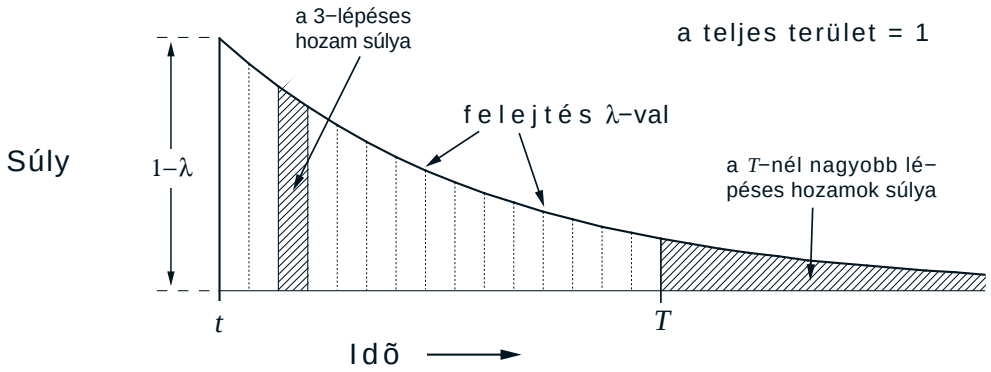


2.9. ábra. Komplex felösszegzési gráf

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 (2.11 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:

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



2.10. ábra. λ-súlyozás

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:

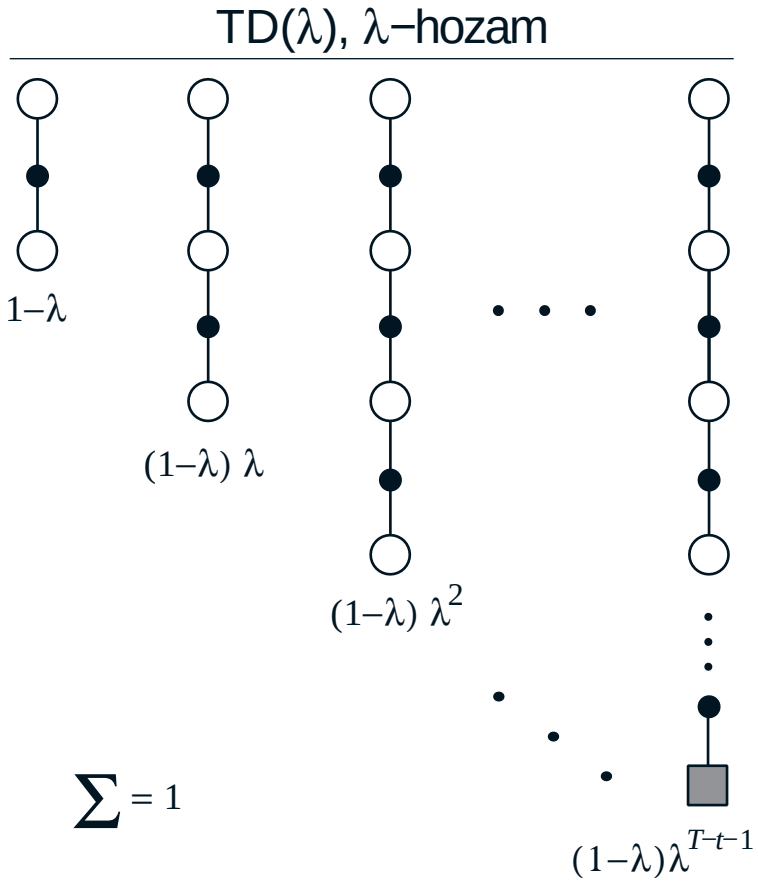
$$R_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \gamma^{T-t-1} R_t. \tag{2.30}$$

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.

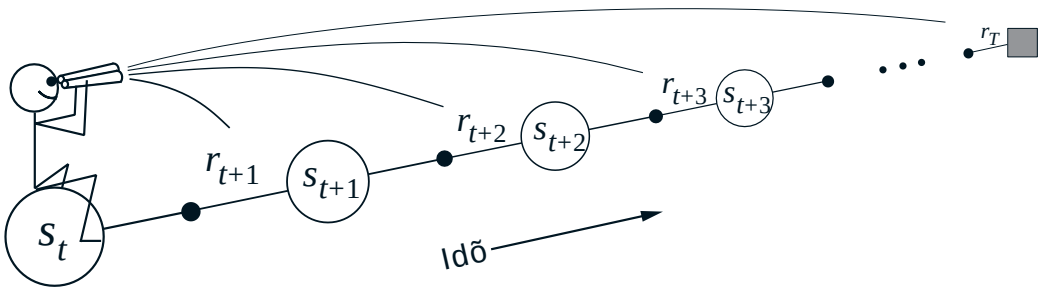
Definiáljuk úgy a λ-*hozam algoritmust*, mint egy olyan algoritmust, amely a λ-hozam segítségével meghatározza a felülírási gráf értékét. Minden egyes t lépésben a $\Delta V_t(s_t)$ változtatás mértékét megadja az algoritmus a következő formában:

$$\Delta V_t(s_t) = \alpha \left[R_t^\lambda - V_t(s_t) \right]. \tag{2.31}$$

($\Delta V(s_t) = 0$ 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 2.12 ábra).



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



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

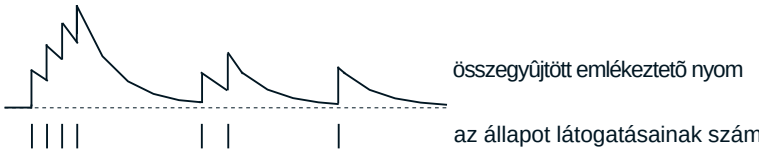
A TD(λ), mint visszatekintő módszer

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:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) \text{ ha } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 \text{ ha } s = s_t \end{cases} \tag{2.32}$$

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.



2.13. 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:

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)\,. \tag{2.33}$$

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

$$\Delta V_t(s) = \alpha\delta_t e_t(s)\,, \quad \forall s \in \mathcal{S} \tag{2.34}$$

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 (2.32-2.34) egyenletek megadják a TD(λ) algoritmus definícióját. A 2.6 táblázat tartalmazza a teljes on-line TD(λ) algoritmust. Minden egyes időpillanatban megnézzük az aktuális TD hibát és

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

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

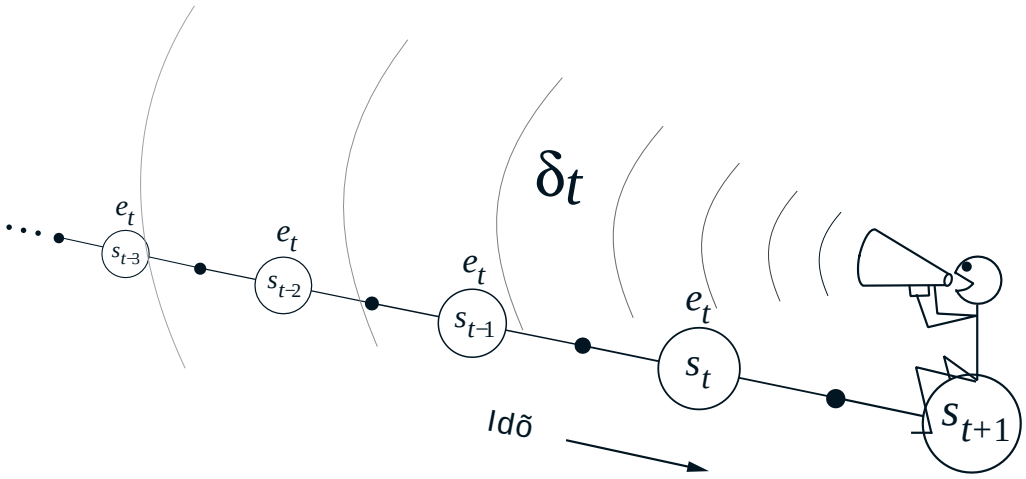
megállapítjuk a felülírási értéket az összes olyan állapotra, amelynek nyoma van az adott pillanatban. Ezt szemléletesen a 2.14 ábra mutatja be.

Vizsgáljuk meg a paraméterek lehetséges értékénél hogyan viselkedik a TD(λ) módszer. Ha $\lambda = 0$, akkor a TD(λ) (2.34) felülírási szabály a TD(0) módszer felülírási szabályára (2.27) 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.

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 (2.31) esetén, és $\Delta V_t^{TD}(s)$ pedig a t-edik időpillanatbeli s érték felülírást a mehanikusan definiált TD(λ) esetén (2.34). A célunk az, hogy megmutassuk a felülírások összege az epizód végén mindkét algoritmus esetében megegyezik:

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



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

$$\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) = \sum_{t=0}^{T-1} \Delta V_t^{\lambda}(s_t)\, \mathcal{I}_{ss_t}\,, \quad \forall s \in \mathcal{S}\,, \tag{2.35}$$

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:

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

Eképpen a (2.35) egyenlet bal oldala a következő formában írható le:

$$\begin{aligned}
\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) &= \sum_{t=0}^{T-1} \alpha\delta_t \sum_{k=0}^{t} (\gamma\lambda)^{t-k}\, \mathcal{I}_{ss_k} \\
&= \sum_{k=0}^{T-1} \alpha\delta_k \sum_{t=0}^{k} (\gamma\lambda)^{k-t}\, \mathcal{I}_{ss_t} \\
&= \sum_{k=t}^{T-1} \alpha\delta_k \sum_{t=0}^{T-1} (\gamma\lambda)^{k-t}\, \mathcal{I}_{ss_t} \\
&= \sum_{t=0}^{T-1} \alpha\mathcal{I}_{ss_t} \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t}\, \delta_k \,. \qquad (2.36)
\end{aligned}$$

Most nézzük a (2.35) egyenlet jobb oldalát. Alkalmazzuk a λ-hozam algoritmus sajátos felülírását:

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

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 (2.35) 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.

2.3. Kapcsolat a függvény-approximátorokkal

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 amelyekkel 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.

2.3.1. 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 ebben az esetben 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. Például DP esetén az érték mentés $s_t \mapsto E_\pi\{r_{t+1} + \gamma V_t(s_{t+1}) | s_t = s\}$, TD(0) esetén $s \mapsto r_{t+1} + \gamma V_t(s_{t+1})$ és TD(λ) esetén $s \mapsto R_t^\lambda$ alakul.

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. Így 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 *négyzetes hiba* (Mean-Square Error MSE) minimalizálására törekszik az állapotok egy bizonyos eloszlásával.

$$MSE(\vec{\theta_t}) = \sum_{s \in S} P(s)[V^\pi(s) - V_t(s)]^2, \tag{2.37}$$

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.

2.3.2. Gradiens keresési eljárás

A gradiens keresési eljárás egy, a függvény approximátorok terén széleskörben elterjedt eljárás, amely 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 minimalizálását a négyzetes hiba függvény $\vec{\theta_t}$ szerinti gradiensével ellentétes irányba haladva érhetjük el, azaz:

$$\begin{aligned} \vec{\theta}_{t+1} &= \vec{\theta_t} - \frac{1}{2}\alpha \nabla_{\vec{\theta_t}}[V^\pi(s_t) - V_t(s_t)]^2 \\ &= \vec{\theta_t} + \alpha[V^\pi(s_t) - V_t(s_t)]\nabla_{\vec{\theta_t}} V_t(\vec{s_t}) \end{aligned} \tag{2.38}$$

ahol α a lépésköz paramétere és $\nabla_{\vec{\theta_t}} f(\vec{\theta_t})$ az f függvény $\vec{\theta_t}$ paramétervektor szerinti gradiense. Belátható, hogy a 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, feltéve, hogy teljesül a standard sztochasztikus feltétel, azaz a nullához való konvergencia nem túl gyors. 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 helyettesítünk 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 *torzítatlan 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:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha[R_t^\lambda - V_t(s_t)]\nabla_{\vec{\theta}_t} V(s_t) \tag{2.39}$$

Sajnos $\lambda < 1$-re R_t^λ nem torzítatlan becslése a $V^\pi(s_t)$-nek, és így nem feltétlenül 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 csak az epizód után tudjuk 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:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\delta_t\vec{e}_t \tag{2.40a}$$

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t), s \tag{2.40b}$$

$$\vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \nabla_{\vec{\theta}_t} V_t(s_t). \tag{2.40c}$$

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.

2.3.3. Lineáris eljárás

Az egyik legfontosabb eljárás a lineáris eljárás, amelyben V_t lineáris függvénye a $\vec{\theta}_t$ paramétervektornak.

$$V_t = \vec{\theta}_t^T \vec{\phi}_{s_t} \tag{2.41}$$

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

Az előző fejezetben tárgyalt $TD(\lambda)$ eljárás is konvergál abban az esetben, ha a lépésköz-paramétere csökken a lépésekkel. Ekkor természetesen nem a globális optimumhoz fog konvergálni, hanem egy olyan $\vec{\theta}_\infty$ paramétervektorhoz, 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. A lineáris eljárás nem feltétlenül függ kritikusan attól, hogy hogyan alakítjuk ki az állapotokból a tulajdonságvektort.

3. fejezet

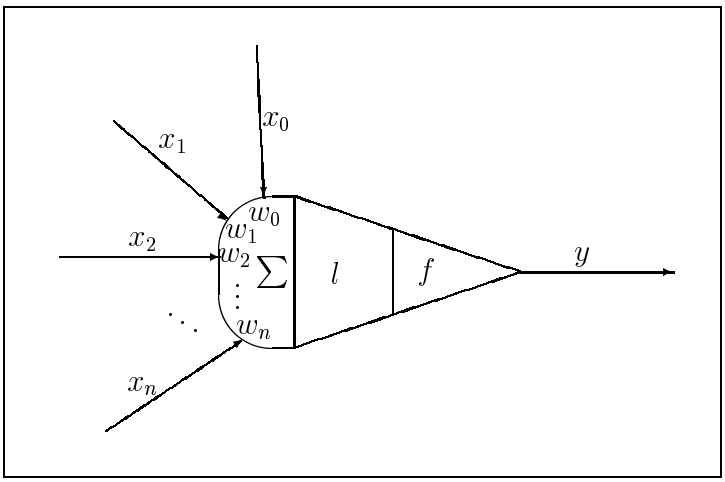
Az többrétegű perceptron

Bármennyire is misztikusnak tűnik a *mesterséges neuronhálózat* kifejezés, egy többrétegű perceptron, mint látni fogjuk, egyszerűen egy $\mathbb{R}^p \mapsto \mathbb{R}^s$ típusú vektor-vektor leképezést valósít meg, ahol $p \in \mathbb{N}$ a bemenő minta, $s \in \mathbb{N}$ pedig a hálózat kimenetének a mérete. A többrétegű perceptron tanítása egy matematikailag jól kezelhető függvényapproximációs feladat.

3.1. Mesterséges neuron

Egy mesterséges neuron egy önálló számítási egység: a beérkező jeleket feldolgozva generálja a kimenetét. Minden, a neuronba vezető csatornához, amin valamilyen jel érkezhet a neuronhoz, hozzá van rendelve egy valós érték (*súly*), amit értelmezhetünk úgy is, mint a kapcsolat erőssége a jel forrása és a neuron között. A jel forrás lehet egy másik neuron, de a környezettől is származhat az információ. Általában a feldolgozás, amit a neuron végez bemenő jeleken, annyit jelent, hogy a kapcsolatok erősségével képzett súlyozott összegre alkalmazunk még egy nemlineáris transzformációt. Ezt a nemlineáris transzformációt hívják aktiválási függvénynek. Az aktiválási függvény általában korlátos értékű, így a neuron által megvalósított legképezés is az lesz. A 3.1 ábrán egy mesterséges neuron sematikus modellje látható.

Formalizálva:



3.1. ábra. A mesterséges neuron

$w_0, w_1, \ldots, w_n \in \mathbb{R}$	a neuron bemenő kapcsolataihoz rendelt súlyok
$x_0, x_1, \ldots, x_n \in \mathbb{R}$	a neuron bemenő kapcsolatain aktuálisan megjelent értékek
l	a neuron lineáris aktivitása. $l = \sum_{i=0}^{n} w_i x_i$
f	a neuron aktiválási függvénye. Az általánosan használt aktiválási függvények a 3.2 ábrán láthatóak.
y	a neuron aktuális kimenete. $y = f(l)$.

Az általánosan használt aktivizációs függvények a 3.2 ábrán láthatóak. Képletben:

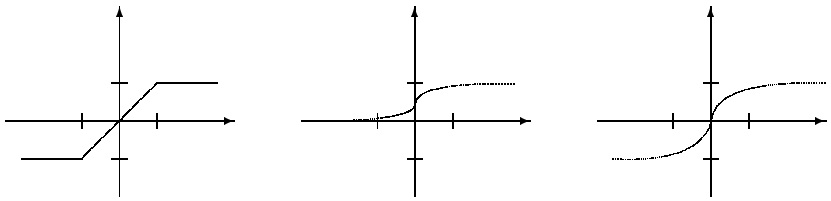
$$f(x) = x \tag{3.1}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

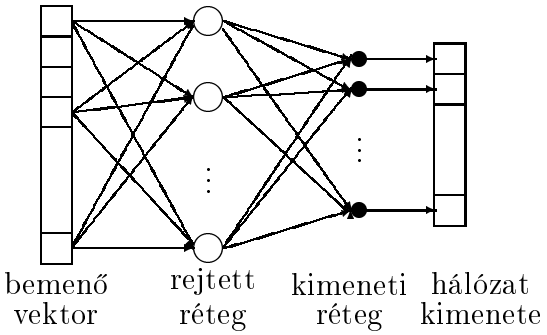
$$f(x) = \mathrm{th}x \tag{3.3}$$

3.2. Többrétegű perceptron szerkezete

A többrétegű perceptron egyike a legegyszerűbb mesterséges neuronhálózatoknak, amelyek a fent ismertett mesterséges neuronokból felépíthetőek. A



3.2. ábra. A szigmoid-szerű aktiválási függvények



3.3. ábra. A többrétegű perceptron

többrétegű kifejezés általában egészen pontosan két réteget jelent, ettől nagyobb hálózatokat ritkán alkalmaznak. Ezért a jelen tanulmány keretein belül csak azt a speciális esetet vizsgáljuk, amikor pontosan kettő rétegbe vannak szervezve a neuronok. Az első réteget rejtett rétegnek, a másodikat kimeneti rétegnek nevezik. A neuronok közötti kapcsolat teljesen definiált: azonos réteghez tartozó neuronok között nincs összeköttetés, a rejtett réteg neuronjai minden bemenő értékkel kapcsolatban vannak, éppen úgy, mint a kimeneti réteg neuronjai a rejtett réteg neuronjaival. Ha a bemenő értékek vektorát szintén egy rétegnek tekintenénk, akkor egyszerűbben azt mondhatnánk, hogy a hálózatban minden két, egymás közvetlenül követő réteg teljes összeköttetésben van, és más kapcsolat nincs a neuronok közt[1]. Az 3.3 ábrából minden világossá válik.

[1]A neuronhálózatok elméletében nincs egységes nézet, hogy mit tekintenek rétegnek. Vannak, akik neuronok rétegéről beszélnek, vannak akik súlyok rétegéről. Emellett vannak, akik a bemenő vektort speciális neuronoknak tekintik, a 0-dik réteg neuronjainak. Mi csak a tényleges neuronokat hívjuk neuronoknak.

3.3. A tanulás

A többrétegű perceptron tanítása egy, a sztochasztikus gradiens módszeren alapuló függvényapproximáció. Az approximálandó függvényt, F-et valahány $\vec{x} \in \mathbb{R}^p$ pontban ismerjük, vagyis adott egy $P = \{(\vec{x}^{(i)}, \vec{d}^{(i)}) \mid x^{(i)} \in \mathbb{R}^p, d^{(i)} \in \mathbb{R}^s, F(\vec{x}^{(i)}) = \vec{d}^{(i)}, i = 1, 2, \ldots\}$[2] halmaz. Azt szeretnénk elérni, hogy a hálózat által kiszámított $\hat{F}(\vec{x}^{(i)})$ értékek egyenletesen közel legyenek a tényleges $F(\vec{x}^{(i)})$ értékekhez.

A hálózat hibája az $(\vec{x}, \vec{d})$ páron az alábbi módon van definiálva:

$$E = \frac{1}{2}\sum_{j=1}^{s}(\vec{d_j} - \hat{F}(\vec{x})_j)^2 = \frac{1}{2}\sum_{j=1}^{s}(\vec{d_j} - f(l_j^o))^2. \tag{3.4}$$

A sztochasztikus gradiens módszer értelmében ezt a hibát próbáljuk iteratív módon csökkenti minden egyes $(\vec{x}, \vec{d}) := (\vec{x}^{(i)}, \vec{d}^{(i)})$ párra, vagyis egyszerre egyetlen ponttal foglalkozunk csak. A minták felső indexét az könnyebb átláthatóság érdekében elhagyjuk. Egy lépésben a hálózat w_{ij}^h, w_{ij}^o súlyait hangoljuk: egy adott w súlyt a képződött hiba w szerinti deriváltjával ellentétes irányba mozdítjuk:

$$\triangle w_{ij}^h = -\eta \frac{\partial E}{\partial w_{ij}^h} \tag{3.5}$$

illetve

$$\triangle w_{ij}^o = -\eta \frac{\partial E}{\partial w_{ij}^o}. \tag{3.6}$$

ahol $\eta > 0$ az ún. tanulási ráta vagy lépésméret. Talán jelentéktelennek tűnik, de az η nagyon fontos paramétere a gradiens módszernek. Ha túl nagy, akkor nincs semmi garancia arra, hogy a fenti változtatást végrehajtva és a deriválttal ellenkező irányba lépve csökkenni fog az approximáció hibája. Ha viszont η biztonsági szempontból nagyon kicsi, az rendkívül lelassítja a konvergenciát, mivel nagyon apró lépéseket teszünk csak a cél irányába. A helyzetet tovább bonyolítja, hogy a gradiens módszer elmélete szerint η-t idővel csökkenteni is kellene, vagyis $\eta - t$ közelíteni kell a 0-hoz. Ez újabb problémát vet fel: mikor kezdjük el csökkenteni? Az η paraméter helyes megválasztása nem könnyű feladat de néhány heurisztikus megoldást kitaláltak rá. A 2.4.2. szakaszban ismertetjük az egyiket.

Feladatunk tehát a súlyok szerinti deriváltak kiszámítására vezethető vissza.

[2] A P lehet véges is, de előfordulhat, hogy nem áll egyszerre rendelkezésre az összes $(\vec{x}, \vec{d})$ pár, hanem folyamatosan generálódnak ezek az értékek. Ebben az esetben P végtelen.

(a) Kimeneti rétegben levő neuron esetén:

$$\frac{\partial E}{\partial w_{ij}^o} = \frac{\partial E}{\partial l_j^o}\frac{\partial l_j^o}{\partial w_{ij}^o} = -\delta_j^o\, y_i^h. \tag{3.7}$$

ahol $\delta_j^o := -\frac{\partial E}{\partial l_j^o}$. Figyelembe véve, hogy f a kimeneti réteg neuronjainak az aktiválási függvénye:

$$\begin{aligned}\delta_j^o :&= -\frac{\partial E}{\partial l_j^o} = -\frac{\partial \frac{1}{2}\sum_{k=1}^{n^o}(\vec{d}_k - f(l_k^o))^2}{\partial l_j^o} \\ &= -\sum_{k=1}^{n^o}(\vec{d}_k - f(l_k^o))(-\frac{\partial f(l_k^o)}{\partial l_j^o}).\end{aligned} \tag{3.8}$$

Mivel $\frac{\partial f(l_k^o)}{\partial l_j^o} = 0$, ha $j \neq k$, ezért az összegből egyetlen tag marad csak, így:

$$\delta_j^o = (\vec{d}_j - f(l_j^o))\,\frac{\partial f(l_j^o)}{\partial l_j^o}. \tag{3.9}$$

ami a speciális aktiválási függvények választása miatt, továbbá kihasználva, hogy $f(l_j^o) = y_j$, még tovább egyszerűsödik:

1. $f(x) = x$ esetén $\delta_j^o = \vec{d}_j - \vec{y}_j$.
2. $f(x) = \frac{1}{1+e^{-x}}$ esetén, mivel $f' = f\,(1-f)$,
   ezért $\delta_j^o = (\vec{d}_j - \vec{y}_j)\,\vec{y}_j\,(1-\vec{y}_j)$.
3. $f(x) = \mathrm{th}x$ esetén, mivel $f' = f - f^2$,
   ezért $\delta_j^o = (\vec{d}_j - \vec{y}_j)\,(1-\vec{y}_j^2)$.

Mindhárom aktiválási függvény esetében a δ_j^o értéke csak a várt és a tényleges kimenetektől függ.

(b) Rejtett rétegben levő neuron esetén:

$$\frac{\partial E}{\partial w_{ij}^h} = \frac{\partial E}{\partial l_j^h}\frac{\partial l_j^h}{\partial w_{ij}^h} = -\delta_j^h\,\vec{x}_i. \tag{3.10}$$

ahol

$$\begin{aligned}\delta_j^h :&= -\frac{\partial E}{\partial l_j^h} = -\sum_{k=1}^{s}\frac{\partial E}{\partial l_k^o}\frac{\partial l_k^o}{\partial l_j^h} = \sum_{k=1}^{s} -\frac{\partial E}{\partial l_k^o}\frac{\partial l_k^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial l_j^h} = \\ &= \sum_{k=1}^{s}\delta_k^o\, w_{jk}^o\, f'(l_j^h) = f'(l_j^h)\sum_{k=1}^{s}\delta_k^o w_{jk}^o.\end{aligned} \tag{3.11}$$

ami szintén egyszerűsíthető, egyrészt a speciális aktiválási függvények miatt, másrészt pedig, mert l_j^h egyszerűen a hálózat bemenő vektorának a j-edik tagja.

1. $f(x) = x$ esetén $\delta_j^h = \sum_{k=1}^{s} \delta_k^o w_{jk}^o$.
2. $f(x) = \frac{1}{1+e^{-x}}$ esetén, mivel $f' = f\,(1-f)$, ezért $\delta_j^h = y_j^h\,(1-y_j^h) \sum_{k=1}^{s} \delta_k^o w_{jk}^o$.
3. $f(x) = \tanh x$ esetén, mivel $f' = 1 - f^2$, ezért $\delta_j^h = (1-(y_j^h)^2)) \sum_{k=1}^{s} \delta_k^o w_{jk}^o$.

Látható, hogy a rejtett réteg neuronjainak δ-értékei az adott neuron kimenetétől, és a kimeneteti réteg δ-értékeitől függenek.

Ezek után, az egyes w súlyok szerinti deriváltak kiszámítása könnyen algoritmizálható. Első fázisban a rétegeken előrefelé haladva kiszámítjuk a neuronok kimenetét. Második fázisban a kimeneti rétegtől indulva visszafelé kiszámítjuk a δ_j^o, majd a δ_j^h értékeket. Ezután megvan minden szükséges adat a súlyok hangolásához:

$$w_{ij}^h := w_{ij}^h + \triangle w_{ij}^h = w_{ij}^h + \eta\,\delta_j^h\,\vec{x}_i. \tag{3.12}$$

illetve

$$w_{ij}^o := w_{ij}^o + \triangle w_{ij}^o = w_{ij}^o + \eta\,\delta_j^o\,y_i^h. \tag{3.13}$$

Megjegyzés: a δ_j^h, δ_j^o értékeket szokás a megfelelő neuron hibájának is nevezni. Mivel a neuronok hibáját a hálózaton hátrafelé haladva számítjuk, az algoritmust *hiba hátra-terjesztésnek* (error back-propagation, back-propagation) nevezik.

3.4. A tanulás javításai

Sajnos a hiba-hátraterjesztéses algoritmus rendkívül lassú, ezen felül megvan az a kellemetlen tulajdonsága, hogy alkalmanként képtelen az hálózat hibáját(E-t) csökkenteni. Legalább két oka van ennek a gyenge teljesítménynek:

- A deriválttal ellentétes irányba tett lépés növelheti a hibát, ha a lépésméret (η, tanulási ráta) túl nagy
- A derivált általában nem a hibafelület (lokális) minimuma felé mutat, ami az algoritmusban cikk-cakk hatáshoz vezet.

A második probléma könnyebb és legalább részben megoldható. A megfelelő lépésméret megválasztására is ismertetünk egy általánosan használható heurisztikát.

3.4.1. A momentum tag

A cikk-cakk hatást két egymást követő derivált közötti jelentős változás okozhatja: ha gyorsabban szeretnénk haladni a minimum felé, akkor el kell simítani irányeltéréseit. Egyszerű módja ennek, hogy megjegyezzük az utolsó lépés irányát és mozgó átlag eljárással módosítjuk az aktuális lépés irányát.

$$\triangle w_{ij}^{h,\text{új}} = \triangle w_{ij}^{h} + \mu \triangle w_{ij}^{h,\text{előző}} \tag{3.14}$$

$$\triangle w_{ij}^{o,\text{új}} = \triangle w_{ij}^{o} + \mu \triangle w_{ij}^{o,\text{előző}} \tag{3.15}$$

A bevezetett μ paramétert momentum tagnak nevezik ($0 \leq \mu < 1$, általában $\mu = 0.9$). Ez az összegzés a deriváltak (exponenciális) átlagolásának felel meg, azaz, ha a deriváltak egyirányúak, akkor a momentum tag nagyobb lépéseket eredményez, felgyorsítva a konvergenciát, míg a váltakozó irányú deriváltak kiejtik egymást.

3.4.2. Vogl-féle gyorsítás

E gyorsítási technika alapötlete az, hogy a tanulási rátát és a momentum tagot a tanulás folyamán a hiba-felülethez illeszkedően szabályozzuk. A Vogl-gyorsítás röviden a következőképpen foglalható össze:

1. $\eta(0) := \eta_0; \mu(0) := \mu_0$
2. ha $E(t) < E(t-1)$, akkor $\eta(t) := \phi\eta(t-1)$; $\mu(t) := \mu_0$
   ha $E(t) \leq (1+\epsilon)\,E(t-1)$, akkor $\eta(t) := \beta\eta(t-1)$; $\mu(t) := 0$
   ha $E(t) > (1+\epsilon)\,E(t-1)$, akkor $\eta(t) := \beta\eta(t-1)$; $\mu(t) := 0$, és emellett az utolsó lépést érvénytelenítjük is.

ahol t az iteráció számláló, $\phi > 1$ a gyorsítási-tényező, $\beta < 1$ a lassítási-tényező, és $0.01 \leq \epsilon \leq 0.05$ a hibatűrési határ.

A módszer viselkedése kézenfekvő. Ha a hiba két iterációs lépés között csökken, akkor a tanulási rátát ϕ-szeresére növeljük (mert jó az irány, és várhatóan gyorsabban is haladhatunk) és a momentum tagot visszük magunkkal (mert segíti a konvergenciát).

Ha a hiba csupán néhány százalékkal növekedett, akkor a lépés még elfogadható, a tapasztalat azt mutatja, hogy így nehezebben akad el a hálózat a hibafelszín sekélyebb lokális minimumaiban.

Különben, ha lépés néhány százalékkal nagyobb hibát produkál, mint az előző érték, akkor a tanulási rátát lecsökkentjük ($\beta < 1$ számmal szorozzuk) és a momentum tagot nullázzuk, mert csak félrevezetne. Vogl szerint ebben az esetben jobb, ha nem is változtatjuk a súlyokat és ugyanabból a pontból újra próbálkozunk.

A módszer meglehetősen robusztus a kezdő lépésméret paraméterét tekintve: ha az η_0 túl nagy, akkor néhány iteráció alatt kialakul a megfelelő tanulási ráta. Másrészről, a módszer kifejezetten érzékeny a gyorsítási és a lassítási tényezők beállítására, ezek erősen befolyásolják a konvergencia sebességét (és „létezését" is). Vogl $\phi = 1.05$ és $\beta = 0,7$ értékeket javasol.

3.4.3. YPROP

A YPROP technika a Vogl-féle módszer egy továbbfejlesztett változatának tekinthető. Alapötlete az, hogy még a gyorsítási-, illetve a lassítási tényezőt se tartsuk állandó értéken, hanem minden egyes iterációs lépésben az alábbiak szerint változtassuk:

$$\phi(t) := 1 + \frac{K_a}{K_a + \eta(t-1)} \tag{3.16}$$

$$\beta(t) := \frac{K_d}{K_d + \eta(t-1)}. \tag{3.17}$$

ahol K_a és K_d két konstans, értelemszerűen a gyorsításra illetve a lassításra vonatkoznak. A YPROP módszernek ugyanannyi paramétere van, mint az előző Vogl-gyorsításnak, viszont tapasztalatok szerint ez a technika kevésbé érzékeny a paraméterek beállítására, vagyis a K_a, K_d paraméterek értékei nem annyira kritikusak a konvergencia szempontjából.

Az algoritmus viselkedése a következő: a gyorsítási fázisban, ha $\eta(t-1)$ kicsi ($\eta(t-1) \ll K_a$) akkor $\eta(t) \cong 2\,\eta(t-1)$, különben pedig, ha $\eta(t-1) \gg K_a$, akkor $\eta(t) \cong \eta(t-1)$. A heurisztika az alábbi: azt szeretnénk, hogy η nagyon gyorsan növekedjen, ha még nagyon kicsi, viszont ha már nagy, akkor szinte ne is növeljük, legyünk óvatosak.

A lassítási fázisban éppen ellenkező viselkedést szeretnénk: ha $\eta(t-1) \gg K_d$ akkor $\eta(t) = K_d$, azaz a tanulási ráta drasztikusan lecsökken egyetlen lépés alatt. Ha $\eta(t) \ll K_d$, akkor $\eta(t) \cong \eta(t-1)$, ebben az esetben valószínűleg egy lokális minimum környékén járunk (mert a hiba növekedett, annak ellenére, hogy a tanulási ráta nagyon pici). Ha ekkor nem változtatunk a lépésközön, akkor megnő az esély a lokális minimumhely elhagyására.

4. fejezet

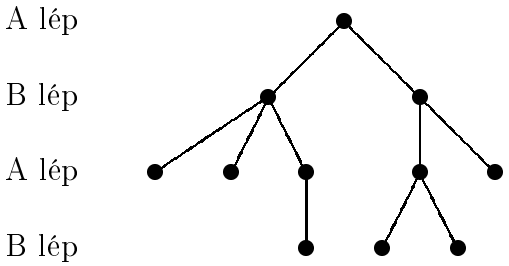
A Minimax kiértékelés

A kétszemélyes játékok elméletében nagy szerepe van a Minimax algoritmusnak, amellyel elméletben minden játék nyerő stratégiája meghatározható és a gyakorlatban is elég jól használható. A kétszemélyes, teljes információjú játékok osztályát azok a játékok alkotják, amelyekre igaz, hogy:

- A két játékos felváltva lép.
- A játékosok birtokában vannak a játékkal kapcsolatos összes információnak.
- A véletlentől nem függ a játék végeredménye.
- Minden állásban véges sok lehetséges lépés van.
- Véges sok lépésben véget ér a játék.
- A játék nulla-összegű, azaz a nyertes ugyanannyit nyer, amennyit a vesztes elveszit.
- Egyszerre nem győzhet és nem veszíthet mindkét játékos.

A játékelmélet nagyon fontos tétele, hogy egy teljes információjú kétszemélyes játék esetén mindig létezik az egyik játékos számára nyerő stratégia feltéve, hogy a játék nem végződhet döntetlennel. Ha előfordulhat döntetlen is, akkor az egyik játékos számára létezik legalább döntetlent elérő stratégia.

A kétszemélyes játékokat ábrázolhatjuk fával, amit játékfának neveznek. A 4.1. ábrán egy ilyen játékfa látható. Az A játékos teszi meg a kezdő lépést. A fa csúcsaiban játékállások vannak, élei az egyes állások közötti szabályos lépéseknek felelnek meg, így a fa gyökerébe a kezdőállás, a levelekbe pedig a végállapotok kerülnek. Azonos csúcs több helyen is előfordulhat, akár a fa azonos szintjén is. A fa azonos szintjén található csúcsokban levő állásokban



4.1. ábra. Egy fiktív játékfa

mindig azonos játékosnak (A-nak vagy B-nek) kell lépnie, ezért beszélhetünk a játékfa A-szintjéről illetve B-szintjéről.

A minimax algoritmus a játékfa csúcsaihoz rendel egyet a -1, 0, 1 értékek közül. Ez az érték

1, ha az A játékosnak van nyerő stratégiája a csúcsban levő állásból

0, ha az A-nak nincs nyerő stratégiája, de döntetlent elérő stratégiája van

-1, ha az A mindenképpen veszít a csúcsban levő állásból

az optimális ellenfél ellen.

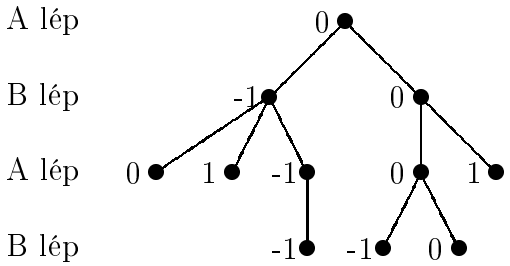
A címkézést a levelektől indulva kezdjük. A levelekhez egyértelműen hozzárendelhető a megfelelő érték, a teljes információjú, kétszemélyes játékok definíciója miatt. A belső csúcsok megcímkézése különbözik, attól függően, hogy azok a fa A-szintjén, vagy B-szintjén vannak.

- A szinten a közvetlen leszármazottak értékeinek maximuma lesz a csúcs értéke. Minthogy az A játékos következik lépésen, van választási lehetősége, ezért léphet a neki legmegfelelőbb irányba.

- B szinten a közvetlen leszármazottak értékeinek a minimuma lesz a csúcs értéke. A B szinten a B játékos lép, várhatóan az A játékos számára a legrosszabbat lépést váltasztja

Az 4.2. ábrán egy kiértékelt játékfa látható. A felcímkézés eredményeként a fa gyökerének is lesz valamilyen értéke, ami alapján eldönthető, hogy a gyökérben levő állásból a játék megnyerhető, vagy ha nem, akkor legalább döntetlen elérhető-e.

A gyakorlatban általában nem építhető fel a teljes játékfa, csak néhány lépés mélységig. A kiértékelés (illetve a fa) egy bizonyos mélységben el van



4.2. ábra. Egy kiértékelt játékfa

vágva. A levelek ilyenkor nem végállásokat tartalmaznak, nem lehet egyértelműen eldönteni róluk, hogy nyerő vagy vesztő állások. Statikus[1] kiértékelő függvénynek hívják azt a függvényt, ami számszerűsíti a levelekben levő állások értékét, mégpedig úgy, hogy pozitív értékeket rendel az A számára kedvező állásokhoz, 0-hoz közeli értéket a döntetlen-szerű állásokhoz, és negatív számokat az A-nak kedvezőtlen állásokhoz.[2] A levelek kiértékelése után ugyanúgy alkalmazható a minimax értékfelterjesztés, mint az elméleti esetben. Végül, a gyökérben levő állásban a legjobb lépés az az él, amely a maximális értékű csúcsba vezet. A megtalált lépés csak a statikus kiértékelő függvény szerint optimális, ezért nagyon sok múlik ennek a függvénynek a helyes megvalósításától, hogy mennyire jól minősíti az állásokat.

[1] Azért statikus, mert csak egyetlen állást értékel, nem több állás sorozatát

[2] A statikus kiértékelő függvény megegyezik a megerősítéses tanulás értékelő függvényének becslésével minimax politika mellett.

5. fejezet

Az Othello-n végzett tanítási módszerek

A tanulórendszer elemzéséhez többrétegű perceptront és megerősítéses tanulási eljárást implementáltunk. Az elvégzett tesztek különböző felépítésű gépi játékosok egymás elleni játszmáiból álltak. Először a játékosokat ismertetjük, majd a kísérleteket és azok eredményeit. A közösen használt jelöléseket még itt bevezetjük:

Legyen

$\mathcal{B}$ a táblák (játékállások) halmaza

$\mathcal{T}$ azon táblák halmaza, amelyeken a játszma már befejeződött. $\mathcal{T} \subsetneq \mathcal{B}$.

$\mathcal{P}$ egy kételemű halmaz, a játékosok halmaza.

$\mathcal{V}^T : \mathcal{T} \times \mathcal{P} \mapsto \{-1, 0, 1\}$ a végállapot értékfüggvénye

$$\mathcal{V}^T(t,p) := \mathcal{V}_p^T(t) := \begin{cases} 1, \text{ ha } p \text{ győzött } t\text{-ben} \\ 0, \text{ ha a } t \text{ állás döntetlen} \\ -1, \text{ ha } p \text{ veszített } t\text{-ben} \end{cases}$$

$\mathcal{A}_p(b)$ a p játékos lehetséges lépései a b állásban. $(b \in \mathcal{B}, p \in \mathcal{P})$

$\mathcal{F}_p(b)$ a b állásból p játékos lehetséges lépései elvégzése után előforduló állások halmaza, vagyis a b p-szerinti *afterstate*-jei.

Továbbá alkalmazzuk az MLP rövidítést a többrétegű perceptron (Multi Layer Perceptron) és az RL rövidítést a megerősítéses tanulás (Reinforcement Learning) kifejezésekre.

80	4	16	12	12	16	4	80
4	-30	-4	-5	-5	-4	-30	4
16	-4	1	0	0	1	-4	16
12	-5	0	0	0	0	-5	12
12	-5	0	0	0	0	-5	12
16	-4	1	0	0	1	-4	16
4	-30	-4	-5	-5	-4	-30	4
80	4	16	12	12	16	4	80

5.1. ábra. A Minimax-játékos értéktáblája

5.1. A játékosok

5.1.1. A statikus Minimax játékos

A tesztek elvégzéséhez szükségünk volt egy „ellenfélre”, amely tudott valamilyen szinten játszani. Az ellefél általában egy minimaxt alkalmazó, nem tanuló gépi játékos volt. Ennek a játékosnak a stratégiai jellemzőit főként a minimax keresés statikus állapot-kiértékelő függvénye és a keresés mélysége adja meg. A kiértékelő függvény meglehetősen egyszerű: végállapotokra egyértelműen eldönthető, hogy melyik játékos győzött. Nem végállapotok esetén pedig egy statikus értéktáblát alkalmaztunk, az értéktábla mezőihez az 5.1 ábrán látható értékek voltak rendelve.

A tábla értéke a p játékos szempontjából a következő: az értéktábla mezőihez rendelt értékeket súlyozzuk azzal, hogy melyik játékos korongja áll egy adott mezőn, és ezeket a mennyiségeket összegezzük az összes mezőre. Vagyis

$$V_p(B) = \sum_{i=1}^{64} \chi_p(b[i])\, v[i]. \tag{5.1}$$

ahol b egy tábla, $v[i]$ az értéktábla i-edik mezőjének értéke és

$$\chi_p(x) := \begin{cases} 1 \text{ ha } x = p \\ 0 \text{ ha } x = \textit{üres} \\ -1 \text{ ha } x = p \textit{ ellenfele} \end{cases} \tag{5.2}$$

A heurisztika a tábla mezőihez rendelt értékek mögött az alábbi: a sarkok a legértékesebb mezők, mert ha egy korong az egyik sarokba kerül, akkor azt többé nem lehet átfordítani. Szintén értékesek a tábla szélén található mezők, mert ezeket csupán két irányból lehet elfoglalni, szemben azzal, hogy a tábla más pozícióját nyolc irányból lehet közrefogni. Ezek után ezt a gondolatot kell csak tovább görgetni, ha egy f mező értékes, akkor azok a g mezők is értékesek, amely g mezők elfoglalásával lehetőség nyílik f-et elfoglalni, és azok a mezők kifejezetten kerülendőek, amelyek az ellenfelet hozzásegítik az értékes mezők elfoglalásához. Megjegyezzük, hogy a számítógépes Othello programok ettől jóval kifinomultabb heurisztikákkal is ki szokták egészíteni az állás-kiértékelő függvényt, pl. gyakran vizsgálják, hogy a játékosnak mekkora a mozgástere, illetve, hogy mennyire tekinthető állandónak már a játékállás. Sokszor a kiértékelő függvényt az említett fogalmak lineráris kombinációjaként kezelik, és az együtthatókat a játszma folyamán változtatják esetleg még magasabb szintű heurisztikával, annak érdekében hogy eldönthessék, vajon az adott állásban melyik tényezőnek lehet nagyobb szerepe.

A minimax játékos stratégiája támadó jellegű. Ha két elég mélyen kereső játékos játszik egymás ellen (3-4 lépés már elég mély), akkor a játszma a következőképpen alakul. A első 12 lépésben a két játékos igyekszik nem kilépni a tábla középső 4×4-es részéről. Mikor ez a belső kis tábla betelt, a kezdőjátékoson van a sor (a 13-dik lépés a kezdő játékosé.) A 13-dik lépés már negatív értékű mezőbe kényszerül és legtöbbször lehetőséget kell adnia az ellenfélnek az valamelyik szélső mező elfoglalására. Viszont a mélyebb keresésnek köszönhetően ez a kényszerlépés olyan, hogy néhány lépésen belül az ellenfélnek is ki kell engednie a kezdőjátékost egyik szélső pozícióra. Általában is elmondható, hogy a játékban kevéssé lehet megakadályozni azt, hogy az ellenfél egy adott pozícióra lépjen, inkább ellencsapásokkal kell „védekezni".

Az Othello játék „kellemes" tulajdonsága, hogy a játék vége felé közeledve a lehetséges lépések száma drasztikusan csökken, mert az üres mezők lassan elfogynak. Ezáltal végjátékban jóval mélyebb keresésre van lehetőség, mint a megnyitásban vagy a középjátékban. A statikus minimax játékosnak fontos paramétere, hogy a végjátékban hány üres mezőtől kezdje el a játékfa teljes felépítését és kiértékelését. Ez a kis kiegészítés alaposan megerősíti a játékos játékerejét. Összehasonlítként megjegyezzük, hogy az általunk használt játékos 4 lépés (azaz két lépéspár) mélységű kereséssel, ami az utolsó 6 lépést már teljesen kielemzi, képes legyőzni a Microsoft© mester szintű Reversi programját 16 korong különbséggel is.

A standard minimax algoritmus, alfa-béta levágással[3] mindig a legelső maximális értékű lépést választja, így egy hasonlóan determinisztikus játékos ellen mindössze egyetlen lehetséges játszmát tud játszani - vagyis minden

lejátszott parti ugyanazt a lépéssorozatot valósítja meg. Ezért egy kis módosítással véletlen elemet vittünk az algoritmusba. A keresés időigény csak kis mértékben ront, ha valamely pozícióból az első lépések értékeit kiszámításuk után eltároljuk, és az azonos értékűek közül véletlenszerűen választjuk a javasolt lépést. Azonos értékű lépések az értéktábla diszkrét értékei miatt meglepően gyakran fordulnak elő. Például a játszma legelső lépése is tetszőleges, bármely lépés eredményéül egy 0 értékű táblaállást kapunk.

5.1.2. Az MLP-játékos

Ez a játékos egyetlen MLP-t alkalmaz a nem végállapotot reprezentáló állások ($b \in \mathcal{B} \setminus \mathcal{T}$) kiértékeléséhez. A hálózat bemenete a „nyers" táblaállás volt és egyetlen kimenete a tábla értékét becsülte. A játékos a következő módon választotta lépését: a kapott b álláson sorban kipróbálta az összes lehetséges *afterstate*[1]-et Ha az *afterstate*-ben még nem fejeződött be a játszma, akkor azt az MLP-vel kiértékelte, különben pedig a $\mathcal{V}^T$ függvénnyel. A választott lépés ($\hat{a}(b)$) az lett, amelyik a maximális értékű *afterstate*-et eredményezte.

$$\hat{a}(b) = \arg \max_{a \in \mathcal{A}_p(b)} \{V_{\mathrm{MLP}}(b') \mid b' \in \mathcal{F}_p(b)\} \tag{5.3}$$

ahol

$$V_{\mathrm{MLP}}(b) = \begin{cases} \mathcal{V}_p^T(b), \text{ ha } b \in \mathcal{T} \\ \text{különben az MLP által kiszámolt érték} \end{cases}$$

Ez a játékos képes közvetlen módon tanulni az egyes táblaértékeket a megfelelő (tábla, érték) párok segítségével. Ebben az esetben a játékosba épített MLP egyszerű felügyelettel való tanulást végez. Alternatívaként a játékos képes egy megerősítéses tanulási folyamat aktív ágenseként működni, azaz akciót választani, valamint a környezettől kapott jutalmat a TD(λ) eligibility-vel kiegészített tanulási módszerrel feldolgozni.

5.1.3. A DMLP-játékos

Az MLP-játékos egyik hiányossága, hogy egyetlen hálózata miatt nem képes önmaga ellen játszani, mert az eligibility követést nagyon elbonyolítaná, másrészt talán feleslegesen nehezítjük meg a hálózat tanulását, ha mindkét játékos állásait egyetlen elemezzük, ugyanis egyszer páros, egyszer páratlan számú korong van a táblán.

A DMLP-játékos[2] éppen ezért már két hálózatot használ, egyet a piros, egyet a kék játékos lépéseinek értékeléseire. Ez a játékos is képes mindarra,

[1] Ld. 53. oldal

[2] A „D" betű a dupla szóra utal.

amire egy MLP-játékos. Újdonság az, hogy a két hálózata által reprezentált értékfüggvények segítségével képes a minimax-elv alapján a saját értékelő függvényén javítani, azaz lényegében egy olyan megerősítéses tanulást produkálni, amiben nincs igazi kapcsolat a környezettel (vagyis az ellenféllel). Elvben az „autodidakta" tanulás a játék optimális stratégiáját eredményezheti. Erre a tanulási folyamatra a 61. oldalon térünk ki bővebben.

5.2. A tesztek

5.2.1. Az értékelő függvény identifikációja többrétegű perceptron segítségével

Legelső lépésként (főként a szoftver funkcionális tesztjeként) egy többrétegű perceptron betanításával foglalkoztunk, még a megerősítéses tanulás alkalmazása nélkül, csupán felügyelt tanítással. Arra voltunk kíváncsiak, hogy mennyire képes egy MLP-játékos megtanulni (identifikálni) az egylépéses Minimax játékos politikáját. Nem az volt a cél, hogy az MLP-játékos hálózata minden álláshoz azonos értékeket rendeljen, mint a Minimax játékos kiértékelő függvénye, hanem csupán az, hogy egy, a Minimax által maximális értékűnek kihozott akcióhoz az MLP is az afterstate-értékek maximális értékét rendelje. Szem előtt tartva ezt az elvárást, a tanítási folyamat játszmák végigkövetésének sorozata volt. Egy ilyen végigkövetés az alábbi módon történt:

$b \leftarrow$ a kezdőtábla;

Ciklus amíg nem ér véget a játék

Ha a tanulandó színnel kell lépni, akkor

$a_{\text{MLP}} \leftarrow$ {az Mlp-játékos lépésjavaslata};
$a_{\text{Minimax}} \leftarrow$ {a Minimax-játékos lépésjavaslata};
Ha a_{MLP} és a_{Minimax} a Minimax-értéke nem egyenlő, akkor
* $\forall b' \in \mathcal{F}_p(b)$-re MLP tanítása ($b', b'$ Minimax-értéke) párokkal

Véletlen lépés a b-n

Ciklus vége

A $*$-gal megjelölt sorhoz még annyi magyarázat tartozik, hogy a b' Minimax-értékét át kell transzformálni a $[-1, 1]$ illetve, ha az MLP kimeneti rétegében levő neuronok aktivációs függvénye a logaritmikus szigmoid függvény ($\frac{1}{1+e^{-x}}$), akkor – mivel ez pozitív értékkészletű – a $[0, 1]$ intervallumba. Ez az átalakítás egyszerű lineáris transzformációkkal történt.

A futtatások alapján kiderült, hogy már egy olyan MLP is, amely mindössze egyetlen neuronnal rendelkezik a rejtett rétegben, képes tökéletesen megtanulni az egy lépés mélységű Minimax politikát. Ha mindkét réteg aktivációs függvénye lineáris volt, akkor rejtett réteg súlymátrixsza arányos volt a Minimax játékos értéktáblájához rendelt súlyaival. Tehát ebben az esetben nem csak a politikát, hanem az értékfüggvényt is sikeresen megközelítettük.

5.2.2. Standard RL futtatások

A tesztek túlnyomó része a diplomamunka eredeti alapgondolatára irányult: azt vizsgáltuk, hogy a tanuló játékos milyen ütemben, milyen szintre fejlődik pusztán a környezetével érintkezve, a megerősítéses tanulás modellje szerint.

Az MLP-játékost alkalmaztuk erre a célra, különböző ellenfelekkel állítottuk szembe, játszmák tíz-, néha százezreit játszattuk velük. A megerősítéses tanulás „fegyelmező eszköze", a *jutalom* képzése – ahogy a játékállások kódolása is – mentesült mindenféle heurisztikától és mélyebb analízistől. A legkézenfekvőbb kódolást választottuk: a játszma alatt a közvetlen jutalom minden lépés után 0 volt, a játékos utolsó lépését pedig 1-gyel jutalmaztuk, ha nyert, -1-gyel büntettük, ha vesztett, és szintén 0-t kapott, ha döntetlent ért el. Ez a kódolás konzisztens azzal az elgondolással, hogy a megtanult értékfüggvény -1 körüli értékeket rendeljen a nagyon rossz állásokhoz, 0 körül a várhatóan döntetlenhez vezető állásokhoz és 1 körüli pozitív értékeket a lényegében már megnyert állapotokhoz; függetlenül attól, hogy az állásból hány lépést kell még megtenni a játszma végéig.

A megerősítéses tanulásra bíztuk, hogy játszmák ezrei alatt a játszma végén kapott jutalmat (vagy büntetést) az egyre korábbi lépéseken előre is éreztesse. A TD(λ) algoritmus eligibility-vel kiegészített változatát használtuk, emlékeztetőül, a TD-hiba az alábbi módon képződik (ld. 24. oldal):

$$\delta(t) = r_{t+1} + \gamma\, V(s_{t+1}) - V(s_t).$$

A γ paramétert 1-nek választottuk, mivel a feladat epizodikus és nem kell végtelen összegzésektől tartani. Az MLP η paramétereit nem változtattuk a tanítás alatt, végig 0.005 volt. Nem alkalmaztunk semmilyen gyorsítást sem, az egyszerű gradiens módszert akartuk elemezni. Az emlékeztető nyom (eligibility trace) lecsengető tényezője $\lambda = 0.4$ volt, ez az utolsó 3-4 (mohó módon

választott) lépésre terjesztette ki a jutalom hatását.[3] Az explorálást, a véletlenszerűen választott felderítő lépéseket a feladat epizódikus jellege miatt lehetőségünk volt úgy korlátozni, hogy a tanítást hatékonyabbá tegye. Kihasználhattuk, hogy általában egy játszma 60 lépésből áll, a tanulás kezdeti szakaszában célszerű a játszmák végére összpontosítani a kísérletező lépéseket, hogy minél több végállapotot lásson a játékos. Ezért az MLP-játékost úgy valósítottuk meg, hogy az explorációt a játszma különböző stádiumában más-más szinten lehessen tartani. Az exploráció kezdeti beállítása: 10% mindenhol, azaz átlagosan minden tizedik lépésben véletlenül választott akciót a rendszer; de az 50-dik lépéstől az exploráció már 30%. A tanulás előrehaladtával a felderítő lépéseket célszerű egyre inkább a játszma elejéhez közelebb megtenni, mert ekkorra a végállások értéke már nagyjából kialakult.

Fontos kiemelni, hogy a megerősítéses tanulás alapmodelljében exploráló lépés után soha nem történik meg az állapot-érték függvény hangolása. Az Othello-probléma azon speciális tulajdonsága miatt, hogy a lehetséges jutalmak értékét ismerjük, ez a szabály enyhíthető. A megerősítéses tanulás mindig csak maximális értékű lépés választásakor hangolja az értékfüggvény becslését. Jelen esetben tudjuk, hogy a lépés mindenképpen maximális értékű, ha 1 jutalmat kap érte a játékos, függetlenül attól, hogy felderítő lépés volt, vagy sem. Ez a kis kiegészítés sokat számít, hiszen éppen a nyerő állapotok előtti állapotok értékét tudjuk gyorsabban becsülni általa.

A környezet és az ügynök áttekintése után térjünk át a kettő összekapcsolására. A tanító algoritmust igyekeztünk általánosan megírni, úgy, hogy az alkalmas legyen két tetszőleges (gépi) játékos egyidejű tanítására. A játékosok egymással játszva tanultak. Ez csak akkor lehet hatékony, ha legalább az egyik játékos a játszma minden lépésében felhasználja a tudását, vagyis a mohó politikát alkalmazza.[4] A tanítási folyamat három fázis ismétléséből állt.

1. A piros játékos mohó politikát alkalmaz, a kék játékos számára engedélyezve vannak a felderítő lépések. A kék játékos javítja az értékfüggvényét (persze csak mohón politika szerint választott lépéssorozat után, vagy a fent említett kivételes helyzetben.) A piros játékos nem tanul, ha a kék játékos ténylegesen véges exploráló lépéseket[5].

2. Mint az első fázis, csak a két játékos szerepet cserél.

[3]Érdemes lenne komolyan elgondolkodni, hogy milyen kapcsolat áll fenn az RL-ben alkalmazott eligibility trace és az MLP momentum tagja között.

[4]„Nem tudok alkalmazkodni hozzád, ha te is folyton alkalmazkodni próbálsz hozzám."

[5]Lehet, hogy a kék játékos nem is tud tanulni, mert egy minimax-játékos. Ebben az esetben nyugodtan tanulhat a piros, mert csak elmélyíti a mohó politikáját.

3. Mindkét játékos mohó politikát alkalmaz, és erre tanul is. A tesztek alatt ellenőrizhető volt, hogy éppen melyik játékos erősebb.

A tesztek során az első és második fázis 5-5 játszmából állt, a harmadik egy játszmára korlátozódott. Ebből következik, hogy az explorálni képes játékosok a tanulás során lejátszott játszmáknak közel a felében nem az addig megszerzett ismereteik teljes kihasználásával választottak lépést.

MLP-játékos Minimax-játékos ellen

Ezekben a tesztekben mindig a piros játékos volt a tanuló, míg a kék egy statikus minimax játékos volt. Minden tesztet kétféleképpen végeztük el. Vagy véletlen súlyokkal indítottuk a tanuló játékos MLP-jét vagy pedig az előző részben ismertetett identifikáció utáni állapot volt a kezdeti háló. Ezzel azt vizsgáltuk, hogy milyen következménnyel jár, ha a hálózatot „előtanítjuk".

A grafikonok[6] a tanulás folyamatát jellemzik, a nyert és az elvesztett játszmák arányát ábrázolják a piros játékos szempontjából, a lejátszott játékok számának függvényében. Ezekben a játszmákban a piros játékos gyakran tett felderítő lépéseket, ezért az ábrák nem tükrözik a tényleges különbséget a két játékos között, de a tanuló játékos fejlődését szépen mutatják.[7] A tanulás után elvégeztünk egy 2000 játszmából álló összehasonlító tesztet, ami alatt a tanuló játékos már nem tanult és nem is kísérletezett új lépésekkel. A kapott eredményeket a 5.1 és a 5.2 táblázatokban közöljük. Hacsak másképp nem jelöltük, akkor a kék játékos a statikus egylépéses Minimax-játékos volt.

Az egyik kiemelkedően sikeres MLP-játékost (amelyik 100%-ot ért el) a kétlépéses minimax játékossal is tréningeztettük. Ennek a tanításnak a végeredményeként, több, mint 200.000 parti után 1000 játszmából 547-et nyert a tanuló játékos és 26 döntetlent ért el. A tanulás folyamatáról készült grafikont a 5.2 ábra mutatja. Ez az eredmény már figyelemre, mert a két lépés mélységben kereső minimax játékos erős kezdő szintnek felel meg. Megjegyezzük, hogy az MLP-játékos értékelő függvényre vonatkozó becslése és a minimax algoritmus kombinációja jobb (tervező) játékost eredményez. A későbbiek folyamán erre még visszatérünk.

MLP-játékos MLP-játékos ellen

A Tesauro-féle TD-Gammon teljesen önállóan fejlődött erős haladó szintre, nem volt szüksége „tanár"-ra. A szerző is rejtélyesnek találta a cikkben,

[6] A grafikonok egy-egy konkrét futtatás eredményét mutatják. Több gépidő segítségével átlagértékeket is lehetett volna képezni (egy 100.000 lépésből álló teszt átlagosan 4 óra alatt futott le egy 200 MHz-es Pentiumon számítógépen)

[7] A grafikonok az „online"-teljesítményt mutatják.

amelyben beszámolt a programjáról, hogy milyen nagy fejlődésre volt képes a program önállóan is. Tesauro azt fejtegette, hogy ez valószínűleg a játék sztochasztikus jellege miatt van így. Ezt az álláspontot sajnos egyenlőre nem tudjuk ellenpéldával megcáfolni, ugyanis az Othello öntanuló tesztjei a korlátozott futtatások miatt egyenlőre nem olyan sikeresek, mint a backgammon esetében. Az elvégzett teszt kezdőlépésnek tekinthető. A két játékos 16-16 előhangolt rejtett rétegbeli neuronnal kezdett el tanulni. A tanulás folyamata a 5.3 ábrán látható. Az összehasonlító teszt eredményei a 5.3 táblázatban találhatóak.

Az öntanulásnak van néhány elméleti problémája.

> Egyrészt az MLP-játékos függvényapproximátora korlátos erőforrásokkal rendelkezik (*neuronok száma* $\times$ 66 + 1 db. valós értékű súllyal), ezért az egyik játékállás hangolása hatással van még több más játékállás hangolására is. Ez a hatás egyik oldalról garantálja az általánosítást, másik oldalról túlzott általánosításhoz vezethet és így nem mindig előnyös. Több neuronnal a rejtett rétegben ez a tényező biztosan enyhíthető, de ez a megoldás lassubbá teszi a hálózat betanulását.
>
> Másrészt a TD(λ) algoritmus „bootstrapping" algoritmus, ami annyit jelent, hogy a becsléseket szintén becslések alapján javítja. A témával foglalkozó kutatók elméleti példákat mutattak be azt illusztrálva, hogy a „bootstrapping" algoritmusok függvényapproximátorokkal kombinálva divergálni is képesek.

Mindezek ellenére használják ezeket az algoritmusokat, mert a gyakorlatban *érdekes módon* gyakran sokkal jobb eredményeket érnek el, mint az elméletileg alátámasztott módszerek.

5.2.3. Minimax-iterációs futtatások

A minimax iteráció kifejezés alatt azt az algoritmust értjük, amelytől nem kevesebbet vártunk, mint hogy az optimális stratégiát építse fel működése eredményeként. Az elmélet nagyon egyszerű, a minimax kiértékelésben szereplő játékfán értelmezhető rekurzív összefüggést használjuk ki.

Annak érdekében, hogy ez a rekurzív tulajdonság még nyilvánvalóbb legyen, módosítsuk a játékfa értelmezését. A fa struktúrája változatlan, csak az egyes csúcsokhoz rendelt értékeket képezzük kicsit más módon. Legyen ez az érték csúcsban szereplő állás értéke annak a játékosnak a szempontjából, akinek az adott állásban lépnie kell.[8] Ez a módosítás változtatja a fa belső

[8] A játékfa eredeti értelmezése szerint az állások értékei mindig a MAX játékos szempontjából számítódtak.

csúcsainak címkézését is: ha x egy belső csúcsa a játékfának, akkor

$$\mathcal{V}(x) = -min\{V(x_i) \mid x_i \text{ az } x \text{ közvetlen leszármazottja}\}.$$

Szavakban ez annyit jelent, hogy egy adott állás annyira jó nekünk, mint amennyire az ellenfélnek a következő állás a rossz lehet.

Az elmélet átültetése a gyakorlatba a következőképpen történt. A DMLP-játékos két neurális hálózatát használtuk a piros illetve a kék állások értékének becslésére. Látható, hogy a minimax-iterációhoz nincs szükség tényleges játszmákra, csak állásokra, mert minden nemterminális állásra fennáll az összefüggés. A véletlenszerűen generált állások azonban lehetnek szabálytalanok is (amelyek sohase fordulhatnak elő), ezért két erősen véletlenszerűsített minimax-játékos által lejátszott partik állásait használtuk fel. A DMLP-játékos minden egyes táblaállás alapján tudta módosítani egyik vagy másik értékfüggvényét a rekurzív összefüggés alapján, sőt, az adott tábla 90, 180 és 270 fokkal történő elforgatásaiból nyert állások értékét is javította. A játszma állásait visszafelé lejátszva még egyszer felhasználtuk annak reményében, hogy ez az irány nagyobb változtatásokat eredményez.

Az elmélet kétségkívül elegáns, viszont az elvégzett tesztek nem hozták meg a hozzájuk fűzött reményt. Néhány teszttel azt vizsgáltuk, hogy képes-e a DMLP-játékos két approximátora konvergálni néhány konkrét játszmából álló játékhalmazon. Ezekből a tesztekből kiderül, hogy már 10-15 játszma megtanulásához is legalább négy rejtett neuron kell, ami még nem lenne probléma, de a tanulási idő is meglehetősen nagy volt.

Egyetlen hosszabb tesztre volt időnk csupán. Ebben a tesztben generált játszmák állapotatait használva javítottuk a becsléseket. 250.000 játszma után[9], 15 rejtett neuronnal a DMLP-játékos még mindig csak az egylépéses minimax játékos szintjén volt.

5.2.4. Következtetések

Az elvégzett tesztek alapján arra következtetünk, hogy a megerősítéses tanulás és a többrétegű perceptron összekapcsolása mindenképpen eredményes az Othello játék esetén is, de az alap módszer önmagában nem elég arra, hogy magasabb szintet érjen el az öntanuló játékos.

Várhatóan igazi eredményt csakis öntanulással (tanuló játékostól való tanulással) lehet elérni. A statikus játékostól való tanulásnak több problémája is van:

> Az egy-, és kétlépéses minimax játékos ellen elért sikerek azt mutatják, hogy ha a tanuló elegendően sokszor képes nyerni (legalább 2-3%) már

[9] 3 nap...

a tanulás elején, akkor viszonylag hamar képes megtalálni a megfelelő lépéseket. Egy erősebb játékos ellen ez nem lehetséges, mert véletlenszerű játékkal annyira ritkán nyer csak, hogy az akkor kapott jutalom szinte nem is javítana az értékfüggvényen, amit a szinte állandó vereség lényegében a konstans -1–re állít.

Másrészt a mélyebb keresést végző játékosokkal nagyon sokáig tart a tanítás, mert sokkal több állapotot kell vizsgálnia a statikus játékosnak is.

Az öntanulással és általában a tanulással az a probléma, hogy a játékerő nem „tranzitív", azaz, ha A megveri B-t és B megveri C-t, akkor nem biztos, hogy A megveri C is, ha A és B tanuló játékos, és A a B ellen tanult játszani. Ennek az az oka, hogy a tanulás alatt a játékos sokkal inkább az ellenfél gyengeségeit keresi meg és nem a játék igazi stratégiáját. Ezt a két fogalmat egyszerre kellene kezelni, vagyis egyiket sem lenne szabad túlzásba vinni vagy elhanyagolni. A két tanítási tesztből talán éppen ez az, ami hiányzik: az RL futtatások alkalmával csak a statikus játékos ellen tanult a program, a minimax-iteráció alkalmával csak az optimális stratégiát kereste. Az optimális stratégia „túl nagy falat", túlzottan erratikus függvény, nincsen könnyen megfogható része, amin el lehet indulni.

További kutatásra egy újabb öntanuló módszert, aminek lényege röviden az következő:

A DMLP-játékost használjuk fel, kicsit továbbfejlesztve, hogy képes legyen a minimax algoritmussal legalább 2 mélységben keresni.

A tanítás első lépéseként identifikáljuk a statikus minimax játékos politikáját a DMLP-játékossal. Ezzel azt értjük el, hogy a DMLP ugyanúgy fog működni nagyobb mélységekben is, mint a minimax játékos.

Következő lépésként csak az egylépéses DMLP-játékos tanul a saját maga kétlépéses változata ellen, amíg elegendően nagy arányban le nem győzi.

Ezután már csak az előző pontban leírt lépést kell ismételgetni, vagyis a betanult játékos mindig a saját statikus kétlépéses változata ellen tanul.

Optimális esetben így növekedhetne a keresés hatása, viszont legfeljebb két lépés mélységig kell azt elvégezni. Ezzel a mély keresések okozta időproblémát kiküszöbölhetjük. Marad az a kérdés, hogy mennyire tudjuk így az

5.1. táblázat. Előhangolás nélkül elért eredmények.

neuronok száma	nyert	vesztett	döntetlen	%
1	1411	531	58	70.55
2	1940	42	18	97
4	1769	92	139	88.45
8	1919	81	0	95.95

5.2. táblázat. Előhangolással elért eredmények.

neuronok száma	nyert	vesztett	döntetlen	%
1	1252	697	51	62.6
2	1739	222	39	86.95
4	2000	0	0	100
8	1789	74	137	89.45

optimális stratégiát is megtanulni, vagyis, hogy az optimalitás szempontjából jobb-e egyáltalán a mélyebb kereséssel dolgozó DMLP-játékos. Ennek kiderítésére további vizsgálatokra van szükség.[10]

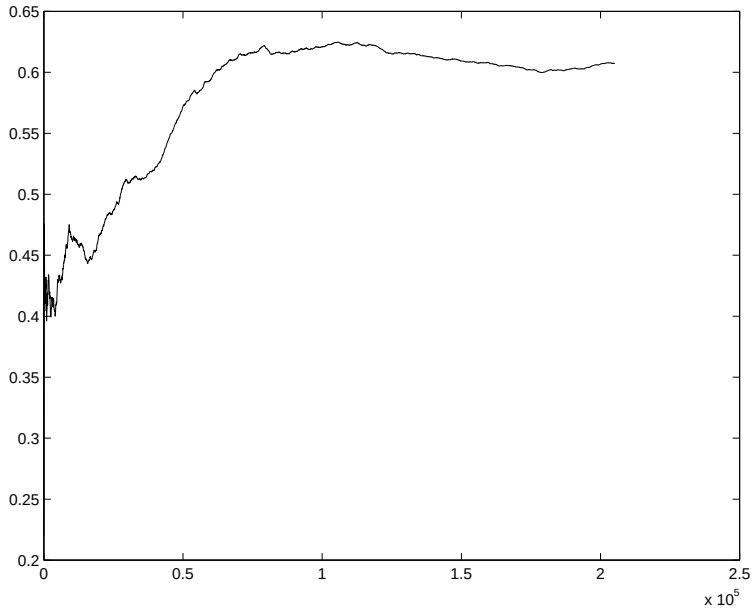
Ugyancsak érdemes lenne a TD(λ) algoritmus helyett Monte Carlo algoritmust kipróbálni. Ez egy elméletileg jobban megalapozott módszer, elkerüljük a torzított becslések okozta problémákat. Az Othello relatív rövid játszmái miatt jó esélyt látunk a könnyű megvalósításra is.

Végül még azt jegyezzük meg, hogy érdekes lehet más típusú függvényapproximátor kipróbálása. A játék azon tulajdonsága miatt, hogy egy akció igen sokat változtat az állapoton, a döntési téret nagyon bonyolult felületekkel darabolja fel. Egy hatékonyabb approximátorral ezek a felületek talán jobban reprezentálhatóak.

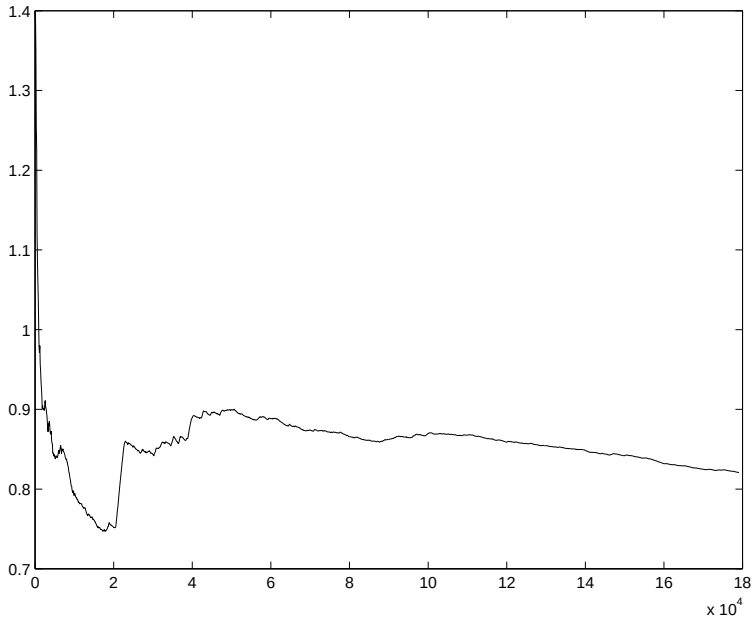
[10] A megoldáshoz valószínűleg elég megvizsgálni, hogy ha az értékelő függvény és az egylépéses minimax értékelőfüggvény által adott értékek megegyeznek, akkor az értékelő függvény kielégíti-e az optimális értékfüggvényre vonatkozó Bellman-egyenletet. Természetesen a konvergencia sebességéről ezáltal még nem nyertünk információt.

5.3. táblázat. Öntanulással elért eredmények.

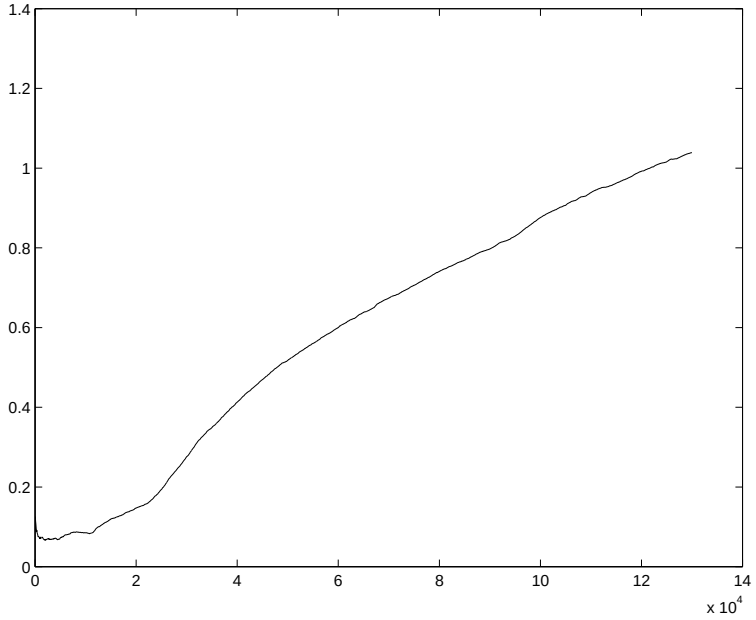
piros játékos	kék játékos	piros nyert	kék nyert	döntetlen
MlpRed	MlpBlue	817	1113	70
MlpRed	MiniMax1	1371	603	26
MiniMax1	MlpBlue	622	1262	116



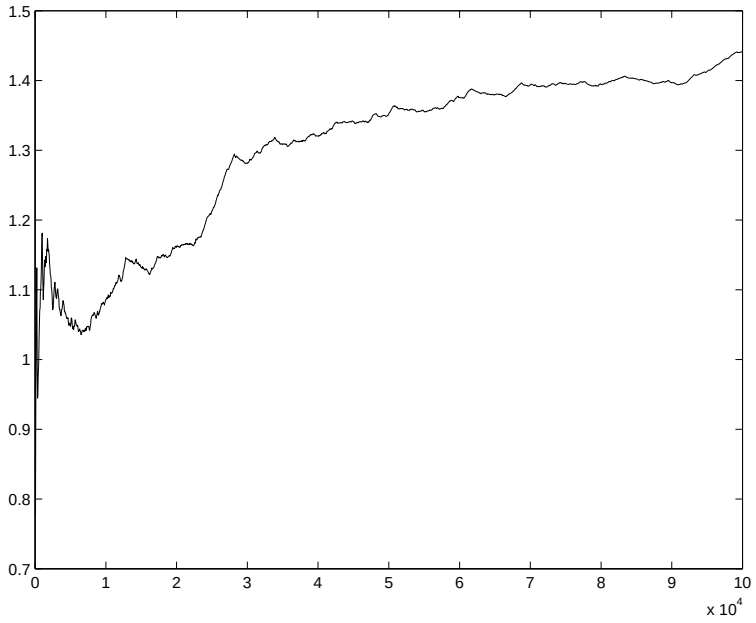
5.2. ábra. A 100%-osan teljesített játékos továbbtanítása Minimax2 ellen



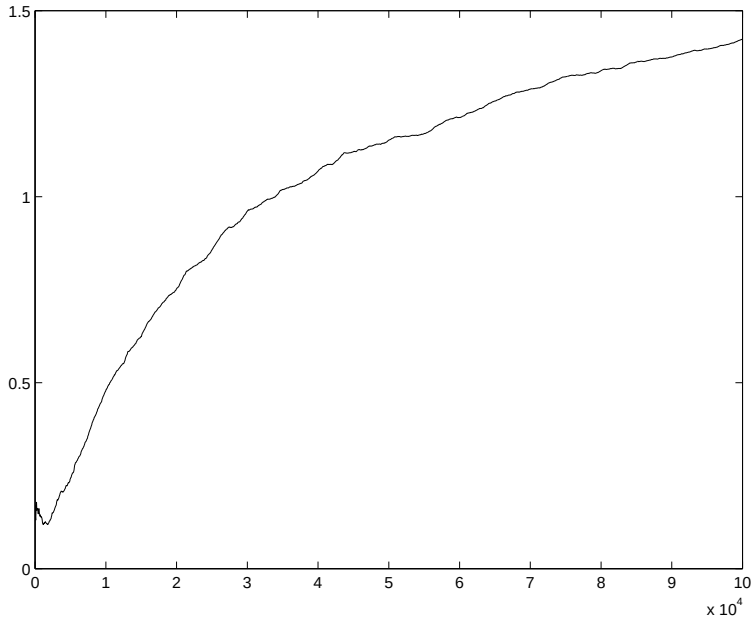
5.3. ábra. A 16-16 neuronnal, előhangolással



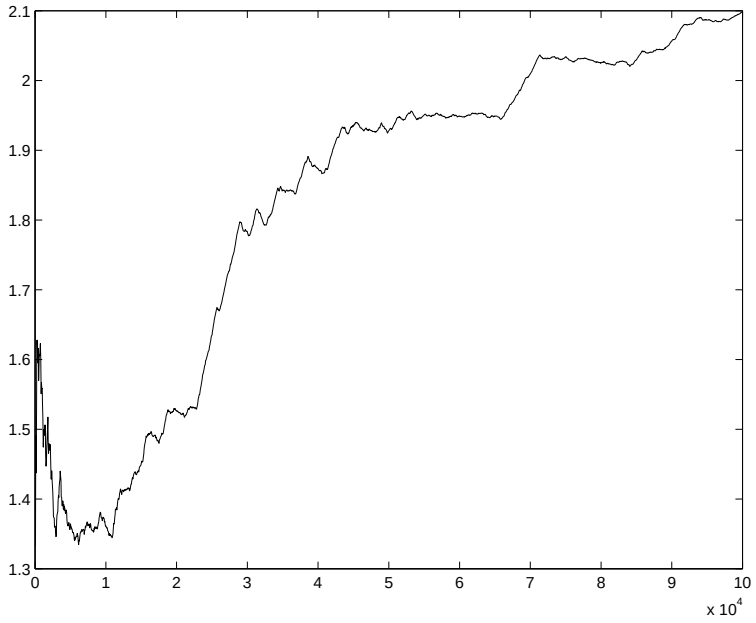
5.4. ábra. Egy rejtett neuronnal, előhangolás nélkül



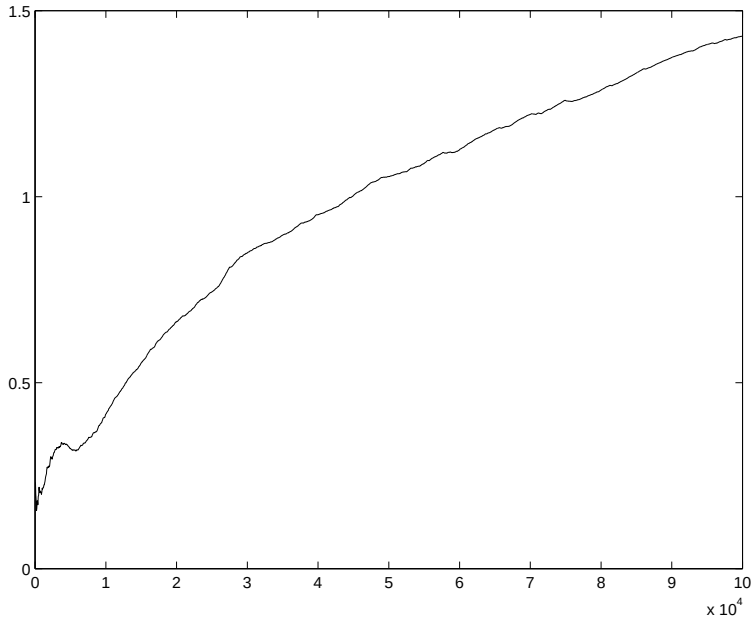
5.5. ábra. Egy rejtett neuronnal, előhangolással



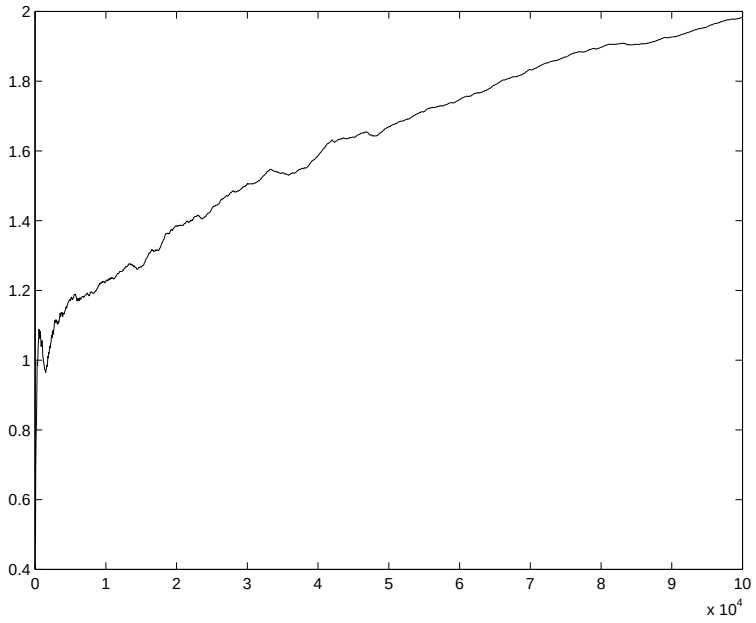
5.6. ábra. Két rejtett neuronnal, előhangolás nélkül



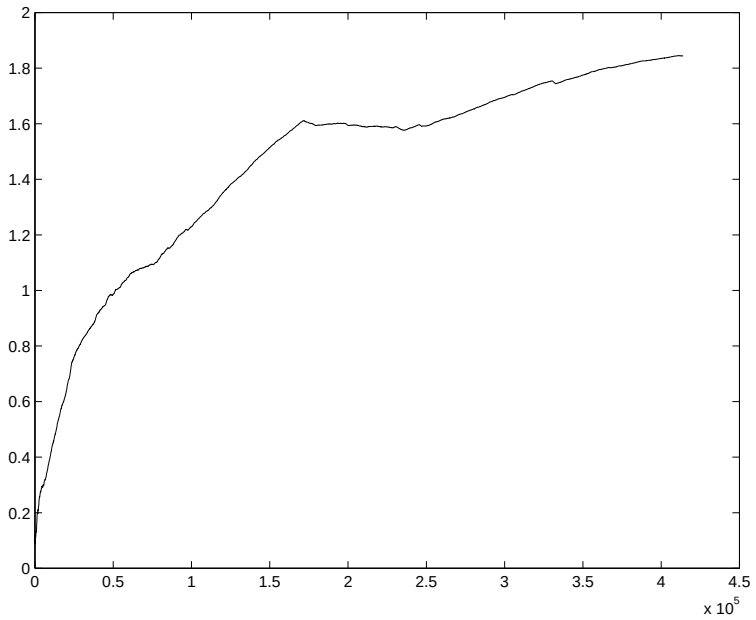
5.7. ábra. Két rejtett neuronnal, előhangolással



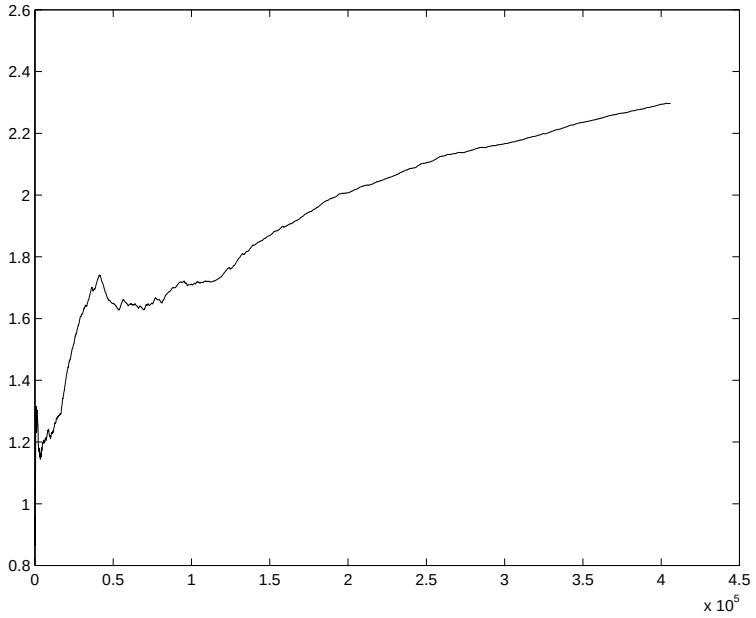
5.8. ábra. Négy rejtett neuronnal, előhangolás nélkül



5.9. ábra. Négy rejtett neuronnal, előhangolással



5.10. ábra. Nyolc rejtett neuronnal, előhangolás nélkül



5.11. ábra. Nyolc rejtett neuronnal, előhangolással

6. fejezet

Függelék

6.1. A backgammon játék

A backgammon egy közismert kétszemélyes táblás játék, sok versenyt és bajnokságot rendeznek belőle világszerte. Valószínűleg több professzionális backgammon játékos van, mint ahány professzionális sakkozó. . A játék szabályrendszere meglehetősen szerteágazó, mi itt csak némi rálátást szeretnénk adni az összetettségére.

A játékot két játékos játsza 15 fehér és 15 fekete bábuval egy 24 részre osztott táblán. A játék célja, hogy a játékosok a bábuikkal körbehaladva a táblán az összes saját bábot az ellenfél negyedébe juttasák. Az győz, akinek ez előbb sikerül. A két játékos bábui egymással szembe közlekednek; a fehér bábok a bal alsó sarokból indulnak és a jobb alsó, majd a jobb felső sarkot érintve (vagy áthaladva rajtuk) a bal felső sarokba érkeznek. A fekete játékosok útja a bal felső - jobb felső - jobb alsó - bal alsó sarkok által kijelölt útvonal. A soron következő játékos két dobókockával dob, és a dobott értékeket használhatja fel a lépéséhez. Egyik bábujával annyit halad előre, amennyi az egyik kockán van, majd ugyanezt megteszi egy másik bábuval és kockával is, de akár ugyanazzal a bábuval is léphet kétszer. A játékos nem léphet olyan mezőre, amit az ellenfél blokkolt, azaz két ellenséges bábu van rajta. Ezen kívül van mód az ellenfél bábuját „kiütni", aminek következtében az visszakerül a kiinduló pontjára; és a dupla dobásoknak is megkülönböztetett szerepe van. Ráadásul a játékot egy speciális módon is meg lehet nyerni, ezt hívják *backgammon*-nak.

6.2. Az Othello játék

Az Othello játék sok szempontból optimális a számítógépes megvalósításra. Kétszemélyes, a játékosok felváltva lépnek, teljes információjú - mindkét játékos tisztában az állással, és a saját, illetve az ellenfél lehetséges lépéseivel, véges sok lépésben véget ér - átlagosan 30-30 lépést tesznek meg a játékosok. Kevés, egyszerű és könnyen kiértékelhető szabálya van, a véletlennek nincs szerepe, a játék nulla összegű. A mesterséges intelligenciából ismert heurisztikus lépéskereső algoritmusok nagyobb lépésmélységben képesek dolgozni, mint amit az emberi játékos követni tudna[1]. Mindennek köszönhetően a számítógépes Othello programok mindig sikeresek és divatosak voltak.

A Othello-t két játékos játsza egy 8×8–as táblán. A játékhoz 64 darab korongra van szükség, minden korongnak egyik oldala piros, a másik pedig kék. A kezdő játékos színe a piros, a másik játékosé a kék[2]. Az 6.1 ábra a kezdőpozíciót mutatja.

A játékosok célja, hogy a parti végén több saját koronggal rendelkezzenek, mint az ellenfél. A játéknak akkor van vége, ha már egyik játékos sem tud korongot elhelyezni a táblán. A soron következő játékos úgy helyezhet el korongot a táblán, hogy a már meglevő korongjai segítségével és az újonnan felrakott koronggal az ellenfelének legalább egy korongját közrefogja egy egyenes mentén. Az 6.2 ábra egy középjátékbeli állást mutat, a c2 mező, ami felett a célkereszt alakú kurzor éppen áll egy szabályos lépés a piros játékos számára, mert pl. a c7-en levő korongjával így közrefogja a kék játékos négy korongját is.

Amikor a játékos lerakja a korongját, az ellenfél összes közrezárt korongját átfordítja[3] így megnövelve a saját korongjainak a számát. A piros játékos c2 mezőre elhelyezett korongja az 6.4 ábrán látható állást eredményezi, amiből majd a kék játékosnak kell lépnie:

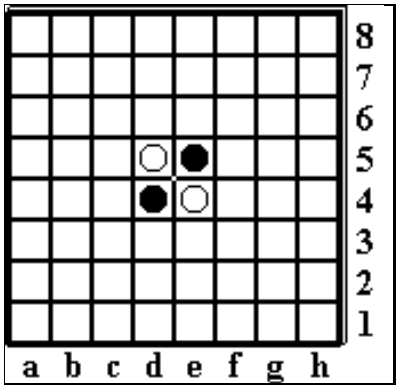
Főleg a végjáték során, gyakran előfordul olyan eset, amikor a soron következő játékos nem tud lépni, mert nincs megfelelő mező, viszont másik játékosnak van szabályos lépése. Ilyen esetben a játékosnak át kell adnia a lépés jogát a másik játokosnak - passzolnia kell. Erre példa a 6.5. ábra (a kék játékos szempontjából). Más esetben viszont nem passzolhatnak, aki tud lépni, annak kötelező is egyben.

Az 6.6 ábrán látható állásban már nem passzol senki - a játszának vége van. Az eredmény: a piros játékos győzött 26:3 arányban.

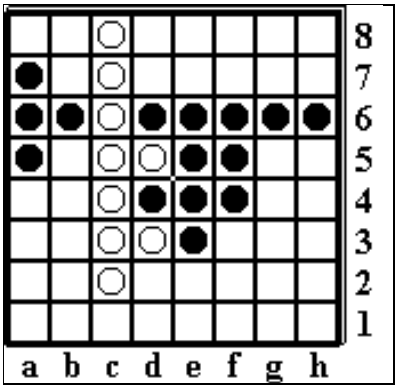
[1] Ez annak köszönhető, hogy egyetlen lépéssel igen nagy mértékben megváltoztatjuk a táblán kialakult állást. Átlagosan 4-5 korongot fordít át egy lépés a játszma folyamán, de előfordulhat olyan helyzet, ahol egy lépés 19 korongot foglal el (a lehetséges 64 pozícióból)

[2] A mellékelt ábrákon a pirosnak az üres, a kéknek a kiszínezett korong felel meg

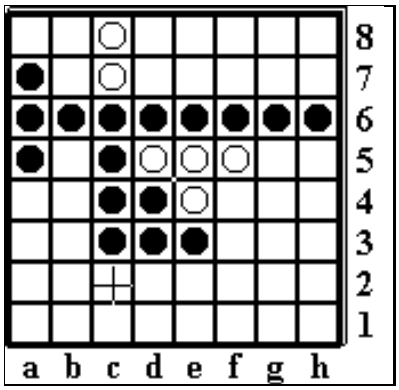
[3] Ezért is hívják az Othello-t Reversi-nek...



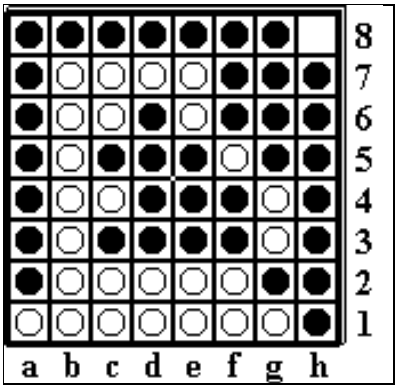
6.1. ábra. Az Othello kezdőállása



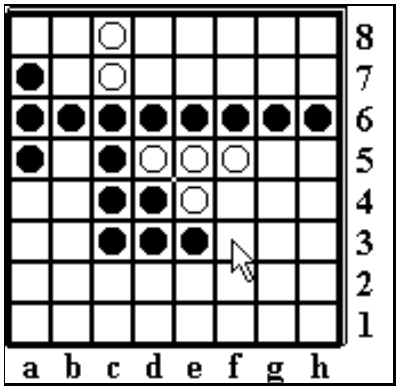
6.4. ábra. A lépés utáni állapot



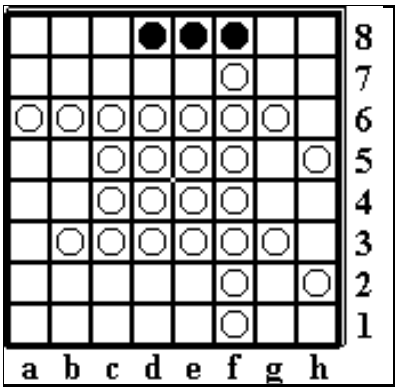
6.2. ábra. Egy szabályos lépés



6.5. ábra. Amikor passzolni kell



6.3. ábra. Ide nem lehet lépni



6.6. ábra. A játéknak vége

6.3. A MiniMax osztály dokumentációja

Ábrák jegyzéke

Táblázatok jegyzéke

Irodalomjegyzék

[1] A. E. Bryson and C. Ho, Y. *Applied Optimal Control.* Blaisdell, New York, 1969.

[2] C. Darken and Moody. J. Towards faster stochastic gradient search. *Advances in Neural Information Processing Systems*, 4:1009–1016, 1992.

[3] I. Futó, editor. *Mesterséges Intelligencia.* Aula kiadó, Budapest, 1999.

[4] S. Haykin and G. Leung. Classification of radar clutter using neural network. *IEEE International Conference on Acoustics, Speech and Signal Processing*, 4:125–128, 1992.

[5] G. E. Hinton and J. A. Anderson. *Parallel Models of Associative Memory.* Lawrence Erlbaum Associates, Potomac, Maryland, 1981.

[6] Y. Le Cun, L. D. Jackel, B. Boser, and J. S. Denker. Handwritten digit recognation: applications of neural network chips and automatic learning. *IEEE Communcations Magazine*, 27:41–46, 11 1989.

[7] K. F. Lee and S. Mahajan. Pattern classification approach to evaluation function learning. *Artificial Intelligence*, 36:1–26, 1 1988.

[8] R.P. Lippman. An introduction to computing with neural nets. *IEEE ASSP Magazin*, 4:4–22, 1987.

[9] W.S. McCulloch and W. Pitts. A logical calculuc of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–137, 1943.

[10] M.L. Minsky and S.A. Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1969.

[11] M.L. Minsky and S.A. Papert. *Perceptrons.* MIT Press, Cambridge, MA, expanded edition edition, 1988.

[12] M.L. Minsky and O.G. Selfridge. Learning in random nets. *Information Theory*, 1961.

[13] G. Palmieri and R Sanna. Automatic probabilistic programmer/analyzer for pattern recognition. *Methodos*, 48:337–357, 12 1960.

[14] A. Papoulis. *Probability, Random Variables , and Stochastic Processes.* McGraw-Hill, New York, 1984.

[15] F. Rosenblatt. *Principles os Neurodynamics.* Spartan, Chichago, 1962.

[16] D.E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. 1986.

[17] J.J. Shynk. Performance surfaces of a single-layer perceptron. *IEEE Transactions on Neural Networks*, 1:268–274, 1990.

[18] R.S. Sutton and A.G. Barto. *Reinforcement Learning.* MIT Press, 1997.