NYILATKOZAT

Név: MOLNÁR ANNA ÁGNES
ELTE Természettudományi Kar, szak: matematika
NEPTUN azonosító: VEJWGG
Szakdolgozat címe: JUTALMAZOTT TANULÁS ÉS A TORPEDÓ JÁTÉK

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2021. 05. 29.

Molnár Anna

a hallgató aláírása

Eötvös Loránd Tudományegyetem

Természettudományi Kar

Molnár Anna Ágnes

Jutalmazott tanulási módszerek és a torpedó játék

Szakdolgozat

Matematika BSc, elemző szakirány

Témavezető:
Bérczi-Kovács Erika
Operációkutatási Tanszék

Budapest, 2021

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Bérczi-Kovács Erikának, amiért megismertette velem a jutalmazott tanulás témakörét és akinek a szakmai iránymutatásain túl hálával tartozom a folyamatos biztatásáért és szüntelen türelméért.

Szeretnék még köszönetet mondani Lukács Andrásnak, aki tanácsokkal látott el a projektem jövőbeli kibővítéséhez, a családomnak, amiért támogattak a tanulmányaim során, valamint hallgatótársamnak, Szökrön Dorottyának a gépi tanulásról folytatott izgalmas beszélgetésekért.

Tartalomjegyzék

Bevezetés

A gépi tanulás korunk egyik legnépszerűbb kutatási területe. Amióta e szakolgozat megírásába belefogtam, egy új ágens, az Agent57 a megerősítéses tanulási ágensek egy ismert pontversenyében, az Atari 2600 hatvankét játéka közül tizenötben rekordot döntött. Talán nem túlzás azt állítani, hogy a gépi tanulás fejlődése napjaink technológiai fejlődésének legfőbb mozgatórugója. Bár az üzleti és adattudományi célokra használt módszerek többnyire a felügyelt és felügyelet nélküli tanulás kategóriái közé sorolhatók, a jutalmazott vagy másnéven megerősítéses tanulás szerepe több évtized után ismét egyre jelentősebbé válik. Az önvezető járművek vezérlése, a chatbotok működtetése és különféle robotok tanítása mind jutalmazott tanítási feladatok. Ezen felül megerősítéses tanulási ágensekkel szimulálható egyének vagy csoportok adott helyzetbeli viselkedése, amely komoly felhasználási lehetőségeket jelent mind az üzleti életben (például vásárlói preferenciák vizsgálatával), az adatbiztonság területén (internetes oldalak felhasználóinak adatvédelmi szokásainak szimulációjával), valamint a gazdaságtudományban és a kognitív tudományban is. Az egyre szélesebb körben alkalmazott mély neurális hálók strukturájának vizsgálatával foglalkozó hiperparaméter-optimalizálásban szintén alkalmaztak jutalmazott tanulási módszereket. Valójában nehezebb példát találni olyan problémakörre, ahol a gépi tanulás ne volna valamilyen módon hasznosítható.

E szakdolgozat célja, hogy a jutalmazott tanulás egyik legismertebb forrásának, a Richard S. Sutton és Andrew G. Barto Reinforcement Learning: An Introduction című könyvének első hét fejezetének tematikáját követve bemutassam az alapvetőbb jutalmazott tanulási módszereket, valamint az, hogy létrehozzak egy személyes projektet, amely a későbbiekben tovább bővíthető. A torpedó játékot azért tartottam jó választásnak, mert a tábla mérete tetszőlegesen módosítható, így a környezet a használt módszer hatékonyságához igazítható. Az implementáció miatt fontos volt, hogy a számítógép-programok eredményeinek helyessége ellenőrizhető legyen, amely egy kis torpedótábla esetében egyszerű. A megvalósítás során igyekeztem különválasztani a környezetet és az ágenseket, így

a környezet alkalmas további módszerek tesztelésére, illetve az ágensek is használhatók más feladatokon. A szakdolgozat végén a Q-tanulás implementációját fogom részletezni.

1. fejezet

A jutalmazott tanulás alapfogalmai

1.1. Mesterséges intelligencia és gépi tanulás

Az emberihez hasonló képességekkel rendelkező gépek vagy programok létrehozását célzó tudomány neve mesterséges intelligencia (artificial intelligence, AI). Ezen tudomány részterülete a gépi tanulás (machine learning, ML), amely gépek vagy programok tanulási folyamaival foglalkozik.

A gépi tanulást az különbözteti meg a számítógép-programozástól, hogy míg a számítógép-programozás esetén a gép egyértelmű, adott helyzetre vonatkozó utasításokat követ, a gépi tanulás folyamán a gép ezek nélkül hoz döntéseket, amelyeket a korábbi tapasztalataihoz igazít. Emiatt a gépi tanulást jellemzően olyan feladatok megoldásánál használják, amikor a legjobb megoldást az emberi programozók sem ismerik, vagy a szokásos módszerek túlságosan számításigényesek.

A gépi tanulásnak három fő ága létezik, a felügyelt tanulás (supervised learning), a felügyelet nélküli tanulás (unsupervised learning) és a megerősítéses vagy jutalmazott tanulás (reinforcement learning). Ezek a fogalmak egyrészt a megoldandó feladat típusára, másrészt pedig a feladatmegoldási módszerekre vonatkoznak. A gyakorlatban az ágak között sok az átfedés, mert előfordul, hogy egy gépi tanulási feladat többféleképpen is megfogalmazható, illetve a módszereket jellemzően kombinálva használják.

1.2. A jutalmazott tanulás

A megerősítéses tanulás az optimalizálási módszerek egyikének tekintető. A megoldandó feladat rendszerint valamilyen érték maximalizálása vagy minimalizálása, ezért a dinamikus programozáshoz hasonló. A két terület legfőbb különbsége, hogy míg az utóbbi esetén a valódi optimum kiszámításához szükséges összes adat a kezdettől fogva rendelkezésre áll, és a feladat részekre bontásával bizonyosan optimális megoldást kaphatunk, a tanulás során ezek az információk nem feltétlenül ismertek, és az optimális megoldás megtalálása nem garantált, a cél ennek közelítése.

A felügyelt tanulással szemben itt nem beszélhetünk tanító- és tesztadatokról. Az úgynevezett ágensnek a rendelkezésre álló adatok mennyiségétől függetlenül döntéseket kell hoznia, amelyekről visszajelzéseket kap és amelyek javíthatják a későbbi működését. Ezeket a visszajelzéseket a környezet szolgáltatja, amelynek feltérképezése hasznos lehet a döntések javításához, ám ez nem közvetlen cél. A felügyelet nélküli tanulással szemben itt nem egy rejtett matematikai struktúrát keresünk, hanem egy minél optimálisabb megoldást egy adott problémára.

Ha az ágens tárol információkat a környezetről, úgy modellalapú, ha nem, akkor modellmentes tanulási módszerről beszélünk. A szakdolgozatomban kizárólag a modellmentes módszerekre koncentrálok. Mivel a megerősítéses tanulást az emberi gondolkodás könnyedén felfogja játékként, sőt egy gyakori alkalmazási terület a mesterséges intelligencia ágensek létrehozása, ezért a jutalmazott tanulást a torpedó játékon próbáltam ki.

1.3. A jutalmazott tanulás elemei

1.3.1. Az ágens

A megerősítéses tanulás egyik kulcsfogalma tehát az ágens (agent). Ez magában foglalja az optimum keresését célzó függvényeket és algoritmusokat, valamint a környezetről esetlegesen eltárolt információkat. Az ágens a gyakorlatban lehet egy robot, egy élőlény, valamilyen fizikai eszköz alkatrésze vagy egy egyszerű programozási objektum is.

1.3.2. A környezet

Az ágensnek képesnek kell lennie érzékelnie a környezet (environment) állapotát és erre vonatkozó döntéseket hoznia az optimalizálási feladatot szem előtt tartva, a környezet pe-

dig a döntésre vonatkozó visszajelzéseket szolgáltat. Ez lehet egy fizikai környezet, például labirintus, de lehet egy eszköz állapotát közvetítő műszer vagy egy elsajátítandó játék, jutalmakkal kiegészítve.

1.3.3. A jutalom

A visszajelzés az ágens egy-egy lépése után, jutalom (reward) formájában történik. Az ágens célja a jutalmak maximalizálása, így a jutalmak meghatározzák az optimalizálási problémát.

1.3.4. Az értékfüggvény

Bár a jutalmak lépésenként érkeznek a környezettől, előfordulhat, hogy egy döntés előnye csak később mutatkozik meg nagyobb jutalomként. A döntések hosszú távú hasznosságát jellemzi az értékfüggvény (value function), amelyet az ágens a különböző időpillanatokban kapott jutalmakból számol. Ez lehet egyváltozós függvény, ahol a változó az állapot, ekkor állapotérték-függvényről beszélünk (state-value function), vagy lehet kétváltozós, ahol az állapot mellett a másik változó az ágens által hozott döntés, ez a döntésérték-függvény (action-value funcion). Ha az értékfüggvény tárolásához táblázatot használunk, akkor tabuláris jutalmazott tanulási módszerről beszélünk.

1.3.5. A stratégia

A stratégia (policy) az az elv, amely meghatározza az ágens viselkedését, egy állapot-döntés hozzárendelés. Ez lehet determinisztikus vagy sztochasztikus, leírhatja például egy függvény vagy táblázat (a stratégiák utóbbi típusát nevezik tabuláris módszereknek is). Az értékfüggvény és a stratégia együttesen jellemzik tanulási módszert.

Egy valódi 10×10-es játék esetén a legcélravezetőbb megoldási módszer valószínűleg a jutalmazott tanulás és egy mesterséges neurális háló kombinálása lenne, ám ennek megvalósítása előtt fontosnak tartottam a lehetséges alapul szolgáló jutalmazott tanulási módszerek megismerését és a környezet átfogóbb vizsgálatát, így a szakdolgozatomban a tabuláris módszerekre koncentrálok.

1.4. A felderítés és kihasználás dilemmája

Az ágens egy-egy döntését alapozhatja a korábbi tapasztalatokra (jutalmakra, értékfüggvényre) vagy hozhat részben vagy teljesen új döntéseket. Nem egyértelmű, hogy hosszú távon melyik eredményez nagyobb jutalmat - a felderítéssel az ágens szuboptimális cselekvését kockáztatjuk, de ez magában hordozza azt a lehetőséget is, hogy jobb eredményt érjen el. Ez az úgynevezett felderítési-kihasználási dilemma (exploration-exploitation dilemma), amely kifejezetten a megerősítéses tanulás jellegzetessége. Minden módszer ezt a két szempontot hivatott kiegyensúlyozni, például valamilyen valószínűségi változó alapján.

1.5. Markov döntési folyamatok

1.5.1. Definíció. *Egy X_n diszkrét sztochasztikus folyamat Markov tulajdonságú, ha teljesül rá az alábbi egyenlet:*

$$P(X_{n+1} = i \mid X_0 = i_0, X_1 = i_1, \ldots, X_n = i_n) = P(X_{n+1} = i \mid X_n = i_n)$$

1.5.2. Definíció. *Egy diszkrét sztochasztikus folyamatot Markov-láncnak nevezünk, ha rendelkezik a Markov-tulajdonsággal.*

Egy ágens és egy környezet interakciója modellezhető egy Markov-lánccal (vagy ezek sorozatával). Egy ilyen Markov-láncot Markov döntési folyamatnak nevezünk.

1.5.3. Definíció. *Tegyük fel, hogy egy környezet és egy ágens a $t = 0, 1, 2, \ldots$ időpillanatokban hatnak egymásra. Jelölje a környezethez tartozó állapotok halmazát $\mathcal{S}$, és jelölje a környezeten az ágens által hozható döntések halmazát $\mathcal{A}$, valamint jelölje $\mathcal{A}(s) \subset \mathcal{A}$ egy konkrét $s \in \mathcal{S}$ állapotból meghozható döntések halmazát. Ezen kívül jelölje $\mathcal{R} \subset \mathbb{R}$ a lehetséges jutalmak halmazát. Ha $S_t \in \mathcal{S}$ a környezet t időpillanatbeli állapota, $A_t \in \mathcal{A}(S_t)$ az ágens ezen időpillanatban meghozott döntése, $R_{t+1} \in \mathcal{R}$ pedig az érte kapott jutalom, akkor az ágens és a környezet interakciója a t időpillanatokban meghatároz egy sorozatot:*

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

1.5.4. Definíció. *Egy Markov döntési folyamat véges, ha $\mathcal{S}$, $\mathcal{A}$ és $\mathcal{R}$ is véges.*

Véges Markov döntési folyamatokban az S_t és R_t valószínűségi változók diszkrét eloszlásúak és csak az őket megelőző állapottól és döntéstől függnek. Ekkor az $S_{t-1}, A_{t-1}, R_t, S_t$

változók sorrendben $s \in \mathcal{S}, a \in \mathcal{A}(s), r \in \mathcal{R}, s' \in \mathcal{S}$ értékeire a következő jelölést vezetjük be:

$$p(s', r \mid s, a) = P(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)$$

1.5.5. Definíció. *A p függvényt a Markov döntési folyamat dinamikájának, a jutalmazási feladat értelmében a környezet dinamikájának nevezzük.*

A megerősítéses tanulás célja a teljes jutalom hosszú távú maximalizálása. Ennek az elvnek a neve jutalomhipotézis.

1.5.6. Definíció. *A maximalizálandó érték neve kumulált jutalom. A t időpillanatot követő kumulált jutalom: $G_t = R_{t+1} + R_{t+2} + \ldots + R_T$, ahol $T \in \mathbb{N} \cup \{\infty\}$.*

1.5.7. Definíció. *Ha az ágens-környezet interakció véges sorozatokra bomlik, azaz a Markov döntési folyamat mindig leáll valamilyen $T \in \mathbb{N}$ időpillanatban, akkor a feladatot epizódikusnak, a T-beli állapotot pedig végállapotnak nevezzük.*

1.5.8. Definíció. *Ha $T = \infty$, feladatot folytatólagos feladatnak nevezzük.*

Folytatólagos feladat esetén a kumulált jutalom, azaz a $G_t = R_{t+1} + R_{t+2} + \ldots$ összeg nem feltétlenül véges. Ezért bevezetünk egy új fogalmat:

1.5.9. Definíció. *A $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ összeget, ahol $0 \leq \gamma \leq 1$, csökkentett kumulált jutalomnak (return), γ-t csökkentési paraméternek nevezzük.*

1.5.10. Megjegyzés. *Ha a jutalmak R_t sorozata korlátos és $\gamma < 1$, akkor a csökkentett kumulált jutalom véges.*

1.5.11. Definíció. *A döntéshozás teljes állapottérre vonatkozó módját stratégiának nevezzük. A π stratégia tehát a következő valószínűségek összességét jelenti az (s, a) állapot-döntés párokra, $(s \in \mathcal{S}, a \in \mathcal{A}(s))$:*

$$\pi(a \mid s) = P(A_t = a \mid S_t = s)$$

1.5.12. Definíció. *Egy π stratégia determinisztikus, ha minden $s \in \mathcal{S}$ állapotra egyértelmű döntést ad, azaz minden $s \in \mathcal{S}$ állapothoz létezik a, melyre $\pi(a \mid s) = 1$.*

Ekkor alkalmazható a $\pi(s) = a$ jelölés.

1.5.13. Definíció. *A π stratégia alkalmazásával nyert, az egyes állapotokhoz, illetve az állapot-döntés párokhoz tartozó kumulált jutalom várható értékét a stratégiára vonatkozó értékfüggvénnyel jellemezzük.*

1.5.14. Definíció. *A π stratégiát jellemző állapotérték-függvény $s \in \mathcal{S}$ esetén:*

$$v_\pi(s) = \mathbf{E}_\pi(G_t \mid S_t = s) = \mathbf{E}_\pi(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s)$$

1.5.15. Definíció. *A π stratégiát jellemző döntésértékfüggvény $s \in \mathcal{S}$ és $a \in \mathcal{A}(s)$ esetén:*

$$q_\pi(s,a) = \mathbf{E}_\pi(G_t \mid S_t = s, A_t = a) = \mathbf{E}_\pi(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a)$$

Egy adott s állapot értékre adott π stratégia esetén rekurzív módon kiszámítható. Az ezt leíró egyenlet a Bellman-egyenlet:[1]

$$\begin{aligned} v_\pi(s) &= \mathbf{E}_\pi(G_T \mid S_t = s) \\ &= \mathbf{E}_\pi(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s) \\ &= \mathbf{E}_\pi(R_{t+1} + \gamma \cdot G_{t+1} \mid S_t = s) \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)(r + \gamma \mathbf{E}_\pi(G_{t+1} \mid S_{t+1} = s')) \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)(r + \gamma v_\pi(s')) \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a)(r + \gamma v_\pi(s')) \end{aligned} \tag{1.1}$$

A Bellman-egyenlet döntésértékfüggvényre vonatkozó változata:[1]

$$\begin{aligned} q_\pi(s,a) &= \mathbf{E}_\pi(R_{t+1} + \gamma v_\pi(S_{t+1} \mid S_t = s, A_t = a) \\ &= \sum_{s', r} p(s', r \mid s, a)(r + \gamma v_\pi(s')) \end{aligned} \tag{1.2}$$

1.5.16. Definíció. *Azt mondjuk, hogy a π' stratégia jobb, mint a π stratégia, ha minden $s \in \mathcal{S}$ esetén $v_\pi(s) \leq v_{\pi'}(s)$.*

1.5.17. Definíció. *Azt mondjuk, hogy a π' stratégia szigorúan jobb, mint a π stratégia, ha van olyan $s \in \mathcal{S}$, melyre $v_\pi(s) < v_{\pi'}(s)$.*

1.5.18. Definíció. *Azt mondjuk, hogy a π^* stratégia optimális, ha jobb, mint bármely más stratégia, azaz $v^*(s) = \max_\pi v_\pi(s)$ minden $s \in \mathcal{S}$ esetén. Ekkor v^* az optimális állapotérték-függvény.*

1.5.19. Tétel. *Mindig van legalább egy optimális stratégia.*[1]

1.5.20. Tétel. *Az optimális stratégia a döntésértékfüggvény alapján is optimális, azaz teljesül, hogy $q^*(s,a) = \max_\pi q_\pi(s,a)$.*[1]

A Bellman-egyenletek optimális stratégiára vonatkozó változatai a Bellman optimalitási egyenletek.

Az optimális értékfüggvényeket a gyakorlatban nehéz kiszámítani, ezért a tanulási módszerek ezek becsléseit célozzák.

A Bellman optimalitási egyenletek a következők:[1]

$$\begin{aligned} v^*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi^*}(s,a) \\ &= \max_a \mathbf{E}_{\pi^*}(G_T \mid S_t = s, A_t = a) \\ &= \max_a \mathbf{E}_{\pi^*}(R_{t+1} + \gamma \cdot G_{t+1} \mid S_t = s, A_t = a) \\ &= \max_a \mathbf{E}_{\pi^*}(R_{t+1} + \gamma \cdot v^*(S_{t+1}) \mid S_t = s, A_t = a) \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)(r + \gamma v_{\pi^*}(s')) \end{aligned} \tag{1.3}$$

$$\begin{aligned} q^*(s,a) &= E_\pi(R_{t+1} + \gamma \max_{a'} q^*(S_{t+1}, a') \mid S_t = s, A_t = a) \\ &= \sum_{s',r} p(s', r \mid s, a)(r + \gamma \max_{a'} q^*(s', a')) \end{aligned} \tag{1.4}$$

Az optimális értékfüggvények kiszámításához szükség van az összes $p(s', r \mid s, a)$ átmenet-valószínűség ismeretére, amelyek a gyakorlatban jellemzően nem állnak rendelkezésre, ha pedig mégis, a folyamat olyankor is számításigényes. A pontos kiszámítás miként-je mégis fontos kiindulópont a tanulási módszerek bevezetésében.

2. fejezet

Dinamikus programozás

Tágabb értelemben a dinamikus programozás olyan matematikai módszert jelent, amelynek célja valamely érték minimalizálása vagy maximalizálása úgy, hogy a feladatot részfeladatokra bontjuk, és az ezekre a részfeladatokra vonatkozó optimumot használjuk fel az eredeti probléma megoldásához. A megerősítéses tanulás kontextusában az optimum az értékfüggvény alapján a lehető legjobb stratégiát jelenti.

A dinamikus programozás során az optimumot közvetlenül számítjuk a Bellmanegyenletből, ezért ez csak véges Markov döntési folyamatokon alkalmazható, és ismerni kell hozzá az összes $p(s', r \mid s, a)$ valószínűséget.

2.1. A stratégiaiteráció

Az egyik dinamikus programozási módszer a stratégiaiteráció (policy iteration). Mivel a feladat az optimális vagy közel optimális stratégia megtalálása, ezért a jutalommaximalizálási feladat két részből áll: a $v(s)$ értékfüggvény vagy a $q(s, a)$ állapotérték-függvény becsléséből, más szóval stratégiaértékelésből vagy predikcióból (policy evaluation), illetve a stratégia javításából (policy improvement). A stratégiaiteráció során e két probléma jól elkülönül egymástól.

2.1.1. A stratégiaértékelés

A stratégiaértékelés célja a $v_\pi(s)$ értékfüggvény megállapítása egy adott π stratégiára. Ismert átmenetvalószínűségek esetén a Bellman-egyenlet egy $|\mathcal{S}|$ változós lineáris egyenletrendszer.

Az iteratív stratégiaértékelés során ennek közelítése numerikusan történik a Bellman-egyenlet módosításával:

$$\begin{aligned} v_{k+1}(s) &= \mathbf{E}_\pi(R_{t+1} + \gamma \cdot v_k(S_{t+1} \mid S_t = s)) \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)(r + \gamma v_k(s')) \end{aligned} \tag{2.1}$$

Ennek az egyenletrendszernek $v_k = v_\pi$ fixpontja, így a v_k sorozatra tetszőleges v_0 esetén érvényes, hogy $\lim_{n\to\infty}\{v_k\} = v_\pi$, ha v_π létezik.[1] Ha $\gamma < 1$ vagy a döntési folyamat a π stratégiát követve véget ér, akkor v_π létezik és egyértelmű,[1] azaz ha ezen feltételek egyike teljesül, a fenti módszer tetszőleges v_0-ból kiindulva végtelen sok lépésben v_π-hez konvergál.

Vezessük be a következő jelölést: $\Delta_{k+1} = \max_{s\in\mathcal{S}}(|v_k(s) - v_{k+1}(s)|)$, ahol $\Delta_0 = 0$.

Ekkor a stratégiaértékelés során tetszőleges $\theta > 0$-val a $\Delta_k < \theta$ leállási feltételt alkalmazva eldönthetjük, hogy a folyamatot $v(s)$ változásának milyen mértékű csökkenéséig végezzük.

2.1.2. A stratégiajavítás

A másik kérdés a π stratégia változtatásának módja, vagyis az, hogy érdemes-e egy adott s állapotban $a \neq \pi(s)$ döntést hozni. Ehhez a régi és az új stratégia állapotértékeinek összehasonlítására van szükség.

2.1.1. Tétel (stratégiajavítási tétel[1]). *Ha π és π' determinisztikus stratégiák, melyek értékfüggvényeire teljesül, hogy $v_\pi(s) \leq q_\pi(s, \pi'(s))$ minden $s \in \mathcal{S}$ állapotban, akkor a π' stratégia jobb, mint π, azaz $v_\pi(s) \leq v_{\pi'}(s)$. Ha egy s állapotban szigorú egyenlőtlenség van, akkor $\pi < \pi'$.*

Bizonyítás. A $q_\pi(s, a)$ döntésértékfüggvény definícióját és a $v_\pi(s) \leq q_\pi(s, \pi(s'))$ feltételt felváltva alkalmazva látszik az egyenlőtlenség:

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbf{E}(R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s) \\
&= \mathbf{E}_{\pi'}(R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s) \\
&\leq \mathbf{E}_{\pi'}(R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s) \\
&= \mathbf{E}_{\pi'}(R_{t+1} + \gamma \mathbf{E}(R_{t+2} + \gamma v_\pi(S_{t+2}) \mid S_{t+1}, A_{t+1} = \pi'(S_{t+1})) \mid S_t = s) \\
&= \mathbf{E}_{\pi'}(R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s) \\
&\leq \mathbf{E}_{\pi'}(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s) \\
&\leq \ldots \leq \mathbf{E}_{\pi'}(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots \mid S_t = s) \\
&= v_{\pi'}(s)
\end{aligned}
\tag{2.2}
$$

□

Ez az egyenlőtlenség kijelöli a stratégia javításának egy lehetséges módját.

2.1.2. Definíció. *A π stratégiából a minden s állapotban egy legnagyobb $q_\pi(s, a)$ értékű a döntés választásával kapott π' stratégiát mohó stratégiának nevezzük:*

$$
\begin{aligned}
\pi'(s) &= \operatorname*{argmax}_a q_\pi(s, a) \\
&= \operatorname*{argmax}_a \mathbf{E}(R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a) \\
&= \operatorname*{argmax}_a \sum_{s', r} p(s', r \mid s, a)(r + \gamma v_\pi(s'))
\end{aligned}
\tag{2.3}
$$

2.1.3. Állítás. *A π stratégia v_π-re nézve mohó megváltoztatása π-nél jobb π' stratégiát ad, nem optimális π esetén pedig szigorúan jobbat.*[1]

Bizonyítás. Az, hogy $v_\pi(s) \leq v_{\pi'}(s) mindens \in \mathcal{S}$, azaz hogy a π' mohó stratégia jobb π-nél, a stratégiajavítási tételből következik.

Azt kell belátni, hogy ha $\pi \neq \pi^*$, akkor $v_\pi(s) \leq v_{\pi'}(s)$, azaz az új, mohó stratégia szigorúan jobb. Ehhez indirekt feltesszük, hogy π egy szuboptimális stratégia, amelynél π' nem szigorúan jobb, azaz az állapotértékeik megegyeznek. Ekkor $v_{\pi'}(s)$ felírható a következő alakban:

$$
\begin{aligned}
v_{\pi'}(s) &= \max_a \mathbf{E}(R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a) \\
&= \max_a \sum_{s', r} p(s', r \mid s, a)(r + \gamma v_\pi(s'))
\end{aligned}
\tag{2.4}
$$

Ez ugyanakkor megegyezik a Bellman optimalitási egyenlettel, azaz a π' stratégia optimális. A $v_\pi = v_{\pi'}$ feltevés miatt π is optimális.

□

Ez azt jelenti, hogy a fenti módszerrel a nem optimális stratégiák ténylegesen javíthatók, emiatt a stratégiajavítás fogalmán jellemzően ezt a mohó módszert szokás érteni. A bizonyítással azt is beláttuk, hogy ha a stratégiajavítás nem eredményez π-nél szigorúan jobb π' stratégiát, akkor π már optimális.

2.1.3. Az iteráció

A stratégiaértékelés tehát a π stratégia v_π állapotérték-függvényére ad becslést, a stratégiajavítás pedig v_π alapján létrehoz π-ből egy legalább olyan jó π'-t. A stratégiaiteráció e két módszer kombinálását jelenti.

Kiindulunk egy $v(s) \in \mathbb{R}$ állapotérték-függvényből és egy $\pi(s) \in \mathcal{A}(s)$ stratégiából, amelyek minden $s \in \mathcal{S}$ állapotra tetszőlegesek. Ezután addig végezzük π értékelését, amíg az új és a régi állapotérték-függvény közötti változás mértéke kisebb nem lesz a becslés kívánt pontosságát jelölő $\theta \in \mathbb{R}^+$ paraméternél. Az így kapott új $v(s)$ szerint mohó módon javítjuk a π-t. Ha a javítás során π változik, akkor visszatérünk a stratégiaértékeléshez és az értékfüggvényt a π-beli döntések alapján javítjuk. A folyamat leáll az első olyan ciklusban, amikor π nem változik.

A véges Markov-féle döntési folyamatokban az állapotok és döntések száma véges, tehát a stratégiák száma is. Mivel a stratégiajavítás nem ront a π stratégián, az értékelés pedig konvergál a hozzá tartozó v_π-hez, ezért az iterációs módszer véges Markov-féle döntési folyamatokra véges idő alatt konvergál az optimális $\pi^*(s)$ stratégiához és a $v^*(s)$ értékfüggvényhez.[1]

2.2. Az értékiteráció

A stratégiaiteráció a jutalmazási feladattól függően lehet kifejezetten gyors, de akár nagyon lassú is. Az értékelés minden egyes futásakor figyelembe kell venni minden s állapotot és az azt követő minden s' utóállapotot.

Ugyanakkor nincs mindig szükség a pontos optimumra, sokszor elég, ha rövidebb futási idő alatt kellően közelítjük az optimális megoldást. A stratégiajavításban minden s állapothoz a legnagyobb várható értékű döntést választjuk. Ha az értékelés során az új $v(s)$

kiszámításához már ezt az a döntést használjuk $\pi(s)$ helyett, akkor új módszert kapunk.

Ekkor az ismételt stratégiajavítás elhagyható. Az értékelés során tulajdonképpen feltételezzük az előnyösebb döntéseket és v-t ez alapján változtatjuk. Ha v_k és v_{k+1} kellően közel vannak egymáshoz, azaz az értékelés véget ér, akkor a stratégiát egyszer változtatjuk mohó módon és az így kapott v-t és π-t adjuk vissza, ami jelentősen csökkenti a futási időt. Ennek a módszernek a neve értékiteráció.

2.2.1. Tétel. *Ez a módszer szintén konvergál a $\pi^*(s)$ stratégiához és a $v^*(s)$ értékfüggvényhez.*[1]

2.3. Egyéb dinamikus programozási módszerek

A fenti módszerekben az értékfüggvény becslésének minden módosításakor minden állapotra számításokat végzünk. A gyakorlatban erre nincs mindig szükség, hiszen vannak állapotok, amelyekre az értékfüggvény becslésének pontossága fontosabb, mint másokén. A konvergencia sebességének szempontjából számít, hogy az értékelés milyen sorrendben halad végig az állapotokon. Azokat a dinamikus programozási módszereket, amelyekben az állapotértékekek becsléseit nem azonos gyakorisággal változtatjuk vagy az állapotok iterációs sorrendjét a konvergencia javításának céljából módosítjuk, aszinkronikus dinamikus programozási módszereknek nevezzük.

A dinamikus programozási módszerek még nem nevezhetők tanulási módszernek, mert nem történik ágens-környezet interakció, csupán függvények numerikus becslése előre meghatározott $p(s', r \mid s, a)$ valószínűségek alapján. Az aszinkronikus módszerek viszont használhatók együtt valódi tanulási módszerekkel.

Bár a dinamikus programozás nagy állapottereken lassú és számításigényes algoritmusokat jelent, ennek elmélete fontos kiindulópont a tanulási módszerek működésében. Az általános értelemben vett stratégiaiteráció vagy kontrollfeladat, azaz az értékfüggvény és a stratégia egymást követő ismételt módosítása jellemző alapjául szolgál a valódi jutalmazott tanulási módszereknek.

3. fejezet

Monte Carlo módszerek

A valódi tanulási módszerek során nincs szükség a környezet $p(s', r \mid s, a)$ ismeretére. Egy ágens képes kizárólag a környezettel való interakció alapján tanulni. Ez az interakció lehet valós vagy szimulált. Ha a $p(s', r \mid s, a)$ valószínűségek nem feltétlenül állnak rendelkezésre, de az ezek eloszlására vonatkozó modell igen, akkor az interakció az eloszlásból vett mintával szimulálható. Ekkor az ágens inputját szimulált tapasztalatnak, ellenkező esetben valódi tapasztalatnak nevezzük.

3.1. Az állapotértékek becslése

Ha a környezet dinamikáját nem ismerjük, akkor a Bellman-egyenlet $v_\pi(s)$ állapotértékre vonatkozó változatai közül a G csökkentett kumulált jutalmat tartalmazóból tudunk kiindulni, azaz a t időpillanatbeli S_t állapotot követő G_t csökkentett kumulált jutalom várható értékét kell becsülni. A Monte Carlo módszerek során ezt a tapasztalatok mintaátlagával végezzük az adott $S_t \in \mathcal{S}$ állapotra.[1] Egy epizódon belül a $v(S_t)$ állapotérték javítása történhet az állapot epizódon belüli első bejárásakor vagy mindegyiken. A módszer első változatát első bejárásos (first visit) Monte Carlo predikciónak, a másodikat teljes bejárásos (every-visit) predikciónak nevezzük. Mindkét változat végtelen lépésben konvergál $v_\pi(s)$-hez.[1]

Ehhez szükség van a tapasztalat generálására. Míg a stratégiaiértékelés $v(s)$ értékét az állapottéren való végigiterálással változtatja, a tanulás során ez az ágens és a környezet között létrejövő $S_0, A_0, R_0, S_0, A_0, R_0, \ldots R_T$ epizód által kijelölt állapotokon zajlik, mégpedig $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ miatt fordított sorrendben.

A G_t csökkentett kumulált jutalom egy adott epizód t időpillanatbeli átmenetére vonatkozik, az s-hez tartozó tapasztalati átlag pedig értelemszerűen az összes epizódbelire. Az s állapothoz tartozó G kumulált jutalmak sorozatát $Returns(s)$-sel jelölve $v(S_t)$ változtatása a következőképpen történik:

$$\begin{aligned}&\text{Append } G_t \text{ to } Returns(S_t)\\&V(S_t) \leftarrow \text{avg}(Returns(S_t))\end{aligned}$$

A stratégiaértékeléshez képest a Monte Carlo predikciót lassítja az átlag kiszámítása, ezért ezt érdemes inkrementálisan végezni, amikor nem tartunk fent listát minden állapot minden csökkentett kumulált jutalmához.

Ha $G^n(s)$ jelöli a $Returns(S_t)$ n. elemét, akkor a t. lépésben a $Returns(S_t)$ átlaga a következő:

$$\begin{aligned}\frac{1}{n}\sum_{i=0}^{n} G^i(s) &= \frac{1}{n}(G^n(s) + \sum_{i=0}^{n-1} G^i(s))\\&= \frac{1}{n}(G^n(s)) + \frac{1}{n}\sum_{i=0}^{n-1} G^i(s)\\&= \frac{1}{n}(G^n(s)) + \frac{n-1}{n}\cdot\frac{1}{n-1}\sum_{i=0}^{n-1} G^i(s)\end{aligned} \tag{3.1}$$

Az $\frac{1}{n-1}\sum_{i=0}^{n-1} G^i(s)$ érték megegyezik a $Returns(S_t)$ korábbi átlagával, azaz $V(S_t)$ módosított értéke egyetlen lépésben kiszámítható a korábbi avg$Returns(S_t)$-ből, ennek n méretéből és az új G_t csökkentett kumulált jutalomból. Ezzel a módszer jelentősen gyorsul.

A fenti képletben a $Returns(S_t)$ átlagában a csökkentett kumulált jutalmak egyszerű számtani átlaga szerepel. A gyakorlatban a G csökkentett jutalmakat az R jutalmakhoz hasonlóan szokás az idő függvényében súlyozni, az idő szó azonban itt nem az epizód időlépésére, hanem az értékadás sorszámára vonatkozik.

Jelölje most $V_n(S_t)$ az optimális v^* n. becslését az S_t állapotban. Tekintsük most a V_n alakulálását ebben a konkrét pontban, azaz jelölje $V_n = V_n(S_t, A_t)$. Így a fenti összefüggés a következő:

$$V_{n+1} = V_n + (1 - \frac{1}{n})(G_n - V_n)$$

Ekkor $\frac{1}{n}$-et egy $\alpha \in (0,1]$ konstanssal helyettesítve v^* korábbi becsléseinek egy súlyozott átlagát kapjuk:[1]

$$\begin{aligned} V_{n+1} &= \alpha V_n + (1-\alpha)(G_n - V_n) \\ &= \alpha G_n + (1-\alpha)V_n \\ &= \alpha G_n + (1-\alpha)(\alpha G_{n-1} + (1-\alpha)V_{n-1}) \\ &= \alpha G_n + (1-\alpha)\alpha G_{n-1} + (1-\alpha)^2 V_{n-1} \\ &= \dots \\ &= (1-\alpha)^n V_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} G_i, \text{ ahol} (1-\alpha)^n + \alpha(1-\alpha)^{n-i} = 0. \end{aligned} \tag{3.2}$$

Az α paraméter neve tanulási ráta (learning rate, step size parameter). Ez a paraméter minden G_i csökkentett kumulált jutalomhoz egy i-től függő $\alpha(1-\alpha)^{n-i}$ súlyt rendel. Mivel $1-\alpha < 1$, ezért az átlagban a csökkentett kumulált jutalmak értéke annál kisebb, minél régebbi iterációból származik.

3.2. Monte Carlo kontroll

A valódi tanulási feladatok során nem állnak rendelkezésre az átmenetvalószínűségek. Egy π stratégia minőségéről az s állapotok $v_\pi(s)$ értékeinél több információt adnak az (s,a) állapot-döntés párokra vonatkozó $q_\pi(s,a)$ értékek, ezért a döntésértékfüggvény alkalmazásának lehetősége általában fontos a tanulási módszereknél.

Az általánosított stratégiaiteráció döntésérték-függvényekre vonatkozó változatában a q^* optimális értékfüggvényt becsüljük és ez alapján javítjuk a π^* optimális stratégiát becsülő π-t.

3.2.1. Állítás. *A stratégiajavítás a $q(s,a)$ értékfüggvény felhasználásával is végrehajtható.*[1]

Bizonyítás. A mohó stratégia továbbra is a determinisztikus $\pi(s) = \operatorname{argmax}_a q(s,a)$ stratégiát jelenti. Jelölje π_{k+1} a π_k stratégiából és a q_{π_k} értékfüggvényből mohó módon kapott stratégiát. Ekkor minden $s \in \mathcal{S}$-re igaz a következő összefüggés:

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \operatorname*{argmax}_a q_{\pi_k}(s, a)) \\ &= \max_a q_{\pi_k}(s, a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s) \end{aligned} \tag{3.3}$$

Az egyenlőtlenségből a stratégiajavítási tétellel következik, hogy π_{k+1} jobb stratégia π_k-nél, egyenlőség pedig csak akkor fordulhat elő, ha π_k optimális. □

Tehát bevezethetjük a Monte Carlo módszerek döntésértékekre vonatkozó változatát. Ekkor q a következőképpen változik:

$$\begin{gathered} \text{Append } G_t \text{ to } \mathit{Returns}(S_t, A_t) \\ Q(S_t, A_t) \leftarrow \text{avg}(\mathit{Returns}(S_t, A_t)) \end{gathered}$$

Ha idővel minden állapot-döntés párt bejárunk, akkor a q és π becslései az állapotérték-függvényre épülő módszerhez hasonlóan konvergálnak.[1] Az, hogy ez a bejárás megtörténik, valójában egy feltételezés a kontrollfolyamat során létrejövő stratégiákra, amelyet felfedező kezdeteknek is szokás nevezni.

Ugyanakkor ha az értékelendő π stratégiában előfordul olyan (s, a) állapot-érték pár, amely a jutalmazási feladat alapján megengedett, azaz $a \in \mathcal{A}(s)$, viszont $\pi(a|s) = 0$, akkor az epizódok generálásakor ez a pár soha nem fog megjelenni, így a hozzájuk tartozó döntésérték változatlan marad. Ennek a legszélsőségesebb példája egy π determinisztikus stratégia. Ez esetben a predikció minden s állapotból az egyetlen $\pi(s) = a$ döntésen ismétlődik és a párhoz tartozó $q(s, a)$ érték sem változik. A felderítés és kihasználás dilemmája éppen azt jelenti, hogy nem elég, ha mindig az értékfüggvény jelenlegi becslésének megfelelő döntést hozzuk, mert ezzel elkerülhetünk egy pontosabb értékfüggvénybecslés szerint jobb lépést. A felderítést biztosíthatja például, ha az ágens döntései a mohó stratégiától valamilyen kis valószínűséggel eltérnek.

Ahhoz tehát, hogy a kontroll során a döntésértékfüggvényt az állapotérték-függvényhez hasonló módon becsülhessük, mindenképpen szükség van a stratégiára vonatkozó megkötésre.

3.2.2. Definíció. *A π stratégia ε-greedy a q értékfüggvényre nézve, ha valamilyen kis valószínűséggel véletlen döntést hoz, különben q-ra nézve mohó, azaz ha van olyan $\varepsilon > 0$,*

amelyre minden $s \in \mathcal{S}$ állapot és belőle hozható minden $a \in \mathcal{A}(s)$ esetén, az s-ből mohó lépést a^*-gal jelölve:

$$\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|}, & ha\ a = a^*, \\ \frac{\varepsilon}{|\mathcal{A}(s)|}, & ha\ a \neq a^* \end{cases}$$

3.2.3. Definíció. *π egy ε-soft stratégia, ha van olyan $\varepsilon > 0$, amelyre minden $s \in \mathcal{S}$ állapot és a minden belőle hozható $a \in \mathcal{A}(s)$ döntés esetén teljesül, hogy $\pi(a \mid s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|.}$*

3.2.4. Állítás. *Ha π egy $\varepsilon-soft$ stratégia, π' pedig a q_π-re nézve egy $\varepsilon-greedy$ stratégia, akkor π' a π stratégia egy javítása.*[1]

Bizonyítás. Belátjuk, hogy $v_\pi(s) \leq q_\pi(s, \pi'(s))$.

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_a \pi'(a \mid s) q_\pi(s, a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \max_a q_\pi(s, a) \\ &\geq \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \sum_a \frac{\pi(a \mid s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1 - \varepsilon} q_\pi(s, a) \\ &= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s, a) + \sum_a \pi(a \mid s) q_\pi(s, a) \\ &= v_\pi(s) \end{aligned} \tag{3.4}$$

A stratégiajavítási tételből következik, hogy $\pi \leq \pi'$. □

3.2.5. Állítás. *Egyenlőség pontosan akkor van, ha π és π' optimális stratégiák az ε-soft stratégiák között.*[1]

A két állítás azt jelenti, hogy a stratégiajavítás tetszőleges ε-soft stratégiából kiindulva ε-greedy javítással alkalmazható döntésérték-függvényekre.

3.3. További módszerek

Az eddigi Monte Carlo módszerek az általánosított stratégiaiterációra épültek. A folyamat során egyetlen π stratégiát tartottunk számon, melynek értékelése az ez alapján tett lépéssel történt. A felderítés biztosításához ez a stratégia eltért a mohó stratégiától, így viszont egy nem optimális stratégia értékeit használtuk az optimális stratégia értékeinek

becslésére. A kontrollra egyetlen stratégiát használó módszerek elnevezése egystratégiás (on-policy) módszer. (Itt természetesen nem egyetlen stratégiáról van szó, hanem stratégiák egyetlen sorozatáról.) Ugyanakkor léteznek kétstratégiás (off-policy) módszerek is. Az ágens az egyik stratégia alapján lép a környezetben és az így kapott jutalmakkal becsli a másik stratégia értékeit. Az egystratégiás módszerek tekinthetők ennek egy speciális változatának, amelyekben a két stratégia megegyezik. A kettős stratégiajavításra a temporális differencia módszerek közül látunk majd példát.

4. fejezet

Temporális differencia módszerek

A dinamikus programozás és a Monte Carlo módszerek elveinek ötvözésén alapulnak a temporális differencia módszerek. A Monte Carlo módszerekhez hasonlóan itt sincsen szükség a környezet dinamikájának ismeretére, a temporális differencia ágensek is tapasztalatból tanulnak. A dinamikus programozáshoz hasonlóan a predikció az értékfüggvény korábbi becsléseinek módosításával történik az epizód végkimenetelének figyelembe vétele nélkül.

Az elnevezés arra utal, hogy egy adott S_t állapot vagy (S_t, A_t) állapot-döntés párhoz tartozó értékfüggvényt a következő néhány lépés után kapott jutalmakból és az utána következő csökkentett kumulált jutalom becsléséből számítjuk. Az $n+1$-lépéses temporális differencia ($TD(n)$) elnevezés arra utal, hogy egy adott állapot (vagy állapot-döntés pár) után hány időlépésig használjuk a csökkentett kumulált jutalmakat az értékfüggvény becslésében. A predikció ezen módja visszavezethető a Monte Carlo predikcióra.

4.1. A predikció

Az elsőbejárásos, kétstratégiás Monte Carlo predikció döntésérték-függvényre vonatkozó változatában $Q(s, a)$ a következőképpen változik:

$$Q(S_t, A_t) \leftarrow \alpha G_t + (1 - \alpha) \cdot Q(S_t, A_t)$$

G_t a t időpillanatot követő csökkentett kumulált jutalmat jelenti az epizódon belül. Tehát a tagok számára nincs megkötés, az csupán az epizód lépésszámától függ, így a Monte Carlo predikció tekinthető ∞-lépéses predikciónak.

A temporális differencia módszerek a fenti szabály módosításával kapott predikcióra épülnek. Az egylépéses predikcióban a G csökkentett kumulált jutalom helyett egyetlen csökkentett jutalom és a későbbi csökkentett kumulált jutalmak becslése, azaz $Q(s, a)$ szerepel:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

4.2. SARSA

A fenti predikcióban a döntésérték-függvény javításához szükség van egy t lépéshez tartozó $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$ sorozatra. Erről kapta a nevét az egystratégiás temporális differencia kontroll (on-policy TD-control), vagyis a SARSA-módszer.

A kontrollfolyamat a stratégiaiteráció szerint történik. A stratégia a Q döntésérték-függvény szerint változik, például ε-greedy módon. A kiinduláshoz szükség van egy $\alpha \in (0, 1]$ és egy $\varepsilon > 0$ paraméterre. $Q(s, a)$ minden $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ esetén tetszőleges, kivége a T végállapotokban, ahol $Q(S_T, .) = 0$. Az ágens minden epizódban egy tetszőleges kezdőállapotból a Q értékfüggvényből kapott π stratégia alapján választ A lépést, majd a kapott R, S', A' szerint változtat $Q(S, A)$-n. A ciklus az (S', A') párból folytatódik a végállapotig, amikor új epizód indul.

4.3. Q-tanulás

A SARSA-tanulás egy széles körben használt módosítása a Q-tanulás. A komplexebb jutalmazott tanulási módszerek, például a mély megerősítéses tanulás igen jelentős része épül a Q-tanulásra.

A Q-tanulás a predikció kis részletében tér el a SARSA-tól, de ez jelentős elvi különbséget eredményez. Az értékfüggvény módosítása a következő:

$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max_a Q(S', a) - Q(S, A))$

Tehát a predikció a $Q(S', A')$ helyett az S'-ből lehetséges legjobb döntés becsült értékével történik, azaz a Q-tanulás egy kétstratégiás módszer.

A Q-tanulás hátránya is éppen az, hogy a döntésértékeket sokszor egy sokkal jobb döntés értékével becsüljük.[1] Ennek kiküszöbölését célozza a kettős Q-tanulás, amelyben egy Q_1 és egy Q_2 függvényt tartunk számon és ezek közül minden lépésben véletlenszerűen frissítjük valamelyiket.

Mind a SARSA-nak, mind a Q-tanulásnak léteznek n-lépéses változatai. A tanulási módszerek közül az egylépéses Q-tanulást implementáltam a torpedó játékra.

5. fejezet

Q-tanulás a torpedó környezetben

Az eredeti torpedó egy stratégiai játék, amelyet két játékos játszik. Mindkét játékosnak van egy hajóállása, amelyet a másik nem lát. A játékosok felváltva igyekeznek lelőni egymás hajóit, és az nyer, aki előbb kilőtte az ellenfél összes hajóját.

5.1. A játék szabályai

A játékot általában 10×10-es táblán játsszák, de a tábla mérete és a hajók típusai változóak lehetnek. A két játékos először két külön táblán elhelyezi a hajóit, azaz bejelöli, hogy a tábla mely celláit foglalják el a hajók. A hajók alakjai és méretei adottak, azaz meg van adva, hogy mely hajó hány cellát foglal el és ezek hogyan helyezkednek el egymáshoz képest. A hajókat elhelyezéskor el szabad forgatni, de minden hajót el kell helyezni a táblán és egy cellát legfeljebb egy hajó fedhet. Az elhelyezendő hajók típusai a két játékos esetében azonosak, azaz nem lehet, hogy az egyik játékosnak van olyan hajója, amilyen a másiknak nincs. A hajók egy elhelyezkedését a táblán flottának nevezem.

Ezután a játékosok elkezdhetik lelőni egymás hajóit. Egy lövés egy olyan cella megjelölését jelenti, amelyről a játékos úgy gondolja, hogy az ellenfél egyik hajója fedi. Az ellenfél ekkor megmondja, hogy van-e a cellán hajó a saját tábláján vagy nincs. Ez az információ segíthet következtetni a hajók elhelyezkedésére és további cellákat kiválasztani a későbbi fordulókban. A lövések jelölhetők egy kezdetben üres négyzeterácson.

A játék egyik változatában felváltva lehet tippelni, míg egy másikban a játékosok minden talált lövése után jár egy újabb lehetőség, mielőtt az ellenfél következne. Ezen kívül van olyan változat, amelyben a játékosok nemcsak a talált cellákról, hanem arról is kapnak információt, ha egy hajó elsüllyedt-e, azaz érte-e már találat a hajó minden celláját.

Ekkor az is kiderül, hogy pontosan melyik hajóról van szó és pontosan hol helyezkedik el. Erre akár úgy is gondolhatunk, hogy az adott hajó kirajzolódik a jelölésre használt négyzetrácson. Az győz, aki elször elsüllyeszti az ellenfél összes hajóját.

A játékban jellemzően nem megengedett többször ugyanarra a cellára tippelni, de ez egyébként sem áll a játékosok érdekében, hiszen nem szolgálna új információval és lassítaná a hajók által foglalt cellák megtalálását.

5.2. A torpedó mint jutalmazási feladat

A játékról többféle feladat is megfogalmazható: egyrészt a torpedó több változata közül is választhatunk, másrészt feladat lehet minél jobb tippelési stratégia keresése, valamint a hajók minél előnyösebb elhelyezése. Ez utóbbi egyáltalán nem egyértelmű feladat és nagyban függ az ellenféltől.

A torpedóban az nyer, aki először elsüllyeszti az összes hajót, tehát a cél az, hogy ezt minél kevesebb lépésben megtegyük. Ez valójában az elhelyezéstől független feladat. Akármennyi lépést is nyerünk egy ügyes elrendezéssel, ezen lépések alatt meg kell találni a nemüres cellákat az ellenfél tábláin, ezért kizárólag a jó stratégia keresésével foglalkoztam. Bár az itt használt implementációban a környezet ad visszajelzést egy-egy hajó elsüllyesztéséről, ezért az ágenst most nem jutalmaztam, mert az állapottér enélkül is igen nagy. Ezért a feladat felfogható a játék azon változataként, amelyben a hajók utolsó cellájának elsüllyesztését nem jelezzük.

Mivel az ellenfél kiismerése bonyolultabb feladat, ezért hajók elhelyezése véletlenszerű, pontosabban álvéletlenszerű lesz. Erre tekinthetünk úgy, hogy az ágens egy másik, a hajókat véletlenszerűen elrendező és nagyon gyengén tippelő játékos ellen játszik. (A valóságban az emberi játékosok közül még a kezdők elrendezései sem véletlenszerűek - a legtöbben akaratlanul is azokat az elrendezéseket választják, amelyek a középpontos szimmetriára emlékeztetnek. Ismert stratégia, hogy egy süllyesztés után az elsüllyedt hajót képzeletben középpontosan tükrözzük, majd az így kijelölt, még felderítetlen cellák valamelyikét választjuk.)

5.2.1. Az állapotok és a környezet

Adott lesz tehát a hajók egy álvéletlenszerű elrendezése és egy kezdetben üres tábla, az ágens pedig minden lépésben dönt, hogy melyik cellára lőjön. A cél az, hogy a hajók által

elfoglalt cellákat minél kevesebb lépésben eltalálja.

A hajók elrendezése és a tábla együttesen alkotják a környezetet. Ha a táblán különböző karakterek jelölik a felderítetlen, a talált, valamint a korábban már választott, de el nem talált cellákat, akkor a tábla az összes információt tartalmazza, amelyet az ágensnek össze kell kötnie a kapott jutalmakkal, ezért a tábla jelenti a környezet állapotát. Ha az ágens minden hajót elsüllyesztett, akkor az adott játékmenet véget ér és a tábla minden cellája felderítetlenre vált. Ezért ez a feladat epizódikusnak tekinthető.

A feladatkitűzésben a játék állásait az állapotoknak, az ágens által kijelölt cellákat a döntéseknek feleltettük meg. A döntések egyértelműen kijelölik a következő állapotokat, ezért nemcsak döntésértékfüggvényeket, hanem állapotérték-függvényeket is használhatunk. Azért, hogy az ágensek rugalmasak legyenek és később más feladatokra is lehessen őket használni, döntésértékfüggvényeket használtam.

5.2.2. A jutalmazás

A következő kérdés a jutalmazás módja. Azt szeretnénk, ha az ágens eltalálná a hajókat, ezért felmerülhet, hogy a környezet a jutalmat a talált cellákért adja, például úgy, hogy mindent talált cella (vagy a süllyesztett hajókat jelző változatban hajókért) egy pontot, egy nem talált tippelés pedig nulla pontot érjen. Ekkor viszont a jutalommaximalizálás elve sérülne, hiszen minden játékmenetért ugyanannyi összjutalom járna, nevezetesen a hajók által foglalt cellák számának összege (vagy a hajók száma). Ezzel az ágens számára nem lenne különbség a különböző játékmenetek, illetve az egyes állapot-döntés párok között. Ugyanez a probléma akkor is, ha az összes hajó elsüllyesztését jutalmazzuk.

Mivel a cél a hajók elsüllyesztése minél kevesebb lépésben, ezért a lépésszámot kell minimalizálni. Ha a sikertelen tippekért negatív jutalmat adunk, a sikeresekért pedig nullát, akkor a jutalom maximalizálása megfelel a lépésszám minimalizálásának.

Azt, hogy az ágens egy cellát többször is kiválasszon, többféleképpen is kiküszöbölhetjük: beleírhatjuk az ágens forráskódjába vagy a tanítási folyamat részévé tehetjük. A megerősítéses tanulásban gyakran a próba-szerencse (trial and error) alapú megközelítés a kívánatos, amikor minél kevesebb előzetes információval látjuk el az ágenst és a szabályokat jutalmak formájában fogalmazzuk meg. A hangsúly általában a tanításon van, nem pedig az előzetes emberi ismeretek megfogalmazásán. Ennek előnye, hogy kevesebbet kell kódolni, hátránya, hogy a tanulási folyamat tovább tart és a módszertől függően időről időre előfordulhat, hogy az ágens mégis hibásan lép. Esetünkben a cellaismétlés

elkerülését erősíthetjük azzal, hogy egy már látogatott cella választását nagy büntetéssel sújtjuk, például úgy, hogy a környezet az összes cella számával ellentétes jutalmat ad érte, majd az epizód megszakad. Ugyanakkor a feladatkitűzés miatt az ismétlések száma idővel egyébként is csökken, hiszen hosszú távon minden ismétlés csökkenti a kumulált jutalmat.

5.3. A környezet implementációja

A számítógépes implementáció legnagyobb kihívását a környezet implementációja jelentette. Abból a célból, hogy a projektet a későbbiekben további módszerekkel egészíthessem ki és ezek megvalósításához minél többféle eszköz a rendelkezésemre álljon, a forráskódokat Python nyelven írtam.

5.3.1. A környezet

Tehát a torpedó környezethez tartozik egy flotta és legalább a jelenbeli állapot, vagyis a játék egy állása. Ezen kívül tartalmazza a játék paramétereit, azaz a tábla méretét és a hajók típusait, valamint a jutalmak listáját. Az epizódok elején a környezet hozza létre az álvéletlen flottát, a későbbiekben pedig az ágens döntése szerint vált az állapotok között.

5.3.2. A hajók és a flották

A flotta egy konkrét hajóját jellemzi a típusa vagy alakja, az elforgatottsága, a táblán való konkrét elhelyekedése (azaz az általa elfoglalt indexek listája), illetve az, hogy már elsüllyedt-e. Egy flottához tartozik a rajta szereplő hajók listája és az ezek által együttesen kiadott hajóállás.

5.4. Az ágens-környezet interakció

A torpedó gyakorlati megvalósításában az ágens-környezet interakciónak két módját fontoltam meg Az egyik esetben az interakció közvetlenül a táblán keresztül zajlik úgy, hogy az ágens valamilyen stratégia alapján lő egy cellára (azaz megad egy koordinátapárt), amelyre a környezettől visszajelzésként egy logikai értéket kap (talált vagy sem), illetve egy jutalmat. A másik lehetőség, hogy minden állapothoz és minden döntéshez tartozik egy azonosítószám. Az ágens egy döntésazonosítót ad a környezetnek, a környezet pedig egy jutalmat és egy állapotazonosítót ad vissza.

Bár a kétféle megvalósítás között az ismertetett tanulási módszerek szempontjából csak jelölésbeli különbség van, a táblázat létrehozásához szükség volt a döntések és állapotok leszámlálására, amelyből az azonosítók természetesen adódtak.

5.5. A Q-tábla létrehozása

A $Q(s, a)$ értékfüggvény táblájához szükség volt $\mathcal{S}$ és $\mathcal{A}$ méreteire. A $\mathcal{A}(s)$ ezúttal minden $s \in \mathcal{S}$-re állandó. A lépések szabályos módjának elsajátítása most része a jutalmazási feladanak, ezért bármelyik döntés bármelyik nem végállapotból meghozható, így ezek száma megfelel a cellák számának.

A lehetséges állapotok száma már nem ennyire egyértelmű. Csak egy olyan tábla része az állapottérnek, amely mögött érvényes flotta van.

5.5.1. A flották előállítása

Először tehát a flottákat kell előállítani. Ez azt jelenti, hogy az összes hajótípus összes elhelyezkedése által meghatározott flottákat össze kell gyűjteni, feltéve, hogy ezek a flották érvényesek, azaz a hajók ténylegesen a táblán helyezkednek el és semelyik kettő sem fed azonos cellát. A hajók elhelyezkedését meghatározza a hajók típusa, kezdőcellája és pozíciója, azaz a forgatása.

Ez meghatároz egy fát, amelynek szintjei a hajótípusok, pontjai pedig a hajótípuson belül a konkrét hajók, amelyek a pozíciójukban és kezdőpontjukban térhetnek el egymástól. Ebben a gráfban egy hajót jelentő pont többször is szerepel az adott szinten, mert egy hajótípus minden hajóját variálni kell a többi hajótípus minden hajójával (persze a listába csak akkor kerül bele, ha nincsen fedés). Ez felfogható úgy is, hogy hajófajták szintjei között van egy másik szint, amelyet a pozíció szerinti elágazás köt össze a típus szintjének pontjaival, az itteni pontból pedig a kezdőpont szerint van elágazás a következő típus szintjének pontjaival.

Tehát ezt a képzeletbeli fát kell bejárni. Egy flottában minden hajótípusnak szerepelnie kell, ezért a flották a leveleken fognak megjelenni, tehát érdemes mélységi keresést használni. Ugyanakkor egyik hajónak sem lehet közös cellája egy másikkal, ebben az esetben felesleges a további szintekről hajókat illeszteni a táblára, visszatérünk az eggyel feljebbi hajótípus szintjére, azaz töröljük a tábláról a be nem illeszthető hajót. Így a flották listáját egy backtracking algoritmus fogja bejárni. Azért, hogy minél kevesebb legyen a felesleges

(flottához nem vezető) lépés, érdemes a hajótípusokat méret szerint csökkenő sorrendben listázni, így a nagyobb méretűek közelebb kerülnek a gyökérhez. A backtracking rekurzív és iteratív módon is történhet, de a Python nyelv sajátosságai miatt a gyakorlatban csak a rekurzió működött.

5.5.2. Az állapotok előállítása

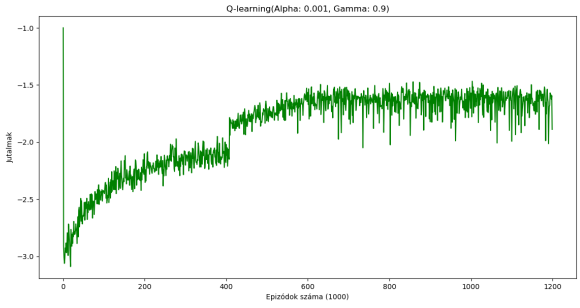
Az állapotok előállítása a flottákból történik. Az állapotokhoz tartozik egy azonosító és a megjelenítésükre szolgáló tábla, valamint a stratégiaiterációhoz használható egyéb attribútumok.

5.6. A tanítás

A Q-tanulást egymillió lépésig futtatam. Az inkrementumok súlyának $\alpha = 0.001$-et adtam meg, a γ csökkentési rátaként 0.9-et használtam. A stratégiajavítás ε paramétere 1-től 0.01-ig változott.

5.7. Eredmények

Az ágens valóban megközelítette a 2×2-es táblán optimális stratégia -1.5 várható értékét. A tanulási folyamat már 3×3-as táblán, egyetlen hajóval is reménytelenül lassú volt. Ezzel szemben a stratégiaiteráció és az értékiteráció mindössze néhány másodperc alatt kiszámították az optimális stratégiát. Tehát nem lehet kijelenteni, hogy a tanulási módszerek mindig jobbak lennének a dinamikus programozásnál. A későbbiekben persze érdemes lehet a tanítást különböző hiperparaméterekkel megismételni, de valószínűtlen, hogy a Q-tanulás ezen a példán megközelítse a dinamikus programozás hatékonyságát.



Irodalomjegyzék

[1] RICHARD S. SUTTON AND ANDREW G. BARTO, *Reinforcement Learning: An Introduction*, The MIT Press, 2020