Szegedi Tudományegyetem

Informatikai Intézet

TDK-dolgozat

Készítette:
Gábrisch Krisztián

Témavezető:
Megyeri István

Szegedi Tudományegyetem

Természettudományi és Informatikai Kar

Deep Reinforcement Learning Stabilitásfejlesztés Egyirányú Mozgást Végző Robotvezérlőkben

Készítette:
Gábrisch Krisztián
5. évfolyam
Programtervező informatikus BSc

Témavezető:
Megyeri István
Tanár segéd

Tartalomjegyzék

Bevezetés

A megerősítéses tanulás a gépi tanulás egyik izgalmas és dinamikusan fejlődő ágazata, amely azon alapszik, hogy az algoritmusok döntéseket hoznak és tanulnak azok következményeiből. Az ágens a környezettel interkacióba lép, és a célja, hogy maximalizálja a jutalmat azáltal, hogy megtanulja optimalizálni a cselekvéseit és interakcióit. A megerősítéses tanulás módszerei a hagyományos felügyelt és felügyelet nélküli tanulástól eltérően egy jól definiált ágens folyamatosan felfedez és alkalmaz különböző stratégiákat, reagálva a környezet változásaira. Ez a tanulási forma különösen hasznos olyan komplex problémák esetén, ahol a helyes cselekvési mód nem közvetlenül nyilvánvaló vagy ahol a jutalmak késleltetettek. Alkalmazási területei rendkívűl széleskörűek, amely magába foglalja a robotikát, a különböző játékstratégiákat, egészségügyi döntéstámogató rendszereket. Egyik klasszikus példája a sakkból vagy a Go játékból ismert algoritmusok, amelyek képesek magas szinten versenyezni az emberi játékosokkal. A technológia fejlődésével egyre bonyolultabb és nagy erőforrást igénylő feladatok[6] is elvégezhetők a megerősítéses tanulást felhasználva, azonban a tanítási folyamat alatt a modellek általában egy előre meghatározott batch és iterációs korlátok közé van szorítva, ami korlátozó tényezőként működhet, és egy nem várt limitációt eredményezhet az ágensek teljesítményében. Ezek a korlátok hozzájárulhatnak a modellek instabilitásához, ezzel akadályozva a képességét a további jutalmak gyűjtésére. A dolgozat mérései a stabilitás optimalizálására irányultak, melyet a bukásig megtett lépések számában definiáltunk. Összességében a megerősítéses tanulás jelentős előrelépést jelent a gépi tanulás területén, kihívásokat és lehetőségeket egyaránt kínálva a kutatók és fejlesztők számára a tudományos felfedezések és technológiai innovációk előmozdításában.

1. fejezet

Megerősítéses tanulás

Ebben a szekcióban megismerkedhetünk a gépi tanulással és a különböző fajtáival, illetve részletesebb belátást nyerhetünk az önmegerősítéses tanulásba, ahol a modell egy előre definiált jutalomrendszer alapján saját magát tanítva juthat el a célállapotig, vagy jelenleg elért nagy eredményeibe, ahol már híres sakk és GO mestereket is legyőztek, mindezen felül több híres játékban is kiemelkedő teljesítményt tudtak produkálni, mint például az Atari játékokban [3] vagy az összetett Starcraft-ban, de FPS játékokban[4] is erősen megállta a helyét és sok esetben a profiknál is jobban teljesített. Többek között kitérünk még a Trust Region Policy Optimization algoritmusra is, amelyet felhasználva tanult a modellünk, több különböző tanítási környezetet is abszolválva.

1.1. Gépi tanulás

A gépi tanulás a mesterséges intelligencia egyik szegmense, amely olyan számítógépes rendszerekre fókuszál, amik "önállóan" képesek tanulni. A rendszer képes új információkat és ismereteket generálni adatokból, mintákból, ennek eredményeképpen a rendszer feltudja ismerni és megtudja határozni a szabályszerűségeket, amik alapján önállóan helyes döntéseket tud hozni ismeretlen adatok esetén is. Az elérni kívánt cél érdekében több algoritmus is rendelkezésre áll, amelyek a feladat jellegétől függően választhatók ki.

A gépi tanulásnak három főbb megközelítése van:

1. Felügyelt tanulás: A tanító adatbázis halmaza adott. A célja, hogy eddig nem látott példákon is a modell helyesen működjön.

2. Felügyelet nélküli tanulás: A megfigyelésekhez nem áll rendelkezésre elvárt célérték. Célja, hogy az adatok között összefüggéseket azonosítson, ami alapján nem látott adatokat is tud értelmezni.

3. Megerősítéses tanulás: A modell akciókat hajt végre, ezekre az akciókra kap megerősítést (jutalmat) vagy akár büntetést, hogy mennyire csinálta jól a feladatát és a jutalom maximalizálására törekedve tanul folyamatosan. Több nehézsége is van, mint például a hatékony jutalombecslés, mivel a modellnek meg kell tudnia határoznia előre a cselekvés jutalmának értékét, ami nehéz feladat lehet. A tanulási idő is egy nehézsége lehet, ugyanis a modellnek sok iterációra van szüksége általában, hogy megtudja határozni megfelelően a cselekvéseinek jutalmai értékeit, ezalapján pedig a legoptimálisabb lépést meghozni. Ezeken felül költségesek lehetnek az interakciók, túl taníthatjuk a modelleket rossz jutalmazási rendszerből kifolyólag, és a környezeti változások is hozhatnak olyan meglepetéseket a működésben és a számításban, amire nem is gondoltunk volna.

1.2. A megerősítése tanulás két típusa: pozitív és negatív jutalom

Az ágensünk jutalmat kap a helyes lépéseiért, amelyeken keresztül folyamatosan tanul, és fejlődik. A jutalom mértéke eltérő lehet, attól függően, hogy az adott lépés mennyivel vitte közelebb a cél eléréséhez. A folyamatos jutalmi visszajelzések segítik az ágenst, hogy megtanulja, hogyan kell optimális döntéseket hozni a cél eléréséhez. Az ágens az összegyűjtött jutalmakat figyelembe véve újra szabályozza a döntési stratégiáját, hogy a kapott jutalom alapján a leghatékonyabb módon juthasson el a célhoz. Az ágens által gyűjtött jutalmak újraszámításakor beépülnek a tanulási folyamatba, hogy növeljék a rendszer képességét a jövőbeli problémák megoldásában.

A pozitív megerősítést úgy definiálják, mint amikor egy esemény egy adott viselkedés miatt következik be, és amennyiben helyes a válasz viselkedés, úgy ennek a gyakoriságát próbálja növelni, nagyobb jutalmat kiosztani, ezáltal megerősíteni és növelni a modellben ennek a mozgásnak az erősségét. A megerősítő tanulás egy megfelelő jutalmazási rendszerrel képes egyre tökéletesebb döntéseket hozni a tanulási folyamat során. Az idővel történő optimalizálás miatt a megerősítő tanulási algoritmusok időigényesebbek, mint más gépi tanulási technikák, de a hosszú távú teljesítményük sokkal magasabb. A modell precizitása folyamatosan javul a tanulási folyamat során, ami biztosítja a hosszú távú teljesítmény fenntartását, még akkor is, ha a környezet változik. Ha a modell túlságosan erős jutalmazásban részesül, az könnyen túltanuláshoz vezethet, ami csökkentheti a várt eredményeket. Ezen okból fontos, hogy egy megfelelően optimalizált jutalmazási rendszer legyen kialakítva, ami biztosítja a modell optimális teljesítményét.

A negatív megerősítés a megerősítő tanulás egy fontos eszköze, amelynek segítségével a modell azt a viselkedést kerülheti, amely nem hozza a kívánt eredményt. A negatív megerősítés célja, hogy ösztönözze a modellt a helyes cselekvések végrehajtására, a nemkívánt viselkedések elkerülésére, ezzel növelhetjük a modell teljesítményét és a pontosabb eredményeket. A negatív megerősítést alkalmazva megfelelően optimalizálhatjuk a jutalmazási rendszert, és elkerülhetjük a túltanulást, amely csökkenthetné a modell eredményességét.

1.3. Value-based módszerek a megerősítéses tanulásban

Olyan módszerek halmaza, amelyek a tanulás során egyes akciókat kiválasztva megpróbálják jósolni a maximum jutalmat az adott futásban és ezalapján kiválasztani a lehető legjobb lépéseket. Jó példa erre a Q-learning algoritmus[14]. A Q-learning lényege, hogy minden egyes művelethez az aktuális állapot alapján megpróbáljuk megjósolni az összes várható jutalmat és ez alapján azt a műveletet választjuk ki, amelyik a legnagyobb várható jutalmat hozza. A Q-learningben a Q jelenti ezt a várható legnagyobb jutalmat. Nem pontosan egy akcióhoz tartozó jutalmat számol, hanem azt, hogy egy akciót kiválasztva a végállapotig eljutva összesen mekkora jutalomra tehet szert. Amikor elkezdjük a tanítást, akkor a rendszer még csak felderíti a környezetet és a megtehető cselekvéseket, így az elején nem tudja meghatározni rendesen a Q értékét, azonban ahogy nő az ismerete, annál pontosabban fogja tudni megjósolni a Q értékét.

1.4. Policy-based módszerek a megerősítéses tanulásban

Közvetlenül a cselekvést meghatározó policy-t próbálják optimalizálni. Egy policy egy olyan szabályrendszer, amely meghatározza, hogy egy ágens milyen cselekvést hajtson végre adott állapotban. A célja ennek a módszernek, hogy megtalálja azt a policy-t, amely maximális jutalmat biztosít az ágens számára hosszabb távon.

A policy-based megközelítés direkt módon módosítja a policy paramétereit, például egy neurális hálózat súlyait, azzal a céllal, hogy a legjobb cselekvési stratégiát alakítsa ki. Ez a megközelítés különösen jól alkalmazható olyan környezetekben, ahol a cselekvések folytonosak, tehát nem korlátozódnak diszkrét választási lehetőségekre.

A policy gradient technika egy gyakran alkalmazott módszer a policy-based tanulás során, amely gradiens emelkedésen alapuló optimalizálással igyekszik finomhangolni a policy paramétereit, ezáltal a tanulási folyamat során az ágens fokozatosan jobban képes előre jelezni, hogy mely cselekvések vezetnek a legmagasabb összjutalomhoz.

Bár ez a megközelítés ígéretes eredményeket hozhat, gyakran magas szórású gradiensbecslésekkel jár, ami bizonytalanságot vihet a tanulási folyamatba és lassíthatja a konvergenciát. Ennek ellenére, ha megfelelően alkalmazzák, a policy-based

módszerek hatékonyan kezelhetik a komplex, folytonos cselekvési térrel rendelkező feladatokat.

1.5. Trust Region Policy Optimization algoritmus

[5] Kifejezetten arra törekszik, hogy nagy, de biztonságos lépésekben tudja frissíteni a policy-t, hogy az az optimális cselekvési stratégiát közelítse.

A módszer alapvető ötlete a "trust region" elve, ami egy matematikai keretet biztosít arra, hogy mekkora mértékben engedjük meg a policy megváltozását egy tanulási lépés során. Ez az elv segít kontrollálni a policy frissítéseit úgy, hogy azok ne térjenek el túlságosan a korábbi állapotától, így elkerülve a tanulási folyamat destabilizálását.

Használja a Kullback-Leibler (KL) divergencia fogalmát, ami egy statisztikai mérték arra, hogy két valószínűségi eloszlás mennyire különbözik egymástól. Az algoritmus úgy optimalizál, hogy minimalizálja a KL divergenciát a régi és az új policy között, miközben igyekszik maximalizálni a várható jutalmat. Ez a megközelítés biztosítja, hogy az ágens fokozatosan és stabilan javítsa teljesítményét anélkül, hogy túlzott kockázatot vállalna nagy policy frissítésekkel.

Így különösen hasznos bonyolultabb vagy folytonos cselekvésterű környezetekben, ahol fontos a nagy, de kontrollált léptékű tanulási frissítések biztosítása. Hatékonyan kezeli a policy gradiens algoritmusok által gyakran tapasztalt problémákat, mint amilyen a nagy szórású gradiensbecslések és a tanulási folyamat instabilitása.

1.6. Előnyei és hátrányai

Előnyei többek között

1. A megerősítéses tanulás olyan problémák megoldására is alkalmas, amelyeket nehéz megoldani a hagyományos tanulási algoritmusokkal, például ahol a jutalom csak később érhető el a cselekvések sorozatától függően. Ezen felül alkalmas a döntéshozás optimalizálására, miközben figyelembe veszi a környezeti tényezőket és változásait. Alkalmazása lehetővé teszi a számítógépes rendszerek számára, hogy képesek legyenek tanulni, hogyan cselekedjenek egy adott környezetben, anélkül, hogy korábbi tapasztalatokat vagy címkézett adatokat igényelnének.

2. Lehetővé teszi a számítógépes rendszerek számára, hogy idővel változó környezetekben is hatékonyan működjenek. Az állandóan változó környezetek megoldása számos nehézséget okoz a hagyományos tanulási algoritmusok számára, de a megerősítéses tanulással ezek a nehézségek könnyen kezelhetők. Az optimális policy változása lehetővé teszi a rendszer számára, hogy alkalmazkodjon a környezet változásaihoz, és így folyamatosan fejlődve érje el a céljait.

3. Kemelkedő teljesítményt tud nyújtani a legkülönbözőbb területeken, beleértve a játékokat, a robotikát és a pénzügyi szektort. A játékterületen olyan számítógépes rendszerek hozhatók létre, amelyek képesek akár emberekkel szemben is versenyképesek lenni. Olyan rendszerek hozhatók létre, amelyek képesek környezetükre reagálni és a céljaik elérésére alkalmazkodni. A pénzügyi szektorban a megerősítéses tanulás segítségével olyan rendszerek hozhatók létre, amelyek képesek például a tőkepiacokon optimalizált döntéseket hozni. Tehát széles körben hasznosítható és alkalmazható, és jelentős teljesítményt nyújthat a különböző területeken.

Hátrányai

1. A tanulás jelentős számítási erőforrásokat igényel a modellek képzéséhez és teszteléséhez, amelyek lehetnek egy adott alkalmazásra nézve költségesek. A pontos politika megtanulásához szükséges idő nagymértékben függ a modellek komplexitásától és a rendelkezésre álló adatok mennyiségétől is. Az eredmények azonban megfelelő idő és erőforrás befektetés esetén jelentős javulást hozhatnak a teljesítmény tekintetében.

2. Az értelmezhetőség hiánya miatt nehéz megérteni a tanult politika okait, ezért nehéz lehet előre jelezni a politika eredményét, és nehéz kijavítani a modell hibáit. A modell által tanult viselkedés megértése fontos lehet, ha szeretnénk továbbfejleszteni, vagy ha a modellnek kellene alkalmazkodnia az új környezeti feltételekhez.

3. Megfelelő jutalomfüggvény meghatározása nehéz lehet, amely pontosan visszaadja a kívánt viselkedést. Ez a legfontosabb kihívás, mert a helytelen jutalmazási rendszer rossz irányba tereli a modellt, ami nem fogja

elérni a kívánt céljait, vagy nem fog megfelelően teljesíteni. Az optimális jutalmazási rendszer meghatározása kritikus, mert az határozza meg a modell végleges viselkedését és hatékonyságát. A jutalmazás meghatározásának nehézsége csökkenthető a megfelelő mérőszámok használatával és a modell tesztelésével.

4. A megerősítéses tanulási algoritmusok esetleg suboptimális megoldásra konvergálhatnak a rossz felfedezés vagy a hiperparaméterek rossz választása miatt. A suboptimális megoldás elkerüléséhez szükséges lehet a modellek változatosságának bevezetése a tanulási folyamatban, vagy más, a tanulást segítő taktikák alkalmazása.

1.7. Környezeti jellemzők, melyekre a megerősítéses tanulás leginkább alkalmas

A megerősítéses tanulás nagyszerű eszköz a komplex környezetek kezelésére, ahol a hagyományos tanulási megközelítések nehézségekbe ütköznek. Az alábbi környezeti jellemzők esetén lehet a megerősítéses tanulás a leginkább alkalmas megoldás:

1. Ismert a környezeti modell, de nem áll rendelkezésre analitikus megoldás: Ha a környezeti modell ismert, de nincs lehetőség analitikus megoldás előállítására, akkor a megerősítéses tanulás segítségével a modell tanulhat a környezetből, és javíthatja a cselekvéseit a céljai elérésére.

2. Csak a környezet szimulációs modellje van megadva: Ha csak a környezet szimulációs modellje áll rendelkezésre, akkor a megerősítéses tanulás segítségével a modell megtanulhatja, hogyan viselkedjen a környezetben, és hogyan javítsa a cselekvéseit a céljai elérésére.

1.8. A Megerősítéses tanulás sikerei

Nagyszerű eredményeket hozott a társasjátékokban és a videojátékokban, mint például a Starcraft-ban, AlphaGo-ban és a Sudoku-ban. Az ágensek a megerősítés segítségével megtanulják, hogyan döntsenek a legoptimálisabb cselekedetek mellett egy adott helyzetben, és hogyan alkalmazzák ezen tudásukat a valós játékban, így lehetővé téve számukra, hogy tanuljanak az emberi játékosok stratégiáiból, saját megoldásokat fejlesszenek ki és folyamatosan javítsák a játékukat. A Go-ban és a sakkban az AI ágensek már legyőzték a világ legjobb emberi játékosait, és a videojátékokban is nagyszerű eredményeket, amely jó példa a gépi tanulás képességeinek alkalmazására a valós világban. Bár a megerősítéses tanulás korántsem új fogalom, a mély tanulás és a számítási teljesítmény terén a közelmúltban elért fejlődés lehetővé tette néhány figyelemre méltó eredmény elérését a mesterséges intelligencia területén.

2. fejezet

OpenAI Gym

A megerősítéses tanulási algoritmusok és kiértékelések készítéséhez, ellenőrzéséhez tökéletes választás lehet az OpenAI-tól a Gym[1] forráskód felhasználása, amely széles választékot nyújt különböző játékok előre megírt implementációjához és ezen játékokon való modellek tanításához. API-kon keresztül tudunk behatással lenni a környezeti változókra, amely a modellek magas skálázhatóságában és a teljesen precíz tanítási folyamat kialakításában is segíthet mindamellett, hogy nyomon követhetjük a modellünk fejlődését és teljesítményét.

A Gym különböző szimulátorokkal való kommunikációt támogat, beleértve a Mujoco-t vagy a Roboschool-t, amelyek szimulációs alapú környezetek. Az ilyen környezeteknek köszönhetően a Gym alkalmas olyan esetek betanulására, amelyeket a valóságban nem szeretnénk megkísérelni, például bonyolult vagy magas rizikófaktorú műveletek. Ezenkívül a környezetek szabályozhatóak, hogy megfeleljenek a kívánt fizikai vagy környezeti feltételeknek, ami növeli a rendszer pontosságát és megbízhatóságát. Az algoritmusok fejlesztése és tesztelése a Gym segítségével gyors és hatékony, mivel az algoritmusok valós időben futnak, így a visszacsatolás gyors és a hibák azonnal javíthatóak. Ezenkívül az OpenAI Gym könnyen skálázható, így nagyobb rendszereket is tesztelhetünk vele. A könyvtár ezen lehetőségeinek köszönhetően a fejlesztőknek szélesebb eszköztár áll rendelkezésükre, amely lehetővé teszi számukra, hogy szélesebb körű problémákat és alkalmazásokat oldhassanak meg a gépi tanulás területén.

A tanításainkhoz és méréseinkhez a Mujoco szimulátort használtuk, amelyben 3D-s humanoidot, egy sétáló és egy ugráló pálcát tanítottunk előre haladni. Szinte bármilyen környezetet tudunk szimulálni, saját objektumokkal létrehozva a világunk fizikai szabályai alapján definiált környezeteken szimulálva, többféle mozgásfolyamat betanítására. A fizikai szabályok természetesen kedvünk szerint felüldefiniálhatóak a környezetben.

Patrick Coady implementációja nagyon jól szemlélteti a megerősítéses tanulás és a Trust Region Policy Optimization (TRPO) algoritmus együttes felhasználásának mekkora sikere lehet különféle mozgások elvégzéséhez szükséges adatok feldolgozására és különféle problémák megoldására, éppen ezért mi ennek a forráskódnak a felhasználásával készítettük a modelljeinket és méréseinket.

2.1. MuJoCo

A Roboti LLC fejlesztette ki a Mujoco-t [2], amely az elejétől kezdve a modellalapú optimalizálást és az érintkezőkön keresztüli optimalizálást segítette. A Mujoco egy ingyenes nyílt forráskódú fizikai motor, amelynek célja, hogy segítse a kutatásokat és a fejlesztéseket egyaránt a robotikában, biomechanikában, grafikában és animációban, illetve olyan más területeken, ahol gyors és pontos szimulációra van szükség. A Mujoco rendkívül népszerű a különböző tanulási módszerek között, ami a benne rejlő lehetőségeknek köszönhető, mint például a realisztikus fizikai modellezés, a magas sebességű szimuláció, valamint a modell alapú optimalizálás lehetősége. Ennek köszönhetően a Mujoco alkalmas a számos nehézséggel járó feladatok megoldására, mint például egy humanoid szimulált robot járásának betanítására. A MuJoCo lehetővé teszi a számításigényes technikák, például az optimális vezérlés, a fizikailag konzisztens állapotbecslés, a rendszerazonosítás és az automatizált mechanizmustervezés felskálázását és alkalmazását összetett dinamikus rendszerekre, érintkezésben bő viselkedésmódokban. Hagyományosabb alkalmazásai is vannak, mint például a vezérlési sémák tesztelése és érvényesítése a fizikai robotokon való telepítés előtt, interaktív tudományos vizualizáció, virtuális környezet, animáció és játék.

2.2. Környezetek cselekvési - és megfigyelési tere

A Gym a MuJoCo-hoz több előre megtervezett környezetet is szolgáltat, mi ezekből a környezetekből választottunk ki kettőt, melyeknek a feladata, hogy egy irányba minél gyorsabban haladjanak anélkül, hogy elesnének, avagy egy előre definiált z index alá esne a törzsük koordinátája.

A Gym környezetek a szimulátorban való lépések megtételéhez egy úgynevezett cselekvési teret vár, amit majd az ágensünk határoz meg, és egy megfigyelési teret ad vissza, amely leírja a szimulátorban lévő környezet aktuális állapotát, melyből az ágens megtudja határozni a következő lépését.

A cselekvési tér a rendszerünk által elérhető valós és érvényes műveletek vagy választási lehetőségek halmaza, amikor kölcsönhatásba lép a megadott környezeti térrel. Meghatározza, hogy milyen cselekvésekre van lehetőség a rendszer részéről a környezetre való reakciók során. Tartalmazza a környezetben szimulált robotok különböző részeinek mozgással kapcsolatos lehetőségeit, amelyeket a szimuláció során végre kell hajtani a cél elérése érdekében.

A megfigyelési tér az a változó tartomány, amely megadja az aktuális állapotát a környezetével való kölcsönhatás során egy ágensnek. Az ágens érzékeléseinek, mint például a testhelyzet, sebesség, erő, stb... állapotát tartalmazza. Meghatározza, hogy milyen információk állnak rendelkezésre a szimuláció során a modell állapotának meghatározásához. A megfigyelési tér értékeit figyelembe véve a rendszer képes reagálni a környezet változásaira, és az alapján hozhat döntéseket a következő lépések megtervezése során.

Ha a környezet unhealthy, azaz elesett (A ‘z’ koordinátán előre definiált érték alá került a törzs) vagy ha eléri az 1000 timestep-et (iterációt), akkor vége egy epizódnak.

2.3. A felhasznált implementáció összegzése

Patrik Coady implementációját alapul véve tanítottuk a modelljeinket az általa ajánlott konfigurációkkal és ezen a kódon tettük a változtatásokat is az újabb módszerek kidolgozásához. Patrick Coady projektjének fő célja az volt, hogy az általa fejlesztett algoritmust többféle robotikai problémára alkalmazva, megoldásokat találjon, mint például a pendulum vagy a spider modellek tanítása, valamint az humanoid robot sebességének növelése. Az általa implementált Trust Region Policy Optimization algoritmus általános megoldást nyújt a robotikai kihívásokra, amelyeket a szimulációk során tanítunk meg és optimalizálunk. A projekt sikeres megvalósítása érdekében nem manuálisan állította be a hiperparamétereket, hanem automatikus optimalizációs technikákat alkalmazott. Így biztosítva volt, hogy az algoritmus megfelelően alkalmazkodjon a környezethez és a modell jellemzőihez, valamint a lehető legjobb eredményt érje el. Ez a folyamat nehézkes, mert a 10 modell között nagyfokú variációk vannak a szükséges hiperparaméterek, az action és observation space méretek, valamint a szükséges kalkulációk tekintetében. Minden modell más és más környezetet igényel, így a hiperparaméterek optimalizálásának folyamata egyedi megközelítést igényel minden egyes modell esetén. Azonban az alkalmazott algoritmus képes volt sikeresen megoldani a kihívást, és szinte az összes OpenAI Gym Mujoco környezetre készített probléma megoldásának eredményei az élmezőnyben voltak, vagy nagyon közel hozzá.

3. fejezet

Kísérletek és megvalósítás

A mérések, tanítások és a felhasznált környezet pythonban íródtak, a szimulátor C-ben. Kísérleteink két meghatározó mérőszáma a sebesség és lépésszám. A sebességet azalapján számoltuk, hogy egy iterációban a szimulátorban az x tengelyen mennyit haladt a modell. A lépésszámot pedig a megtett iterációk alapján számoltuk. Kísérleteink a stabilitás javítására szolgáltak. A MuJoCo szimulátorban az OpenAI GYM Humanoid, Hopper és Walker2d környezeteket vizsgáltuk két különböző megközelítéssel, amelyekben a tanítási folyamat során változtattuk a maximális iterációs korlátot és batch méretet. Egy környezeten belül három modell típust különböztetünk meg, melyeknek a baseline, statikus és dinamikus neveket választottuk. A baseline az ajánlott konfigurációval, a kód változtatása nélkül tanított modellek. A statikus modelleknél a tanítási folyamat közepénél megnöveltük az iterációs korlátot a kétszeresére és a tanítás második felének hosszát feleztük. A dinamikus modelleknél ha egy modell az adott batchben elérte a batch kétharmadában az iterációs korlátot, akkor a batch méretét csökkentettük a felére egy minimum értékig és az iterációkat növeltük a kétszeresére egy maximum értékig. A környezetekben mindegyik típusra 5 modellt tanítottunk. Egy tanítási folyamatot felosztottunk húsz egyenlő részre, amelyek a modell mentési pontjaiként szolgáltak, ezzel nyomon követve a fejlődését, így összesen húsz checkpoint jött létre egy tanítás alatt. A kiértékeléseket minden modell minden mentési pontjára 10 különböző seed-en végrehajtottuk egészen bukásig futtatva, ezzel mérve a stabilitásukat a tanítás különböző szakaszában és a különböző megközelítésekhez képest.

3.1. Epizód és iterációs szám módosítás tanítás közben

3.1.1. Baseline

Az iterációs korlát ezer volt húsz batchenkénti ágens frissítésselel mindhárom környezetben. A humanoidot pedig kétszázezer epizódon, a hopper-t harmincezer epizódon, a walker2d-t huszonötezer epizódon keresztül tanítottunk.

3.1.2. Statikus

Az iterációs korlát a tanítások közepénél megduplázódott és a kiértékeléshez szükséges batch méret ekkor felére csökkent, illetve innentől a baseline epizódszám/4 epizódig folytatódott a tanítás.

3.1.3. Dinamikus

Itt már kezdetben az iterációs korlát csak százhuszonöttel indult, de százhatvan batch mérettel. Az iterációs korláton dupláztunk és a batch méretét feleztük, amikor a modell az aktuális batchnek a 2/3-án elérte az iterációs korlátot tanítás közben. Ezeket a duplázásokat és felezéseket a tanítás alatt addig folytattuk, ameddig az iterációs korlát négyezerre nem nőtt és a batch méret ötre nem csökkent. Öt baseline modellnél megnéztük, hogy egy tanítás alatt mennyi lépést tesznek összesen a környezeten, és ezen összegek átlagát választottuk a dinamikus algoritmus megállási feltételének, ha eléri a modell a tanítás alatt ezt a lépésszámot. A Humanoid baseline átlag lépése a tanítás alatt százötvenmillió, hopper-nek huszonegymillió-nyolcszázezer, a Walker2d-nek huszonegymillió-háromszázezer volt.

3.1. táblázat. A környezetek és típusaik adatgyűjtési stratégiái

	All	**Humanoid**		**Hopper**		**Walker2d**	
	Batch Méret	**Ep**	**It**	**Ep**	**It**	**Ep**	**It**
Baseline	20	200000	1000	30000	1000	25000	1000
Statikus	20/10	100000/50000	1000/2000	15000/7400	1000/2000	12400/6000	1000/2000
Dinamikus	[5;160]	-	150 000 000	-	21 800 000	-	21 300 000

3.2. táblázat. Dinamikus modellek epizód és iterációszám skálázása

	Episode	Iteration	Next State
State 1	160	125	107
State 2	80	250	53
State 3	40	500	27
State 4	20	1000	13
State 5	10	2000	7
State 6	5	4000	-

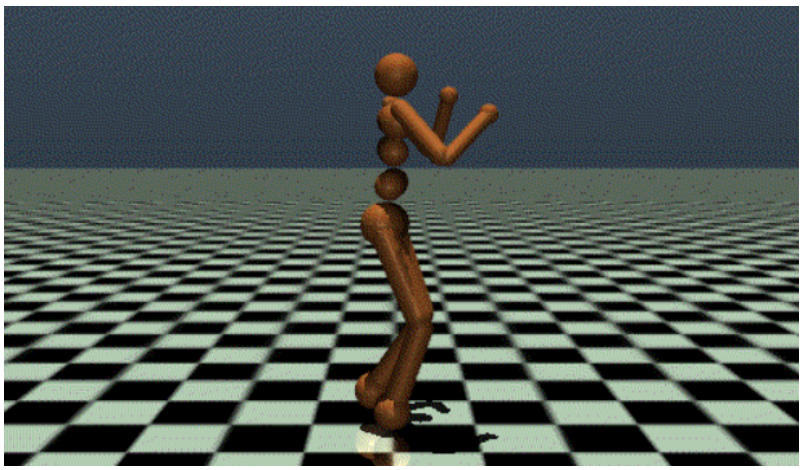
3.2. Robotvezérlő környezetek

Két környezetet vizsgáltunk, amelyeket lentebb definiálunk. Kiválasztás szempontjából fontos volt, hogy a modellek tanítás közben ha instabilak könnyen eltudjanak esni, ne ragadjanak be, mint például az Ant környezet, amely a négy lába miatt könnyedén megállt egyhelyben és nem mozdult semerre.

	Obs space	Act space	1st L	2nd L	3rd L	4th L	Learning rate
Humanoid	(376,)	(17,)	3760	799	170	17	3.18E-05
Hopper	(11,)	(3,)	110	57	30	3	1.19E-04
Walker2d	(17,)	(6,)	170	100	60	6	9.00E-05

3.2.1. Humanoid

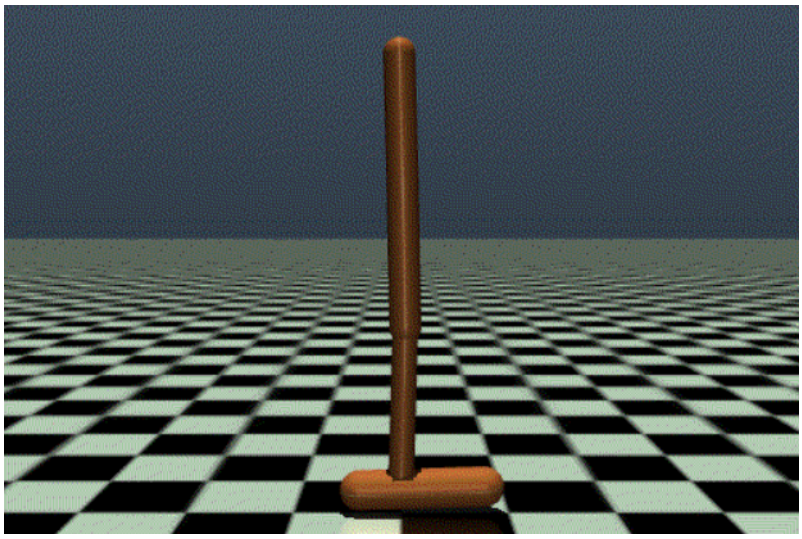
A Humanoid környezet a Tassa, Erez és Todorov által a „Synthesis and stabilization of complex behaviors through online trajectory optimization” című munkában bemutatott környezeten alapul. A 3D kétlábú robotot úgy tervezték, hogy szimulálja az embert. Van egy törzse egy pár lábbal és karral. A lábak és karok egyenként két láncszemből állnak. Egy akció a csuklóponton alkalmazott nyomatékokat reprezentálja.



3.1. ábra. Humanoid környezet a MuJoCo szimulátorban

3.2.2. Hopper

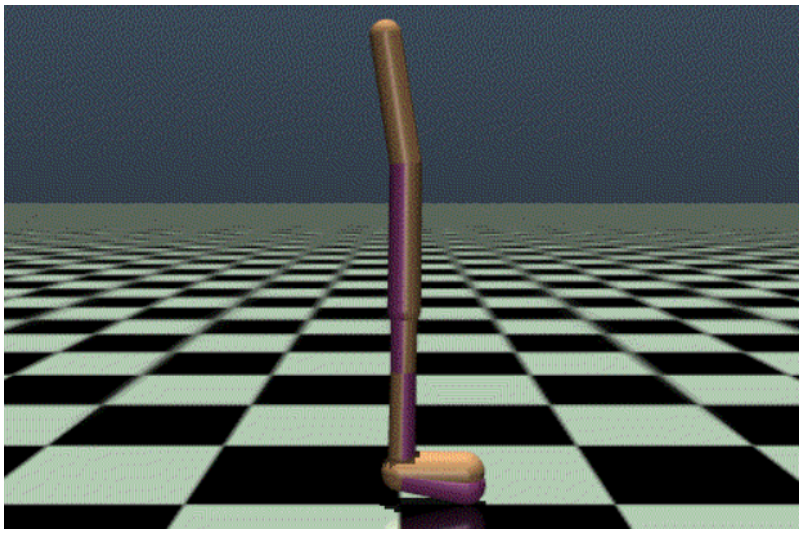
A Hopper környezet Erez, Tassa, és Todorov “Infinite Horizon Model Predictive Control for Nonlinear Periodic Tasks” című munkáján alapszik. A környezet célja a független állapot- és vezérlőváltozók számának növelése a klasszikus vezérlő környezetekhez képest. A hopper egy kétdimenziós, egylábú figura, amely négy fő testrészből áll - a törzsből a felső részen, a combból középen, a lábból alul, és egyetlen lábfejből, amelyen az egész test nyugszik. A cél az, hogy a négy testrészt összekötő három csukló pontra ható nyomaték alkalmazásával előre (jobb) irányba irányuló ugrásokat végezzen.



3.2. ábra. Hopper környezet a MuJoCo szimulátorban

3.2.3. Walker2d

A Walker2d környezet a Hopper környezetre épül, amely Erez, Tassa és Todorov „Infinite Horizon Model Predictive Control for Nonlinear Periodic Tasks" munkáján alapul. A hopper környezetet annyiban módosították, hogy hozzá adtak még egy lábat, így lehetővé téve a robot számára, hogy járjon az ugrálás helyett. A többi Mujoco környezethez hasonlóan ez a környezet is a független állapot- és kontrollváltozók számának növelését célozza a klasszikus kontrollkörnyezetekhez képest. A Walker2d egy kétdimenziós, kétlábú figura, amely négy fő testrészből áll - egy törzsből a felső részen (a két láb a törzs után kettéválik), két combból középen a törzs alatt, két lábból alul a combok alatt, és két lábfejből, amelyek a lábakhoz kapcsolódnak, és amelyeken az egész test nyugszik. A cél az, hogy a hat testrészt összekötő hat csuklópontra ható nyomaték alkalmazásával koordináltan az egész test előre (jobb) irányba mozogjon.



3.3. ábra. Walker2D környezet a MuJoCo szimulátorban

3.3. Hálózati Architektúra

A learning rate heurisztikáját a neurális hálózat mérete alapján számolja az algoritmus amelyet a Hopper környezetben hangoltak (=9e-4 / np.sqrt(int(np.sqrt((obs dim * 10) * (act dim * 10))))). A tanítás során egy discount factor-ral (alapértelmezetten 0.995) is segítünk új és esetleg jobb lépéseket találni, és maximalizáljuk a hosszabb távú jutalomfüggvényt. Egy lambda (alapértelmezetten 0.98) változót használunk az általánosított előnybecsléshez (GAE). A tanítási folyamat során két modell-t optimalizálunk, melyek a policy és value function hálók.

3.3.1. Policy

Megjósolja a következő lépést. A bemeneten és a rejtett rétegeknél véletlenszerű normál inicializáló kernel inicializálást és tanh aktivációs függvényt használunk. A bemeneti réteg mérete observation dim*10. Két rejtett réteggel rendelkezünk, ahol az első méretét az int(sqrt(observation dim * action space dim)) formulával számoljuk. A második rejtett réteg mérete act dim*10.

3.3.2. Value Function

A legjobb lépéseket keresi. A bemeneti és rejtett rétegeken véletlenszerű-normális inicializáló kernel inicializálást és tanh aktiválási függvényt használunk. A bemeneti réteg mérete observation dim*10. Két rejtett réteggel rendelkezik. Az első méretét az int(sqrt(obesvation-space-dim * 5)) függvénnyel számoljuk. A második rejtett réteg mérete 5. Egy négyzetes veszteségfüggvényt és egy AdamOptimizer-t, illetve egy replay buffer-t is használ, amely az előző tanulási lépésben gyűjtött adatokból áll. Ezt összekapcsoljuk a tanítási adatokkal, megkeverjük az adatokat és frissítjük velük a modellt.

3.4. Modell szelekció

A kiértékelések utáni adathalmazokon a legjobb lépésszámok, a legjobb sebesség, és a kettő kombinációja alapján különítettük el és vizsgáltuk a modelljeinket, szeparáltan a különböző környezetektől és típusaiktól és összehasonlítottuk a kiválasztott modellek eredményeit az eltérő típusok szerint sebesség és lépésszám tekintetében.

Mentési Pont	Sebesség	Norm. Sebesség	Lépésszám	Norm. Lépésszám	Norm. Sebesség és - Lépésszám szorzata	Szelekció
200000	3.941	1	2657.6	0.639	0.639	Legnagyobb Sebesség
180000	3.720	0.943	3337.2	0.803	0.758	Legnagyobb Norm. Sebesség és - Lépésszám szorzata
60000	1.242	0.315	4153.1	1	0.315	Legnagyobb Lépésszám

A tíz seed-nyi kiértékelés után átlagoltuk az eredményeket, ezzel csökkentve az adatok szórásán, majd kiválasztottuk az adatokból azokat a mentési pontokat, ahol adott modell a legnagyobb lépésszámot vagy sebességet tette. A harmadik szelekcióba úgy választottunk, hogy normalizáltuk modelleken belül a sebességet és lépésszámot, majd a kettő érték szorzatát rendeltük egy mentési ponthoz, és megnéztük ez a szorzat hol volt a legnagyobb.

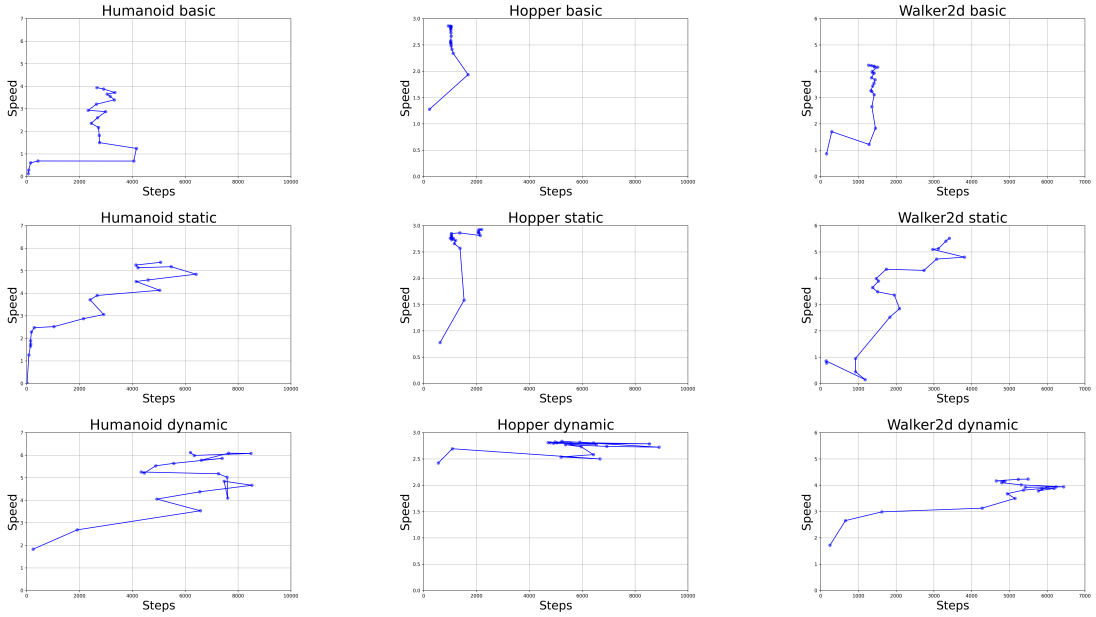
4. fejezet

Eredmények

Az általunk vizsgált robotvezérlő környezetekben végzett elemzéseink során megfigyelhető, hogy a stabilitási mérőszámok a tanítási folyamat egy bizonyos pontjától kezdve romlásnak indulnak, vagy elérnek egy maximum értéket és azt követően stagnálni kezdenek. Ezt a problémát az adatgyűjtési stratégia módosításával kezeltük. A tanítási folyamat kezdetén nagy batch méretet alkalmazunk kis iterációs korláttal, amit fokozatosan növelünk ahogy a modell több lépést képes végrehajtani, ezzel egyidejűleg a batch méretet is csökkentjük, ami lehetővé teszi, hogy már a tanítás kezdeti szakaszában maximálisan kiaknázzuk a rendelkezésre álló memóriakapacitást, és javítsuk a tanulási folyamat stabilitását. Ez a megközelítés segít a korábbi korlátokon túllépni, és javítja az ágensek robosztusságát.

4.1. Modellek instabilitása

Az alábbi ábrákon jól megfigyelhető, hogy a tanítási folyamat egy bizonyos szakaszától kezdve a modellek a gyorsabb mozgás érdekében feláldozzák stabilitásukat, ami hosszú távon az összteljesítményük romlásához vezet. Az optimális mozgási sebesség és stabilitás közötti egyensúly elérése kulcsfontosságú lehet a modellek kifejlesztésében. A stabilitás hiánya nemcsak hogy akadályozza a modell konvergenciáját, de kiszámíthatatlan viselkedéshez és nem megbízható eredményekhez is vezethet.

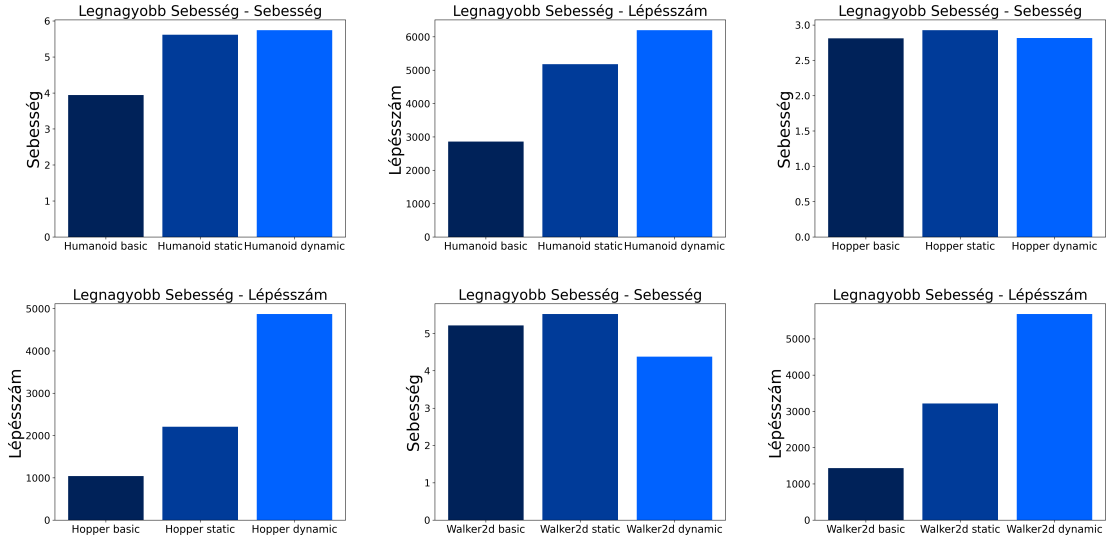


4.1. ábra. Modellek tanulási görbéje a sebesség és lépésszám tekintetében

4.2. Modellek stabilitásának növelése

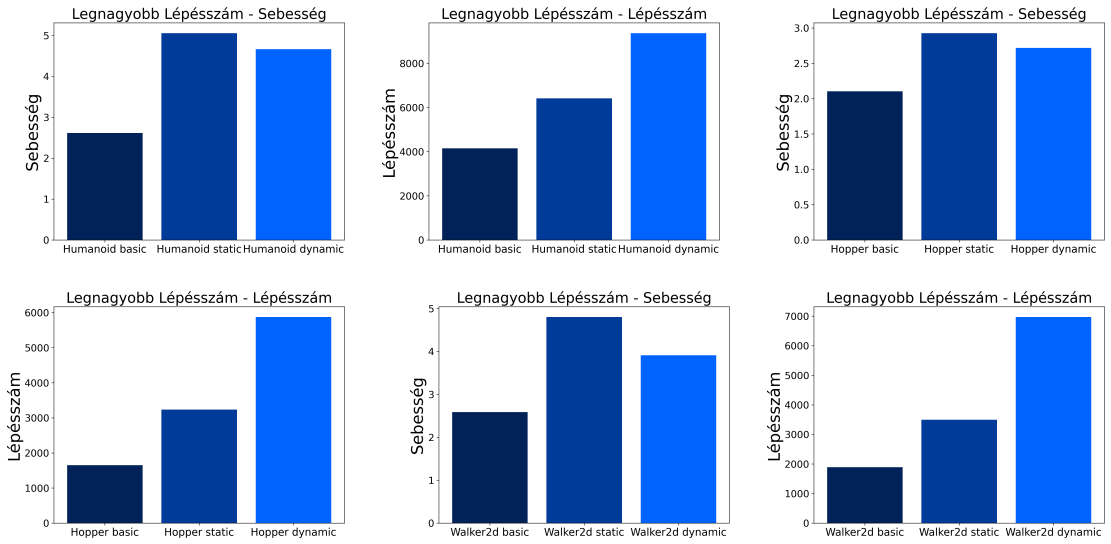
A kutatásunkban alkalmazott statikus és dinamikus módszerek vizsgálata során azt tapasztaltuk, hogy jelentősen hozzájárultak a modellek stabilitásának növeléséhez. A három szelekciós modell kiválasztással összehasonlítottak az eredményeket környezeteken belül lépésszám és sebesség tekintetében.

Legnagyobb sebesség kiválasztással sebesség szempontjából az ágensek jobb eredményeket értek el a baseline-nál és a dinamikus modelleknél, azonban lépésszámban alul maradt a dinamikussal szemben.



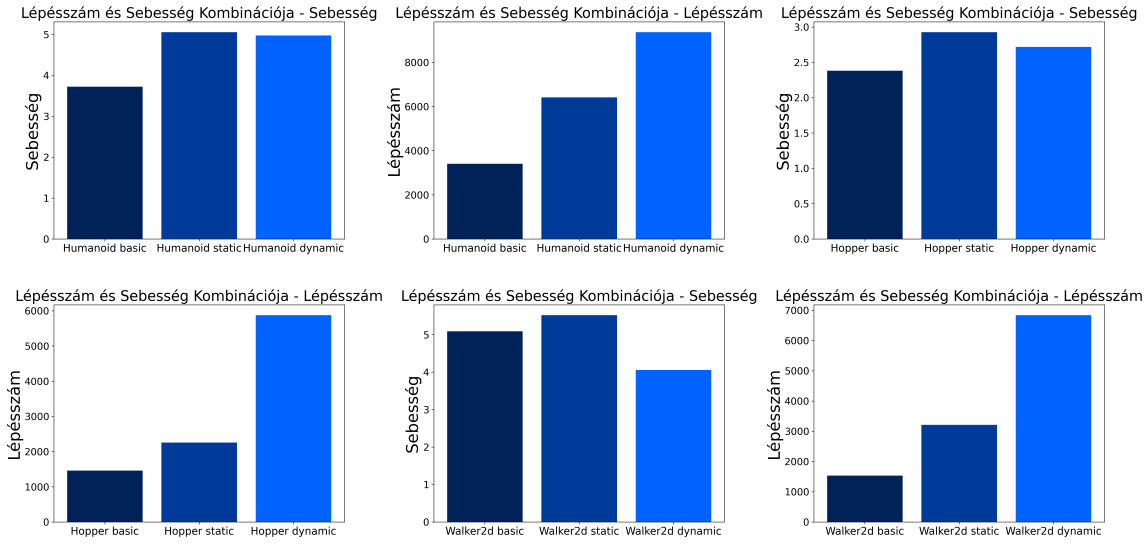
4.2. ábra. Legnagyobb sebesség alapján kiválasztott modell checkpointok mediánjai

Legnagyobb lépésszám kiválasztással a statikus modellek szintén sebességben jobban teljesítettek mindkét típusnál, azonban továbbra is domináns lépésszám tekintetében a dinamikus típus.



4.3. ábra. Legnagyobb lépésszám alapján kiválasztott modell checkpointok mediánjai

A kombinált kiválasztásnál is láthatjuk, hogy a baseline modelleknél a másik két megközelítése a problémának jobban teljesít, sebesség szempontjából a statikus, lépésszám tekintetében a dinamikus modellek dominálnak, tehát a tanítási folyamat alatt egységesen nőtt a sebesség és lépésszám a modelleknél, nem mutatnak eltéréseket.



4.4. ábra. Legnagyobb normalizált lépésszám és sebesség kombinációja alapján kiválasztott modell checkpointok mediánjai

5. fejezet

Konklúzió

A kísérleteink során észleltük, hogy a modellek stabilitása egy bizonyos tanítási szakasz után romlani kezdett. Ezt a problémát sikerült hatékonyan kezelnünk az adatgyűjtési stratégia módosításával, ami kulcsfontosságú lépés volt a stabilitás növelésében. Az adatgyűjtési stratégia módosítása magában foglalta a batch méret és az iterációs korlátok célzott beállítását. Ez a megközelítés a rendelkezésre álló erőforrásokat is optimálisabban használta fel a megszokottnál. Az eredmények kiértékelése során megállapíthatjuk, hogy a statikus és dinamikus adatgyűjtési stratégiák jelentősen javították a modellek stabilitását. Az eredmények pedig rámutattak arra, hogy a sebesség és a lépésszám optimális kombinációjának kiválasztása kulcsfontosságú lehet a hatékonyabb és stabilabb működés érdekében. Összefoglalva, a tanítási folyamat során tapasztalt stabilitási problémák kezelése az adatgyűjtési stratégia átgondolt módosításával nemcsak hogy javította a modellek teljesítményét, hanem optimalizálta az erőforrások felhasználását is.

Felhasznált eszközök és oldalak

.0.1. ChatGPT

nyelvhelyesség korrigálás, adatfeldolgozás, latex kódgenerálás

.0.2. Gépi tanulás alapfogalmak

https://www.inf.u-szeged.hu/∼rfarkas/ML22/

.0.3. Megerősítéses tanulás alapfogalmak

https://www.inf.u-szeged.hu/∼korosig/teach/rl_2023.html

Irodalomjegyzék

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym*. arXiv. doi:10.48550/ARXIV.1606.01540.

[2] Howell, T., Gileadi, N., Tunyasuvunakool, S., Zakka, K., Erez, T., & Tassa, Y. (2022). *Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo*. arXiv. doi:10.48550/ARXIV.2212.00541.

[3] Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., . . . Michalewski, H. (2019). *Model-Based Reinforcement Learning for Atari*. arXiv. doi:10.48550/ARXIV.1903.00374.

[4] Lample, G., & Chaplot, D. S. (2017, February). *Playing FPS Games with Deep Reinforcement Learning*. Proceedings of the AAAI Conference on Artificial Intelligence, 31. doi:10.1609/aaai.v31i1.10827.

[5] Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P. (2015). *Trust Region Policy Optimization*. In F. Bach, D. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning (Vol. 37, pp. 1889–1897). Lille: PMLR. Source: https://proceedings.mlr.press/v37/schulman15.html.

[6] Witt, C., Peng, B., Kamienny, P.-A., Torr, P., Böhmer, W., Whiteson, S. (2020, March). *Deep Multi-Agent Reinforcement Learning for Decentralized Continuous Cooperative Control*. Deep Multi-Agent Reinforcement Learning for Decentralized Continuous Cooperative Control. arXiv. doi:10.48550/ARXIV.2003.06709.