Autonóm ágens tanítása megerősítéses tanulás alkalmazásával autópályán való optimális közlekedéshez

Szőke László*. Aradi Szilárd**

**Budapesti Műszaki és Gazdaságtudományi Egyetem, MSc hallgató,*
(e-mail: szoke.laszlo95@gmail.com).
*** Budapesti Műszaki és Gazdaságtudományi Egyetem, egyetemi adjunktus,*
(e-mail: aradi.szilard@mail.bme.hu)

Absztrakt: A mesterséges intelligencia és gépi tanulás napjainkban egyre jelentősebb szerepet tölt be, a világ szinte minden területén megjelenik, hogy a legkülönbözőbb problémákra, feladatokra megoldást találjanak. Az autóipar is nagy erőkkel próbálja felhasználni a tanuló algoritmusokat a legnagyobb kihívást jelentő absztrakt problémák megragadására és megoldására. A gépi látás, az objektum felismerés, különböző irányítási feladatok megfelelő adat és kód segítségével, részben vagy egészben megoldhatók. Munkánk során egy olyan tanuló algoritmust hoztunk létre, mely képes megtanulni szimulált autópályán való közlekedést, és ebben a környezetben úgy navigálni, hogy azon baleset nélkül végig tudjon haladni az általa irányított járművel. Az algoritmus egy Multi Layer Perceptron struktúrával megépített neurális háló segítségével és megerősítéses tanulást használó algoritmussal alkottuk meg. A kialakított szimulációs környezet egy nyílt forráskódú forgalomszimulátorral való integrált Python kóddal reprodukálja a folyamatosan változó autópályás forgalmat. Több vezetési stílussal rendelkező vezetőt és autót implementáltunk, hogy minél nagyobb véletlenszerűséget biztosítsunk a tanulás folyamán. A cikkben először a projekt implementálása során felmerülő nehézségek és a tanulási folyamat során megfigyelt tanulást lassító vagy megakadályozó hibák kerülnek megfogalmazásra, majd a különböző forgalmi helyzetekben elért eredményeket ismertetjük.

1. BEVEZETÉS

Napjainkban a világ minden területén egyre nagyobb hangsúlyt kap a mesterséges intelligencia és gépi tanulás, illetve ezeknek a legkülönbözőbb feladatokra történő alkalmazása. (Carbonneau, Laframboise and Vahidov, 2008) már igen korán vizsgálta a gépi tanulás alkalmazását a logisztikában, illetve ellátó láncok ingadozásának előrejelzését, optimalizálását. Hasonlót jelenleg az Amazon és egyéb nagy beszállító cégek is használnak a rendelések optimális kezelésére, előre jelzésére és a raktár felkészítésére. Mások az egészségügyet célozták meg: pontosabb betegségfelismerésre és tünetkezelésre alkottak modellt (Abdelaziz *et al.*, 2018). Folyamatosan jelennek meg újabb és újabb algoritmusok, innovatív matematikai megoldások, modellek, melyek a számítógép tanulását javítják és segítik, illetve annak teljesítményét hivatottak egy magasabb szintre emelni, továbbá biztonságosabb és pontosabb működést elérni. Erre jó példa az OpenAI nyelvi és szöveggenerátor modellje (Radford *et al.*, 2019) mely hatalmas potenciált rejt magában, hiszen nem csak szöveget generál, de értelmes, összefüggő tartalmat is ad hozzá.

Az autóiparban is hasonlóan nagy érdeklődés mutatkozik a különféle autonóm rendszerek létrehozására mesterséges intelligencia segítségével. (Russell, Norvig and Davis, 2010)(Amini *et al.*, no date) munkájukban létrehoztak egy általános GPS segítségével is működő end-to-end navigációs neurális hálót, mely képes irányítani az autót, miközben magát lokalizálja az adott térképen. A mindenki számára ismert Tesla vagy Waymo is mesterséges intelligenciával próbálja megoldani önvezető funkcióinak legjavát. Ehhez rengeteg adatuk van, mely hatalmas előnyt jelent ezen a területen, hiszen a feladatok nagyrésze hatalmas mennyiségű felcimkézett adat meglétét követeli, melynek előállítása nem csak költséges, de ugyanakkor időigényes is. A tanuló algoritmusok közül a felügyelt (supervised) és részben felügyelt (semi-supervised) tanulás területén már jelentős sikereket értek el az autóipari környezetben is, mint például a ChauffeurNet (Bansal, Krizhevsky and Ogale, 2018) vagy a járműfelismerésben Lidar és kamera alapon (Asvadi *et al.*, 2018). Ugyanakkor az adatok hiánya, illetve az adott feladatok tulajdonságai miatt vannak esetek, amikor a megerősítéses tanulási módszert választják a fejlesztők. (Silver *et al.*, 2017) a Google DeepMind csapatával közösen ért el kimagasló teljesítményt a témában egy olyan algoritmussal, ahol a gép saját maga ellen játszott és így tökéletesítette tudását a bonyolultságáról híres GO játékban. Mások (Xu *et al.*, 2017) úgy használták fel a megerősítéses tanulás eszközeit, hogy működő energia- és erőforráskihasználást eredményező algoritmust hoztak létre, mely a felhő alapú rádió hálózatokat (Cloud Radio Access Networks (RANs)) szabályozza. A Massachusettsi egyetem által kiírt kihívás (Fridman, Terwilliger and Jenik, no date) is a megerősítéses tanulás

Online: ISBN 978-963-88875-4-2
Paper 16

Editor: Dr. Péter Tamás

területén születő újabb eredmények ösztönzését célozza. A kihívás lényegében egy erős forgalommal rendelkező autópályás környezetet szimulál, amelyben a feladat egy olyan algoritmus készítése, mely egy, vagy akár több jármű mozdulatait irányítja úgy, hogy az általunk megfigyelt (és irányított) jármű, (továbbiakban EGO) a lehető leghamarabb végig érjen az előre meghatározott hosszúságú pályán. Ezen a munkán felbuzdulva, illetve a szerzők kutatási területére összpontosítva került jelen cikk is elkészítésre.

A munkánk célja egy olyan ágens kidolgozása, tanítása és értékelése, mely képes egy adott hosszúságú autópálya szakaszon, változó forgalmi körülményekkel tarkított szituációkban biztonságosan, ugyanakkor gyorsan és pontosan irányítani az EGO járművet, úgy hogy annak viselkedése a többi autót ne veszélyeztesse, illetve balesetmentesen kivitelezze feladatát. Mindezt természetesen a megerősítéses tanulás használatával, hiszen az imént definiált feladat jellege és tulajdonságai miatt olyan adatok meglétét és gyűjtését követelné, mely nem csak hogy egy egyetemi kutatócsoport erőforrásait, de a legnagyobb cégekét, mint például a Tesla vagy Waymo is megterhelnék. A következő fejezetben a megerősítéses tanulás alapjait részletezzük, majd a 3. fejezetben a kifejlesztett tanító környezetet. Ezt követően a 4. fejezetben részletesen ismertetjük a tanuló ágensünket, majd az 5. fejezetben bemutatjuk és értékeljük az elért eredményeket.

2. A MEGERŐSÍTÉSES TANULÁS ALAPJAI

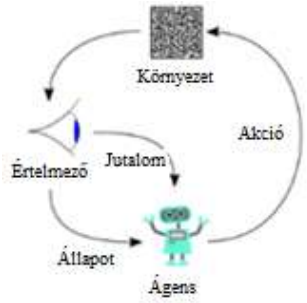
A megerősítéses tanulás a gépi tanulás olyan részhalmaza, ahol nincs szükség előre felcímkézett adatra, illetve olyan modell kialakításra, aminek szüksége van a „ground-truth” információra ahhoz, hogy tanulni tudjon. Két fajtáját különbözteti meg a szakirodalom, melyből az egyszerűbb az úgynevezett modell nélküli tanítás, ahol az ágenst körülvevő világ/környezet nincs modellezve, arról semmilyen információt nem tud a tanulás során, így saját magának kell a világ működését kitapasztalnia és megtanulnia, mikor mi a megfelelő lépés. A másik típus már komplexebb, ebben az esetben az ágens birtokában áll egy környezet reprezentáció, mellyel számolva hozhatja meg döntését a következő cselekvéséről.

A tanulás folyamata tulajdonképpen cselekvések sorozatából áll, mely a kezdeti állapotból kiindulva addig tart, még az előre definiált megszakító esemény be nem következik.

Az 1. ábrán látható folyamatábra jól szemlélteti a lépéseket. Az akciók új állapotokat hoznak létre, melyet a környezet befolyásol. Az egész problémakör visszavezethető a valószínűségszámítás olyan példájára, ahol azt próbáljuk megbecsülni, hogy mennyi a valószínűsége s állapotból s' állapotba lépni a akcióval. Ezt a (1) egyenlet szemlélteti

$$P_{a(s,s')} = \Pr\left(s_{\{t+1\}} = s' \middle| s_t = s, a_t = a\right) \qquad (1)$$

Célunk a környezet megismerése amellett, hogy jó döntést hozzunk a következő akciót illetően, és ezzel az a akcióért kapott jutalmunkat maximalizáljuk (2):



1. ábra – A megerősítéses tanulás folyamatábrája

$$R_a(s, s') \qquad (2)$$

Fontos megjegyezni, hogy a témában örök dilemmát okoz az „exploitation vs. exploration” problémája. Tanulás közben el kell dönteni, hogy mi az a szint, amikor már nem kísérletezünk új akciókkal, és nem tárjuk fel a még lehetségesen jó megoldásokat, hanem a már ismert és jól bevált lépéseket választjuk. Több szakirodalom is foglalkozott ezzel a kérdéskörrel, de összességében mindig az adott probléma határozza meg az optimális határt. (Coggan, no date; Ishii, Yoshida and Yoshimoto, 2002)

A megerősítéses tanulás esetén az ágens egy epizód után valamilyen jutalmat kap, annak függvényében, hogy mennyire teljesítette jól a feladatát. Ennek a jutalom függvénynek a feladathoz illően történő meghatározása a probléma megoldásának szükséges feltétele. Nehézséget okoz, hogy az adott ágens nem kap visszajelzést a folyamat közben, csupán a folyamat legvégén tudja meg, hogy amit csinált milyen jutalmat érdemel. Így a kapott jutalom visszaszámolása a múltban elkövetett akciókra egy újabb fejtörést okoz számunkra, hiszen meg kell találni az egyensúlyt az adott epizód megszakításához vezető, ahhoz még hozzájáruló akciók sorozatának és az azt már nem befolyásoló akciók között. A visszaterjesztés egyenlete a (3) tartalmazza a γ értékét, ami a visszaterjesztés nagyságát, mélységét szabályozza.

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \qquad (3)$$

A már említett „exploration-exploitaion” illetve a visszaterjesztés problémáját az idők során kidolgozott algoritmusok máshogy kezelik. Az algoritmusok osztályozásához számos szakirodalmat találni (Sutton *et al.*, no date; Szepesvári, 2010; Mnih *et al.*, 2016), amelyből a teljesség igénye nélkül az alábbiakat emelnénk ki. A lehetséges megoldások egyike a „brute force”, ahol az összes lehetőséget végig próbálva ismerjük meg mind az akcióteret, mind az állapotteret. Nem igényel különösebb magyarázatot, hogy ez csak kis terek esetén érvényesíthető, hiszen igen

erőforrás- és időigényes megoldás. Pozitívum viszont, hogy biztosan megoldásra vezet. A „value function” megoldási módszerek pedig arra alapoznak, hogy megtalálja a legnagyobb értékű akciót minden lépéshez, melyhez a legjobb várható jutalom tartozik. Ezeket elérhetjük csupán a „policy” maximalizálásával, az akciók maximalizálásával és a kettő vegyítésével is.

$$\pi: A \times S \rightarrow [0,1] \quad (4)$$

$$\pi(a,s) = \Pr(a_t = a \mid s_t = s) \quad (5)$$

A (4)-es egyenletben a π jelenti az aktuális „policy”-t, melynek értéke [0,1] között mozoghat, és a már említett valószínűséget tartalmazza, mely az adott a akció választása s állapot esetén (5).

A (6) ismerteti a „value” megközelítést, ami a várható legnagyobb jutalmat veszi figyelembe egy adott s állapothoz.

$$V_\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right] \quad (6)$$

A „value” és „policy” függvényeket komplex problémák esetén valamilyen függvény approximátorral tudjuk közelíteni.

Habár napjainkban egyre jobban elterjednek a mélyhálós megerősítéses tanulást használó algoritmusok is, melyeknek kezdeti stabilitási nehézségeire már számos megoldás született, mellyel sikerült a tanulási folyamatot és az optimális jutalom értékhez való konvergenciát biztosítani, jelen munkában mi nem mélyülünk el a mélyhálós tanulásban, csupán egyszerűbb algoritmusokkal próbáljuk megfogni a kialakított problémakört.

3. KÖRNYEZETI MODELL SUMO SZIMULÁTORRAL

3.1 A szimulációs szoftver

Bár az általunk kidolgozott megoldás a már említett típusok közül a modell mentes esetet használja, az ágens akcióinak válaszául az új állapot meghatározásához szimulációs környezetre volt szükségünk. Többek között felmerült egy autópályás forgalmi szimuláció implementálása Python nyelven, ám az irodalomkutatás eredményeként egy sokkal jobb, már létező megvalósításra találtunk, a SUMO, teljes nevén Simulation of Urban MObility egy ingyenes nyílt-forráskódú program révén, melyet elsődlegesen forgalmi szimulációk készítésére hoztak létre. Egyszerűen hordozható, mikroszkopikus és folytonos közúti forgalomszimulátor, mely képes hatalmas úthálózatok kezelésére is. [SUMO].

Egyik nagy előnye, hogy ingyenesen elérhető, mely megkönnyíti alkalmazását különböző egyetemi projektekben is. További előnye, hogy mivel nyílt-forráskódú szoftver, így bárki által fejleszthető, illetve nehezebb problémák megoldására felhasználói fórumok nyújtanak felületet. A SUMO többnyire shellből, vagy parancssorból vezérelhető, ám Windows és Linux felhasználásra grafikus felületet is társítottak hozzá a fejlesztők. A SUMO GUI egy kétdimenziós megjelenítőt biztosít (2. ábra) a háttérben futó szimulációkhoz, mely jelenlegi célunknak megfelelő. Moduláris felépítésének köszönhetően gyorsan módosíthatók az egyes komponensek, kiegészíthetők saját igényeink szerint. Számos kiegészítő létezik a szoftverhez, de ezekből a számunkra fontosokat említjükcsak meg. A szimulációtervezés első lépése a NETEDIT programmal kezdődik, melyben lehetőségünk van hálózatot építeni, különböző úttípusokat megadni és komplex úthálózatokat tervezni a későbbi teszteléshez.

A NETCONVERT alprogrammal valós létező hálózatokat importálhatunk OpenStreetMap-ból, vagy már létező más programmal készült hálózatokból, mint például VISUM, NavTeq vagy a Vires VTD. Az úthálózat egy sajátos $.net.xml$-ben kerül mentésre.

Esetünkben egy két kilométer hosszú, három sávval rendelkező egyenes útszakasz került kialakításra, mely elegendő a jelen célkitűzés eléréséhez. Szerencsére a parancssorból futtatható szimuláció megváltoztatására is lehetőséget ad a SUMO, hiszem valós-időben kérhetünk le értékeket és módosíthatunk futás közben a TraCi (Traffic Control Interface) interfész segítségével. Így a Python kód és a SUMO szimuláció összekapcsolható. Ezek után a különböző függvények segítségével tudjuk manuálisan lekérni és állítani

2. ábra - A SUMO egy pillanatképe

az adatokat. Lehetőséget nyújt számunkra az autonóm autónk forgalomba helyezésére és az ágensünk által definiált akciókkal történő beavatkozásra. Erről később részletesen értekezünk.

Fontos megjegyezni, hogy a SUMO biztosít véletlenszerű szimulációs paraméterek megadására megoldást, így két szimuláció nem lesz ugyanolyan, mely a későbbi tanulási folyamatban fontos szerepet játszik.

3.2 A szimuláció részletei

A SUMO tehát egy megfelelő program számunkra, mellyel egyszerűen implementálhatunk olyan szcenáriókat, melyek a valóságot tükrözik, ugyanakkor kivitelezésük nehézkes volna. Megépítésre került tehát a már említett 2 [km] hosszú autópálya részlet, három sávval, 130 [km/h] maximális megengedett sebességgel. Kialakításra került az autópályán véletlenszerűen elhelyezett, megadott normális eloszlású

Online: ISBN 978-963-88875-4-2
Paper 16

Editor: Dr. Péter Tamás

véletlen sebességgel és eltérő tapasztalati szinttel rendelkező többi autós, és egy megfelelő pillanatban az EGO jármű szintén véletlenszerű sávválasztással útnak indul. A szimuláció paramétereit a következő táblázat tartalmazza:

	HOSSZ [M]	σ	SEBESSÉG-FAKTOR	VALÓ-SZÍNŰSÉG
EGO	5.2	0	-	0
CAR1	5.2	0.5	1, 0.3, 0.5, 1.1	0.4
CAR2	4.5	0.8	1, 0.45, 0.55, 1.2	0.3
CAR3	3.7	1	1, 0.2, 0.6, 1.1	0.3

1. táblázat – Az autók paraméterei

A σ a tapasztalati szintet jelöli ebben a táblázatban, mely 0 és 1 közötti érték. Ha 1, akkor tökéletesen szabálykövető, ideális sofőrről beszélünk. Ez a jobbra tartást, követési távolságot és minden egyéb manővert is ideálisan elvégző járművezetőt tükrözi. A sebességfaktor oszlopban az első szám az eloszlás átlaga, mely a maximálisan megengedett sebességhez százalékos arányban van megadva. Így alakul a második szám is, mely százalékban kifejezi a minták tartózkodásának 95 százalékos arányát. Az utolsó két számjegy pedig a random választott sebesség maximum és minimum értékét mutatja, mely szintén a megengedett maximális sebességhez képest van kifejezve. A valószínűség pedig azt mutatja, hogy mekkora valószínűséggel kerül elhelyezésre az új jármű, az adott típusú autóból.

A szimuláció során több forgalmi sűrűség kipróbálásra került, melynek részletezése a következőkben kerül ismertetésre.

4. AZ ALKMAZOTT TANULÓ ÁGENS

Ahogy már említettük, egy modellmentes ágenst használtunk a probléma megoldására. Rengeteg lehetőség és algoritmus áll rendelkezésre a témában, de az egyszerűség kedvéért és a jól tanulhatóság miatt, a „Vanilla Policy Gradient” megoldást választottuk ágensként. Ebben a felállásban egy osztály került kialakításra Python nyelven, mely implementálja a megnevezett algoritmust. A Pythonban implementált kód fontos részét képezi a Pytorch keretrendszer, mely függvényei és funkciói a kódolás szerves részét képezik. A Pytorch egy elsődlegesen Facebook R&D csoport által fejlesztett keretrendszer, melyben folyamatosan kerülnek implementálásra az újonnan megjelenő algoritmusok. Mindezt memória és számítási igény kompatibilisen és optimalizált változatban publikálják. Temérdek matematikai függvényt valósítanak meg benne, a gradiens visszavezetéstől a különböző optimalizálási függvényeken és módszereken keresztül egészen a legmodernebb háló és neuron típusokig. Az összes függvény kézenfekvő használata és gyors implementálhatósága miatt vált széles körben elterjedté.

4.1 Vanilla Policy Gradient (PG) módszer:

Az alap algoritmusok közé tartozó Vanilla PG megvalósítása a policy változtatáson alapszik. A módszer során a neurális háló bemenetére érkező állapotvektor alapján minden lépésben egy valószínűség vektort ad eredményül, melyből kategorikus függvény segítségével egy valószínűségi eloszlást csinálunk, majd ebből súlyozott véletlenszerű kiválasztással egy akciót generálunk. Ezt elmentjük, hogy ha az epizódnak vége, tudjuk milyen akciók vezettek az adott jutalomhoz. A háló bemenetére a kialakított környezetből a következő adatok érkeznek.

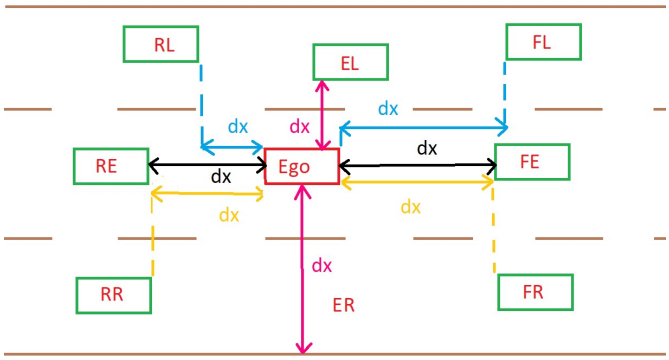
4.2 Bemenet

Az EGO járművünk sebessége, elfordulási szöge, elfoglalt sáv indexe, y irányú pozíciója, illetve a környezetében lévő mátrix szerűen nyolc rácsponton tartózkodó járművek relatív távolsága és sebessége. Ez összesen 20 változó, a következő sorrendben:

$$\begin{bmatrix} front_{left}(FL), & front(FE), & front_{right}(FR), \\ rear_{left}(RL), rear(RE), rear_{right}(RR), \\ left_{ego}(EL), right_{ego}(ER), \\ speed, angle, y_{pos}, lane \end{bmatrix}.$$

Ez a 3. ábrán is látható. A dx relatívan mutatja az objektumok EGO-hoz képesti elhelyezkedését, míg az oldalsó járműveknél $(left_{ego}, right_{ego})$ a dx a legközelebbi y irányú objektum távolságát tartalmazza, ezzel segítve, hogy az EGO tudja az autópálya szélétől való távolságot vagy a mellette tartózkodó autó távolságát.

A különböző állapotváltozók távolság esetén maximálisan 200 [m] nagyságig, míg sebesség esetében 50 [m/s] értékig mehetnek pozitív és negatív irányban is. Az elfordulási szög – π és π között mozoghat, de értéke csak $\pm\frac{\pi}{2}$ között változik a szituációkból fakadóan. Az elfoglalt sáv indexe pedig 0-2 közötti egész szám, mely lentről felfelé, azaz jobbról balra történő számozással értendő.



3. ábra – Az állapotváltozók alakulása

4.2 Háló

Ezt a bemenetet megkapva a háló egy Multi Layer Perceptron (MLP) struktúrán, mely egy igen egyszerű, több rétegből felépülő háló, kerül átvezetésre, melynek rétegeit a 2. táblázat tartalmazza. Az első réteg bemenete az állapotváltozóktól függ, míg a 3. réteg kimenete az akciótér nagyságától. Az aktivációs függvénynek a ReLu-t választottuk a pozitív tulajdonságai és tanulást segítő felépítése miatt. Az utolsó réteget a Softmax függvény biztosítja, hogy a háló kimenete az egyes akciók valószínűségét tartalmazza. Ezt a kimenetet, ahogy már korábban említettük, egy kategorikus eloszlási függvénybe helyezzük, melyből mintavételezve születik meg az előző állapot alapján a következő lépés akciója.

	TÍPUS	BEMENET	KIMENET
RÉTEG 1	*Linear*	20x1	128x1
AKTIVÁCIÓ 1	*ReLu*	128x1	128x1
RÉTEG 2	*Linear*	128x1	64x1
AKTIVÁCIÓ 2	*ReLu*	64x1	64x1
RÉTEG 3	*Linear*	64x1	9x1
RÉTEG 4	*Softmax*	9x1	9x1

2. táblázat – A háló paraméterei

4.3 Akciótér

Az ágens diszkrét akcióteret használ, amely három akciót tartalmaz a kormányzással kapcsolatban (jobbra 0.3°, balra 0.3° és egyenesen), illetve három akciót, mely az EGO sebességét csökkenti, növeli, vagy nem változtatja. Ezen akciók összesen 9 kombinációt eredményeznek, így 0 és 8 közötti egész számokkal reprezentálhatók. A későbbiekben részletesen kifejtésre kerül, hogy miért ilyen és ennyi akciót használtunk.

4.4 Visszaterjesztés

A hálónál a visszacsatolás a Pytorch-os *loss.backward* függvénnyel kerül számításra, ahol a $loss$ számítását a következő egyenlet mutatja:

$$loss = policy_{history} * log_{prob}(action)$$

Itt a $polic_{history}$ az adott állapotra adott akció kategorikus függvényeit tartalmazza az epizód minden megtett lépéséhez.

A háló tanításakor többféle kombinációt is figyelembe vettünk a rétegek nagyságát és a „learning rate”, valamint a γ értékét illetően, és több iteráció alapján a 2. táblázatban részletezett háló paramétereket találtuk megfelelőnek. A „learning rate” változtatása nagyban befolyásolta a modell taníthatóságát, ahogyan ez várható is volt. A túl nagy (jelen esetben 0.001) egy szuboptimális megoldáshoz rendelte a hálót, így az nem tanult meg rendesen közlekedni, ütközéssel vagy autópálya elhagyással fejeződtek be a jelenetek. A túl kicsi (0.000001) érték esetén pedig nagyon lassú a folyamat és az epizódszám is jelentősen megnőtt.

4.5 Optimalizáló

A tanítás egyik legfontosabb pontja az optimalizáló függvény/metódus kiválasztása. Az irodalomkutatás azt indokolja, hogy az ADAM optimalizálót használjuk. Többek között, az általunk kihasznált tulajdonsága az, hogy a látens tér nem minden irányába ugyanakkora lépéssel történik a visszapropagálás, hanem a változás nagyságával arányosan. Így megengedi, hogy az előzés és a gyorshajtás más súlyozással kerüljön optimalizálásra.

4.6 A jutalom

Egy epizód a kezdeti állapotból egészen addig tart még valami megszakító esemény be nem következik. A kedvező eset az, ha az előre definiált autópálya szakaszon az EGO végig ér, de a tanítás folyamán előfordulhat még pályaelhagyás, ütközés, esetleg túl lassú közlekedés is. Ezek az események mind az epizód végét jelentik. Ilyen esetekben a háló negatív jutalmat kap. Ha azonban az EGO teljesíti a 2 km-t hiba nélkül, nem kap mínusz pontot, hanem 0 értékű jutalmát kell elkönyvelnie. Ez a globális jutalmazás az epizódok végeredményének kifutását hivatott mutatni az ágensnek. Míg minden lépés után a sebességgel és előző lépéshez képest megtett távolsággal arányosított jutalmat, azonnali jutalomnak nevezzük, mely az adott akció instant jóságát indikálja, így ezzel ösztönözzük az ágenst a minél nagyobb és gyorsabb haladásra. A már bemutatott γ pedig nagymértékben meghatározza, hogy a lokálisan kapott jutalom, illetve a globális jutalom az epizód végén milyen erős kapcsolatban van egymással, melyik hangsúlyosabb.

5. A TANULÁS EREDMÉNYEI

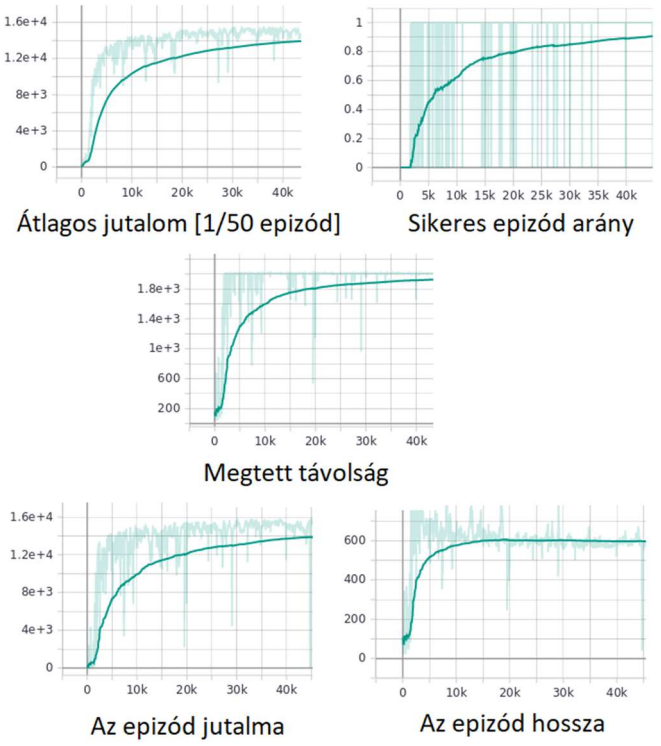
5.1 Első szituáció és eredményei

A modell és szimulációs környezet implementálása után első körben egy olyan beállítással tanítottuk az ágenst, melyben csak az EGO jármű volt jelen. Nem kapcsoltunk be a szimulációba egyéb járművet, csupán a kezdeti sebessége és a kiindulási sávja változott az epizódok során. A tanítás során elvárt viselkedés az volt, hogy az EGO megtanulja az autópályán maradást, illetve az adott szakaszon való minél gyorsabb végig haladást. Az ingerszegény környezet beállítással és az alkalmazott már említett jutalmazással a modell viszonylag gyorsan megtanulta az autópályán maradást, és jól optimalizálta a sebességét, mely a lehető legrövidebb epizódot eredményezte. A tanulás során megfigyelt indikátorok, melyeket a TensorBoard keretrendszerrel monitoroztunk, az epizódok hossza, jutalma. Ezek mellett figyeltük a sikeres epizódok számát, illetve 50 epizódonként az átlag jutalmat, és a megtett távolságot. Az

Online: ISBN 978-963-88875-4-2
Paper 16

Editor: Dr. Péter Tamás

értékek alakulása a 4. ábrán láthatók. A „learning rate” a már említett 0.0001 volt, és nem változott a tanulás során. A γ értékét 0.99 re állítottuk, ezzel biztosítva a sikeres teljesítés esetén kapott jutalom kellően messze (múltba) való visszanyúlását.



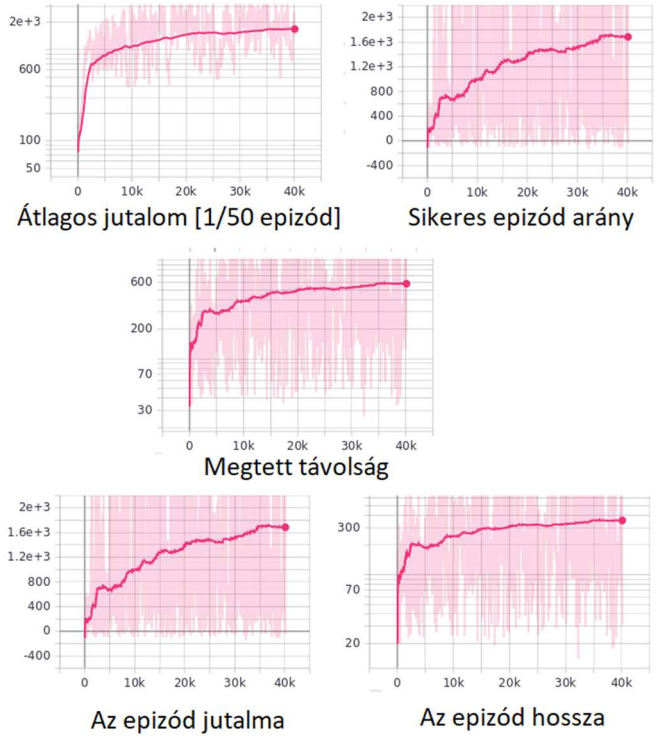
4. ábra – Az 1. beállítás tanulási folyamata

5.2 Második szituáció és eredményei

Az első beállítás sikeres tanulását követően az ágensünknek egy előre véletlenszerűen elszórt járművekkel teli környezetben kellett teljesítenie a feladatot, megtanulnia közlekedni. Itt a helyzete már bonyolódott, ugyanis meg kellett tanulnia mikor fékezzen, kezdjen gyorsulni, váltson sávot, vagy húzódjon le. Elvárt viselkedésben kezdeti ütközések, pályaelhagyás és lassú haladás is szerepelt, melynek időbeli csökkenését vártuk. Ahogy az 5. ábrán is látszik, a tanulási folyamat nagyságrendekkel több epizódot igényelt, és a konvergencia is lassabb volt, hiszen a komplexebb feladatot optimalizálni nehezebb. Az első feladatnál használt „learning rate” ugyanaz maradt, illetve a γ értékén sem változtattunk.

Megjegyzendő továbbá, hogy bár a tanulásban egyre tovább halad a pályán és egyre nagyobb jutalommal teljesíti azt, az epizódok sikerességének szórása még a tanulás végén is nagy. Egyre kevesebbszer ütközik, de sajnos 50-ből egyszer hibázik. Ez lehet a SUMO környezetben kialakított járművek nem determinisztikus viselkedése miatt, vagy annak okán, hogy az ágens nem tudta teljes mértékben optimalizálni a megadott állapottérhez kapcsolódó akcióteret. Felmerül a jutalom egyszerűsége is, mely komplexebb megvalósítása további javulást, ugyanakkor hosszabb tanulási folyamatot eredményezne.

Kiértékelésre nem csak a már említett TensorBoard vizualizációt használtuk, hanem a betanított ágens viselkedését a SUMO GUI segítségével is analizálhattuk. Ebből jól látszik, ahogy az ágens előzésekbe kezd, próbál minél jobban és gyorsabban haladni, ám az esetleges ütközések a túl lassú járművek, vagy a tömött autópályán kényszerhelyzetből fakadnak. A fékezés értékei néhol nem elegek az ütközés elkerülésére, vagy nem kezdi meg időben azt. További észrevétel, hogy numerikus számolásokból fakadó esetleges sávváltások is közre játszhatnak az ütközésben, hiszen a kormányszög nem egész számmal történő csökkentése vagy növelésekor a Python néhány tízezredes eltéréseket produkál, melyek nagy sebességnél már sávváltást okozhat. Ezt természetesen programozási eljárásokkal ki lehet küszöbölni. Egyéb hibákkal a következő rész foglalkozik.



5. ábra – A 2. beállítás tanulási folyamata

5.4 Elkövetett hibák

A tanítás első próbálkozásai alkalmával elkövetett hibák részletezésére is fordítunk egy rövidebb bekezdést, ugyanis nagyon tanulságos lehet.

Az állapottér és az állapotváltozók számos változata is tesztelésre került. Első iterációknál az érzékelt többi autó 500

méteres környezetben volt. Ez azonban feleslegesen soknak bizonyult, valamint a gyakorlatban alkalmazott környezetérzékelő szenzorok, úgy mint LIDAR vagy kamera sem képes ekkora távolság érzékelésre. Így redukálva lettek a „szenzor" érzékelési távolságai a már említett 200 m-es nagyságra. Itt is megemlíthető, hogy még ez a távolság is nagy lehet, de jelentősen segített az állapottér egyszerűbb megismerésében. A sebességeket is nagyobb határok között kezdtük reprezentálni, ám figyelembe véve, hogy relatív sebességről beszélünk a +- 50 m/s több mint elegendő. Az elfordulási szögről már esett szó, így a +- PI/2 között mozog, mely reálisan nézve felesleges, de extrém esetekben előfordulhat.

A sávinformációt az ágens 0-2 számokkal kapja, mely reprezentáció nem a legideálisabb, azonban jól működött így az optimálisabb „one-hot-vector" reprezentációt nem alkalmaztuk. Az imént említett értékek kezdetben normalizálás nélkül kerültek az állapotváltozó vektorba, viszont később optimalizálva az értékeket a normalizálás mellett döntöttünk. Ez is segítette a tanulást, hiszen így 0-1 közé értékeket kapott a háló, mellyel a gradiens „vanishing" és „exploding" jellegét sikerült kiküszöbölni.

Az akciótér nagysága is egy fontos faktora a tanulási folyamat optimalizálásának. Első megközelítésünkben az akciótér nem csak három-három akciót tartalmazott, mint ahogy azt korábban bemutattuk, hanem 7x7 akcióból, azaz 49 kombinációból választhatott az ágens. Ez nagymértékben hátráltatta a tanulást, hiszen a megismerendő akciótér és ezzel együtt az arra válaszul érkező állapotváltozók kiismerése is nehezebb volt.

6. KONKLÚZÓ

Összességében egy egyszerűnek mondható neurális hálóval sikerült egy olyan ágenst betanítani, ami képes az adott autópályán forgalomban, vagy anélkül, végigmenni a lehető leggyorsabb módon. Ennek ellenére nem képes optimálisan, komfortosan vezetni, szabályokat betartani, illetve olyan előzéseket produkálni (megtanulni), melyet egy autonóm járműtől elvárnánk. Egy komplexebb háló implementálásával, finomabb jutalmazási rendszer kiépítésével, illetve interaktívabb és szofisztikáltabb környezet kialakításával ez is megvalósíthatónak látszik, mely a jelenlegi munka folytatása lesz. A megerősítéses tanulásnál megtapasztalt konvergencia kritériumok és szükséges illetve elégséges feltételek kitapasztalása történt a projekt során, melyből konklúzióként leszűrhető, hogy a tanulás lényegesen gyorsítható az állapotváltozók normalizálásával, azok megfelelő meghatározásával, sokat számít a tanulásban az akciótér nagysága, mely minél kisebb, annál jobb. A projekt folytatásában egy mélytanuló hálóval és folytonos akciótérrel próbáljuk majd megvalósítani egy erős forgalomban minden téren optimálisan navigáló ágens tanítását.

7. KÖSZÖNETNYILVÁNÍTÁS

EFOP-3.6.3-VEKOP-16-2017-00001: Tehetséggondozás és kutatói utánpótlás fejlesztése autonóm járműirányítási technológiák területén - A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

HIVATKOZÁSOK

Abdelaziz, A. *et al.* (2018) 'A machine learning model for improving healthcare services on cloud computing environment', *Measurement*. Elsevier, 119, pp. 117–128. doi: 10.1016/J.MEASUREMENT.2018.01.022.

Amini, A. *et al.* (no date) *Variational End-to-End Navigation and Localization*. Available at: https://arxiv.org/pdf/1811.10119.pdf (Accessed: 6 August 2019).

Asvadi, A. *et al.* (2018) 'Multimodal vehicle detection: fusing 3D-LIDAR and color camera data', *Pattern Recognition Letters*. North-Holland, 115, pp. 20–29. doi: 10.1016/J.PATREC.2017.09.038.

Bansal, M., Krizhevsky, A. and Ogale, A. (2018) 'ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst'. Available at: http://arxiv.org/abs/1812.03079 (Accessed: 6 August 2019).

Carbonneau, R., Laframboise, K. and Vahidov, R. (2008) 'Application of machine learning techniques for supply chain demand forecasting', *European Journal of Operational Research*. North-Holland, 184(3), pp. 1140–1154. doi: 10.1016/J.EJOR.2006.12.004.

Coggan, M. (no date) *Exploration and Exploitation in Reinforcement Learning*. Available at: http://neuro.bstu.by/my/Tmp/Papers-RL/FinalReport.pdf (Accessed: 6 August 2019).

Fridman, L., Terwilliger, J. and Jenik, B. (no date) *DeepTraffic: Crowdsourced Hyperparameter Tuning of Deep Reinforcement Learning Systems for Multi-Agent Dense Traffic Navigation*. Available at: https://selfdrivingcars.mit.edu/deeptraffic. (Accessed: 6 August 2019).

Ishii, S., Yoshida, W. and Yoshimoto, J. (2002) 'Control of exploitation–exploration meta-parameter in reinforcement learning', *Neural Networks*. Pergamon, 15(4–6), pp. 665–687. doi: 10.1016/S0893-6080(02)00056-4.

Mnih, V. *et al.* (2016) *Asynchronous Methods for Deep Reinforcement Learning*. Available at: http://proceedings.mlr.press/v48/mniha16.pdf (Accessed: 6 August 2019).

Radford, A. *et al.* (2019) *Language Models are Unsupervised Multitask Learners*. Available at:

https://github.com/codelucas/newspaper (Accessed: 13 April 2019).

Russell, S. J. (Stuart J., Norvig, P. and Davis, E. (2010) *Artificial intelligence : a modern approach*. 3rd edn. Prentice Hall.

Silver, D. *et al.* (2017) 'Mastering the game of Go without human knowledge', *Nature*. Nature Publishing Group, 550(7676), pp. 354–359. doi: 10.1038/nature24270.

Sutton, R. S. *et al.* (no date) *Policy Gradient Methods for Reinforcement Learning with Function Approximation*. Available at: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf (Accessed: 6 August 2019).

Szepesvári, C. (2010) 'Algorithms for Reinforcement Learning', *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool Publishers , 4(1), pp. 1–103. doi: 10.2200/S00268ED1V01Y201005AIM009.

Xu, Z. *et al.* (2017) 'A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs', in *2017 IEEE International Conference on Communications (ICC)*. IEEE, pp. 1–6. doi: 10.1109/ICC.2017.7997286.