VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

INFORMATIKOS KATEDRA

Bakalauro baigiamasis darbas

Skatinamojo mokymo paremto neuroninias tinklais optimizavimas taikant žinių pernešimą

(Optimizing Deep Reinforcement Learning Process by Applying Transfer Learning)

Atliko:

Martynas Jakimčikas (parašas)

Darbo vadovas:

Lekt. Andrius Grevys (parašas)

Darbo recezentas:

Lekt. Irmantas Radavičius (parašas)

Santrauka

Šiame darbe buvo mėginama taikyti žinių pernešimo metodus giliam skatinamajam mokyme, kad optimizuoti ir paspartinti mokymosi procesą. Darbe bendrai apžvelgiama mašinų mokymasis ir jo šakos. Taip pat trumpai apžvelgiama skatinamojo mokymo raida, bendra teorija ir algoritmai kurie reikalingi atlikti tyrimus. Apibūdinamos dažniausiai naudojamos aplinkos mokyti agentus, bei įrankiai kuriais yra naudojamasi kurti dirbtinius neuoroninius tinklus. Darbo gale atliekami keturi eksperimentai, kurie leidžia įvertinti ar įmanomas gilaus skatinamojo mokymo optimizavimas taikant žinių pernešimą

Raktiniai žodžiai: Gilus skatinamasis mokymas, dirbtiniai neuroniniai tinklai, žinių pernešimas, OpenAI Gym, Atari, Q-mokymasis, Aktoriaus ir kritiko mokymasis

Summary

In this thesis we try to apply transfer learning to deep reinforcement learning, to optimize and speed up the learning process of an agent. The thesis reviews machine learning types and branches. The history of reinforcement learning is also briefly covered, together with the needed theory and algorithms for the experiments. We also take a look at the environments, frameworks and tools that we use to train agents and construct artificial neural networks. At the end of the thesis we complete four experiments, to evaluate if we can optimize deep reinforcement learning by applying transfer learning

Keywords: Deep Reinforcement Learning, Artificial neural networks, transfer learning, OpenAI Gym, Atari, Q-learning, Actor critic learning

Įvadas ***5***

1. Mašinų mokymasis ***7***

1.1. Mašinų mokymasis ir jo atšakos **7**

1.2. Dirbtiniai neuroniniai tinklai ir gilus skatinamasis mokymasis **8**

1.3 Žinių pernešimas **8**

2. Gilus skatinamasis mokymasis ***10***

2.1 Skatinamojo mokymo raida **10**

2.2 Markovo procesai ir mokymo problemos formulavimas **10**

2.3 Vertės funkcija ir Bellmano lygybė **12**

2.4 Aplinkos tyrinėjimo problema **13**

2.5 Konvoliuciniai neuroniniai tinklai **14**

2.6 Gilus skatinamasis mokymas **14**

2.7 Problematika **15**

2.8 Žinių pernešimas. **15**

3. Gilaus skatinamojo mokymo optimizavimas ***17***

3.1 Karkasai kurti dirbinius neuoroninių tinklų modelius **17**

3.2 Mokymosi aplinkos „OpenAI Gym“ ir „Arcade learning environment“ **17**

3.2.1 „LunarLander-v2“ 18

3.2.2 „Acrobot-v1“ 19

3.2.3 „Space invaders-v4“ 20

3.2.4 „Breakout-v4“ 20

3.3 Q-mokymasis **21**

3.4 Aktoriaus ir kritiko mokymasis **24**

3.5 Eksperimentų išvados **28**

Išvados ***30***

Šaltiniai ***32***

Įvadas

Gebėjimas kurti ir dirbti su dirbtinio intelekto sistemomis ir mašinų mokymosi algoritmais yra vienas iš penkių labiausiai paklausių įgūdžių dabartinėje informacinių sistemų darbo rinkoje, pagal „linkedin“ tinklapio statistiką[LIN]. Dirbtinis intelektas, toliau DI, per labai trumpą laiką tapo neatsiejama visuomenės dalis. Jis yra taikomas daugelyje technologijų, pavyzdžiui, telefonuose balsą versti į tekstą, marketingo ir reklamų taikyme, kai vartotojų grupėms rodomi specialūs pasiūlymai (angl. targeted advertisement) ir, be abejo, socialiniuose tinkluose. Turbūt žinomiausias ir daug populiarumo sulaukęs socialinis tinklas „Tik Tok“, skirtingai nei kiti socialiniai tinklai kaip „Facebook“ ar „Twitter“, yra labai stipriai paremtas dirbtiniu intelektu ir mašinų mokymu. Naudojant šį tinklą yra renkama informacija apie vartotojo pomėgius, perduodama dirbtiniam intelekto moduliui, kuris vėliau pagal ją atrenka vaizdo įrašus ir kanalus, kurie domintų naudotoją, taip skatindami jį praleisti daugiau laiko naudojantis pačia aplikacija.

Pagrindinės priežastys, kodėl dirbtinis intelektas ir mašinų mokymas susilaukė tokio plataus taikymo ir susidomėjimo yra, tai kad šiuo metu yra labai daug duomenų ir standartiniais metodais ne visada galima išgauti žmogui suprantamą rezultatą arba yra problemų kurias tiesiog nėra žinoma kaip išspręsti naudojantis algoritmais. Taip pat labai sparčiai išaugo kompiuterių skaičiavimo galia bei pats kompiuterio procesorius tapo mažesnis, ko pasėkoje DI pradėtas taikyti daugiau sferų. Bet šiuo momentu DI ir mašinų mokymas nėra atsakymas į visas kylančias problemas, ši mokslo sritis tai pat turi savo ribas ir problemas.

Dirbtinį intelektą treniruoti arba mokyti, priklausomai nuo sprendžiamos problemos, gali būti labai brangu, tiek finansiniu atžvilgiu, tiek laiko atžvilgiu. Pats modelio kūrimas vyksta iteratyviu procesu. Yra suprogramuojamas modelis, bandoma jį mokinti spręsti problemą ir stebimi jo rezultatai. Apžvelgus rezultatus daromos išvados pagal kurias tobulinamas modelis ir toliau vykdomas procesas tol kol mus pradeda tenkinti rezultatas. Taip pat nėra jokio metodo sužinoti kiek ir kokių duomenų reikia, kad mūsų modelis galėtų atlikti jam duotą užduotį su tikėtinu tikslumu. Vienas iš metodų kaip galima spręsti šią problemą yra bandyti per naudoti DI modelius, kurie jau gerai moka spręsti kitą panašią problemą kaip pradinį tašką kitom užduotim spręsti. Tai su paprastina duomenų kiekio problemą, nes modelis jau geba gerai apibendrinti duomenis ir jam reikia išmokti tik tam tikrus kitų duomenų bruožus.

Taigi šio darbo tikslas bus mėginti optimizuoti DI modelių mokymąsi, pernešant žinias iš vieno modelio į kitą. Tas bus daroma konkrečioje mašinų mokymo atšakoje – gilus skatinamasis mokymas. Pradėsime nuo bendro suvokimo, kas yra mašinų mokymas, bei kokie yra jo tipai, aptarsime dirbtinius neuroninius tinklus ir kaip taikomas žinių pernešimas. Po to pereisime prie

skatinamojo mokymo jo raidos ir kaip šiame mokymosi procese yra taikomi dirbtiniai neuroniniai tinklai. Apžvelgsime kelis gerai žinomus ir plačiai taikomus skatinamojo mokymo algoritmus, kaip „Q-learning“, „A2C“ bei „A3C“. Pamėginsime jais naudojantis iš mokinti žaisti kompiuterinius žaidimus naudojantis „Open AI Gym“ įrankiu ir pamėginsime optimizuoti procesą pernešdami agentų įgytas žinias žaidžiant vieną žaidimą į kitą. Po atliktų tyrimų įvertinsime žinių pernešimo naudą skatinamajam mokymui.

1. Mašinų mokymasis

1.1. Mašinų mokymasis ir jo atšakos

Mašinų mokymasis, toliau MM – kompiuterių ir dirbtinio intelekto mokslo atšaka nagrinėjanti algoritmus, kurie automatiškai patys gerina savo veikimą. MM algoritmai konstruoja matematinius modelius pagal jiems duotus duomenis, kurie dažniausiai vadinami treniravimosi duomenys (angl. training data), tam kad galėtų priimti sprendimą arba atlikti spėjimą pagal duotus realius duomenis. Tokie algoritmai yra gan artimi statistiniams skaičiavimams, kurie bando daryti spėjimą pagal tikimybių pasiskirstymą ir dažniausiai taikomi užduotims kurioms parašyti algoritmą nėra žinoma kaip. Pavyzdys būtų elektroninių laiškų filtravimas ieškant šiukšlių ir brukalo, automatinis automobilio valdymas, NP-sudėtingumo problemoms ir kai idealus tikslumas nėra svarbus. MM turi tris pagrindines atšakas:

- Prižiūrimas mokymasis (angl. supervized learning), toliau PM – konstruoja duomenų matematinį modelį, kai yra pateikiama įvesties ir išvesties duomenys. Tai yra daroma generuojant funkciją, kuri gavus įvesties vektorių turi išmokti aproksimuoti išvestį. Mokymasis šiuo atveju vyksta minimizuojant baudos funkciją (angl. loss function), kuri parodo skirtumą tarp mūsų PM generuojamos funkcijos ir tikrosios reikšmės. Tokio tipo mokymasis taikomas dažniausiai klasifikacijos ir regresijos problemoms, pavyzdžiui, iš duotų transakcijų nustatyti, kurios yra plauna pinigus ir kurios ne, arba iš duoto teksto nustatyti jo tematiką.
- Neprižiūrimas mokymasis (angl. unsupervized learning), toliau NM – konstruoja duomenų matematinį modelį, kai yra pateikiami tik įvesties duomenys. Dažniausiai bandoma rasti duomenų struktūra, spiečių (angl. cluster), grupavimą ar duomenų sąryšį. Mokymosi procesas vyksta bandant ieškoti panašumų ir skirtumų tarp duotų duomenų ir pagal tai priimti sprendimą gavus nematytus duomenis. Šis mokymas taikomas dažniausiai duomenų analizėje (angl. data mining), pavyzdžiui, asociacijų analizė - duotos parduotuvės transakcijos ir pagal jas rasti taisykles kurios nurodytų, kad jei yra perkamas produktas A, tai bus perkamas ir produktas B.
- Skatinamasis mokymasis (angl. reinforcement learning), toliau SM – Bando iš mokinti virtualų agentą rinktis veiksmus, kurie maksimizuotų atlygį darant užduotį kokioje nors aplinkoje. Aplinka dažniausiai aprašoma, kaip „Markovo sprendimų procesas“ (angl. „Markov decision process“. SM yra naudojamas tada kai matematinis modelis nėra žinomas, pavyzdžiui autonominiuose automobiliuose ar mokantis žaisti žaidimą.

1.2. Dirbtiniai neuroniniai tinklai ir gilus skatinamasis mokymasis

Dirbtinis intelektas – kompiuterių mokslų šaka, kurios tikslas kurti modelius ir mašinas, kurios gali atlikti užduotis reikalaujančias žmogaus intelekto. Pagrindinis komponentas yra dirbtinis neuronas, kuris yra modeliuojamas pagal biologinio neurono abstrakciją. Neuronus sujungia tarpusavyje gauname dirbtinį neuroninį tinklą, kur kiekvienas neuronas turi $(x_0, x_1, x_2 \ \dots \ x_n)$ įvesčių ir vieną išvestį, o kiekviena jungtis yra kaip sinapsė nešanti signalą. Signalas DNT atveju yra realus skaičius, kurį gauname atlikę matematines operacijas su įvesties duomenimis ir neurono svoriais $(\theta_0, \theta_1, \theta_2 \ \dots \ \theta_n)$. DNT mokymosi procesas yra optimalių svorių ieškojimas, su kuriais mūsų tinklas gavęs įvestį grąžintų norimą rezultatą.

Gilus skatinamasis mokymasis (angl. deep reinforcement learning), toliau GSM – Tai yra skatinamasis mokymasis, taikant DNT mokymosi algoritmo procese. Dirbtinių neuroninių tinklų taikymas skatinamajame mokyme prasidėjo XX a. ir populiarumas per paskutinius metus smarkiai išplito, nes išaugo kompiuterių skaičiavimo gale, rasti nauji efektyvūs modeliai spręsti problemas bei buvo sukurta sistemų, įrankių ir karkasų, kurie palengvina vystymą šios mokslo šakos. GSM taikymo sferos irgi tapo plačiau taikomos, pradėta taikyti finansuose, kalbos apdorojime, robotikoje ir žaidimuose. Per paskutinius 5 metus GSM pasiekė nemažai proveržių tokių kaip “Deep Q-network” - sprendžia vieno agento mokymosi ir kontroliavimo problemą aplinkoje. “Google DeepMind” kompanijos GSM algoritmai, su kuriais pavyko įveikti tokius žaidimus kaip Go ar Shogi, naujausias „MuZero“ sugeba ne tik žaisti šiuos žaidimus, bet ir visus „Atari 2600“ tipo žaidimus.

Dirbtinis intelektas kaip ir minėta ankščiau turi savo ribas ir negali išspręsti visų problemų, GSM nėra išimtis. DNT mokymas yra interpretuojamas, kaip „Black box“, kurio veikimą įrodyti šiuo metu trūksta matematinio pagrindo. Hiper-parametrų pasirinkimui nėra jokio apibrėžto metodo, jie yra randami atliekant eksperimentą su skirtingomis reikšmėmis ir pasirenkat tas, su kuriomis mokymosi rezultatas geriausias. Giliam skatinamajam mokyme iškyla eksperimento pakartojimo problema, nes aplinkose kuriose yra mokomasi yra dinamiškos. Tačiau atsižvelgus į pasiekimus per pastaruosius metus ir kaip trumpai ši mokslo šaka yra nagrinėjama, galima tikėtis, kad GSM taikymas ir toliau plis kasdieninėse aplikacijose.

1.3 Žinių pernešimas

Žinių pernešimas (angl. transfer learning) – Kai bandoma naudoti jau iš mokytą MM modelį kaip pradžios tašką naujam modeliui mokinti. Tas gan dažnai taikoma kompiuterių matyme (angl. computer vision), kur turime modelį kuris jau gerai gali atpažinti vienus objektus nuotraukoje,

pavyzdžiui, šunis ir tęsiam jo mokymą, kad galėtų atpažinti kitus objektus, pavyzdžiui katinus. Tas yra daroma dėl to, nes DNT paremti MM modeliai reikalauja daug duomenų, kad pasiektų gerą rezultatą. Mokymo pradžioje modelis bando bendrai išsiaiškinti duomenų struktūra o tik po to kaip iš tos struktūros gauti reikiamą rezultatą. Naudodami žinių pernešimą mes sutrumpinam pirmąją mokymosi dalį ir reikalingą duomenų kiekį, kas išplečia panaudojimo sferą, nes užduotims, kurioms spręsti neturima tiek daug duomenų galima mėginti per naudoti jau mokytus modelius. Kitą problemą, kurią padeda spręsti žinių pernešimas yra mokymosi trukmė. DNT paremti modeliai turi gan ilgą mokymosi periodą, tas yra ypač būdinga SM tipo modeliams, su žinių pernešimu nebūtina turėti labai galingą kompiuterį, kad iš mokinti modelį spręsti kitą problemą. Tai yra labai naudinga metodika, nes GSM šios dvi problemos yra ne naujos ir sprendimų ne visada yra arba jie yra brangūs laiko ir/ar finansų atžvilgiu.

2. Gilus skatinamasis mokymasis

Šiame skyriuje apžvelgsime gilaus skatinamojo mokymo raidą, kas yra Markovo procesai bei kaip naudojantis jais yra formuluojama SM problema. Apsibrėžime Bellmano nelygybę ir kaip algoritmai sprendžia aplinkos tyrinėjimo bei išnaudojimo problemą. Susipažinsime su konvoliuciniais neuroniniais tinklais bei kaip jie yra taikomi skatinamajame mokyme. Ir pabaigai aptarsime kokios problemos kyla giliam skatinamajame mokyme bei kaip galime optimizuoti mokymą naudojantis žinių pernešimu.

2.1 Skatinamojo mokymo raida

Ankstyva SM istorija turi dvi pagrindines kryptis, kurios vystėsi nepriklausomai viena nuo kitos ir galiausiai susijungė moderniam skatinamajame mokyme. Pirma kryptis, remiasi gyvūnų ir žmogaus psichologija, kur mokymasis vyksta darant klaidas ir mokantis iš jų (angl. learning by trial and error). Ši kryptis yra viena ankščiausiai atsiradusių ir suteikė pagrindus moderniam SM. Pirmieji mėginimai naudoti tokius algoritmus prasidėjo 1933 metais, kai Thomass Ross, garsus elektronikas Vašingtono Universitete, sukonstravo elektroninę pelę, kuri galėjo rasti kelią paprastame labirinte ir jį atsiminti. Panašų eksperimentą pakartojo ir „Informacijos teorijos tėvas“ Claude Elwood Shannon, pademonstruodamas pelę, „Theseus“ gebančią spręsti labirintus, kur pats labirintas įsimindavo sprendimo kelią naudojantis magnetiniais jungikliais, kurie buvo po pele. Antra kryptis, kurios problema optimali aplinkos kontrolė, tai yra valdiklio konstravimas siekiant minimizuoti ar maksimizuoti sistemos dinamiškumą laikui bėgant. Tokias sistemas buvo bandoma spręsti naudojant būsenos vertės funkcijas ir dinaminį programavimą. Richard Bellman buvo vienas pirmųjų, kuris 1950 metais pasiūlė algoritmą, kaip spręsti tokias sistemas - maksimizuoti būsenos funkcijos reikšmę. Būsenos funkcija nusakė kokia vertė modeliui laiko momentu *t* būti būsenoje *s*. Dabar tai yra žinoma kaip Bellmano lygybė.

2.2 Markovo procesai ir mokymo problemos formulavimas

Markovo procesas, toliau MP, tai metodas formuluoti diskretaus laiko stochastinį procesą tenkinantį Markovo savybe. Šis procesas remiasi principu, jog praeitis nereikšminga numatant ateitį, svarbi tiktai esamojo laiko informacija. Remiantis MP, skatinamojo mokymo aplinką galime aprašyti aibių pora (S, P), kur S yra būsenų aibė o P, tikimybės pereiti iš būsenos $s_t \rightarrow s_{t+1}$.

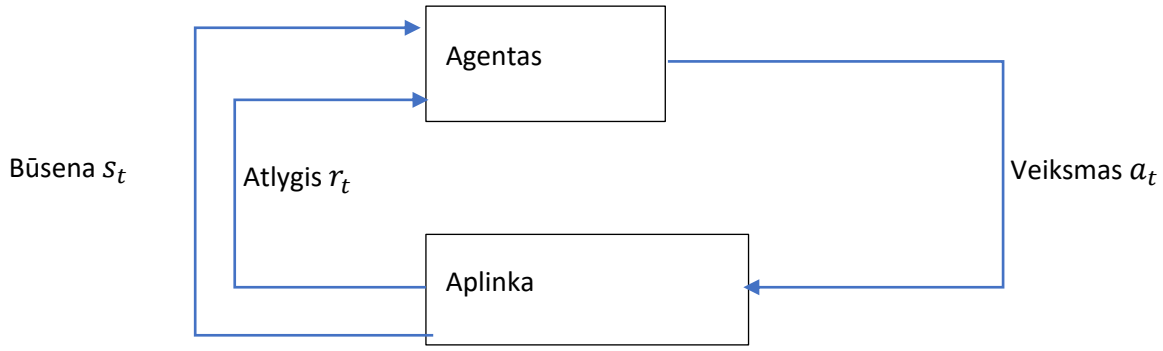
Markovo atlygio procesas[DRL], toliau MAP – tai yra Markovo procesas, kuriame perėjimas iš vienos būsenos į kita, turi tam tikrą papildomą skaitinę vertę ir yra apibrėžiamas aibe

(S, P, R). R aibe galime apibrėžti atlygio funkciją, kurią stengiasi maksimizuoti besimokantis agentas:

$$G_t = r_1 + r_2 + r_3 + \cdots + r_T \quad (1)$$

Kur $r_t \in R$, r_T – paskutinis atlygis, o G_t yra bendras sukauptas atlygis, už visą epizodą. SM mokyme ši formulė yra koreguoja ir pritaikoma MAP, kur būsenų aibė yra begalinė. Pavyzdžiui, jeigu mūsų agento tikslas išmokti vairuoti mašiną, tai galutinė būsena neapibrėžta $T = \infty$. Tokiu atveju ir atlygis gali būti $-\infty < G_t < \infty$. Kad to išvengti papildomai taikomas nuolaidos parametras $\gamma \in [0; 1]$. Jis nustato dabartinę vertę būsimų atlygių, kitaip tariant – atlygis už t žingsnių į priekį yra vertas γ^{t-1} kartų mažiau. Jei $\gamma = 0$, tai reiškia agentas fokusuosis į atlygį, kurį gauna šioje būsenoje, jei $\gamma = 1$ į atlygį, kurį gaus ateityje. Šią formulę galime išreikšti rekursyviai.

$$\begin{aligned} G_t &= \gamma^0 r_{t+1} + \gamma^1 r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots \\ &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots) \\ &= r_{t+1} + \gamma G_{t+1} = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned} \quad (2)$$



*Paveiksliukas 1. Šaltinis [*RLI*]. Skatinamojo mokymosi procesas*

Remiantis pirmu paveiksliuku galime formaliai užrašyti SM ir GSM problemą. Kiekviename laiko momente $t \in$ N, agentas yra būsenoje $s_t \in S$, kur S – galimų būsenų aibė duotoje aplinkoje, ir gali pasirinkti vieną arba kelis veiksmus $a_t \in A$, būdingus tai būsenai kur A – visų galimų veiksmų duotoje aplinkoje aibė. Pasirinkdamas veiksmą agentas už jį gauna atlygį r_{t+1}, tokia sąveika su aplinka generuoja MP seką: $s_0, a_0, r_1, s_1, a_1, r_2$, ... remiantis šia seka agentas siekia maksimizuoti G_t, kol pasieks ribą L. Riba L reiškia, kokį atlygį reikia agentui pasiekti, kad galėtume sakyti, jog jis išmoko duotoje aplinkoje spręsti jam pateiktą problemą.

Kai ši erdvė yra baigtinė - (S, A, R) aibės turi baigtinį elementų skaičių. Tada kintamieji s_t ir r_{t+1} turi tikimybę p, kuri nusako koks šansas į jas patekti iš esamos būsenos į kitą būseną pasirinkus veiksmą a_t

$$\sum_{s \in S} \sum_{r \in R} p(s_{t+1}, r_{t+1} | s_t, a_t) = 1 \tag{3}$$

Turint šią formulę, mes galime, bet kuriuo laiko momentu sužinoti:

- Perėjimo tikimybes iš duotos būsenos: $p(s_{t+1}|s_t, a_t) = \sum_{r \in R} p(s_{t+1}|s_t, a_t)$
- Atlygį pasirinkus veiksmą a_t, būsenoje s_t: $r(s_t, a_t) = \sum_{r \in R} p(s_{t+1}|s_t, a_t)$
- Tikėtinas atlygis būsenoje s_{t+1} pasirinkus veiksmą a_t, būsenoje s_t:

$$r(s_t, a_t, s_{t+1}) = \sum_{r \in R} r \frac{p(s_{t+1}, r_{t+1}|s_t, a_t)}{p(s_{t+1}|s_t, a_t)} \tag{4}$$

*Formulių šaltiniai: [*RLI*] atlygio ir perėjimo tarp būsenų tikimybių formulės*

2.3 Vertės funkcija ir Bellmano lygybė

Įvertinti tikėtiną atlygį, SM algoritmai naudoja vertės funkcijas. Vertės funkcija v_π, pasako, kokią vertę turi kiekviena būsena s ar būsenos ir veiksmo pora (s, a). Vertė šiuo atveju yra tikėtinas galutinis rezultatas, kurį gaus agentas sekdamas veiksmų pasirinkimo strategiją. Strategija formaliai užrašoma $\pi(a|s)$ ir nusako veiksmų pasirinkimo tikimybes būsenose. Vertės funkcijas galima apsibrėžti taip:

$$v_\pi(s) = E[G_t|s_t = s] = \mathrm{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid s_t = s\right] \tag{5}$$

$$q_\pi(s, a) = E[G_t|s_t = s, a_t = a] = \mathrm{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t = s, a_t = a\right] \tag{6}$$

Vertės funkciją, būsenos ir veiksmo porai vadinsime q_π, E[·] - žymi tikėtiną galutinį rezultatą žingsnyje t. Abi funkcijas galima įvertinti iš agento praeities veiksmų. Pavyzdžiui, jeigu agentas sekantis strategiją π išsaugo vertės vidurkį kiekvienai būsenai kurią aplanko, tai vykdant begalinį epizodų skaičių vidurkis konverguos į tos būsenos vertę. Svarbi savybė, kuri dažnai taikoma SM yra tai, kad vertės funkcijas galime išreikšti rekursyviai, vadinama Bellmano lygybe.

$$\begin{aligned} v_\pi(s) &= E[G_t|s_t = s] = E[R_t + \gamma G_{t+1}|s_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|, s, a)[r + \mathrm{E}[G_{t+1}|s_t = s'] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|, s, a)[r + v_\pi(s')] \end{aligned} \tag{7}$$

SM agento tikslas yra rasti strategiją π', su kuria vertės funkciją įgytų didžiausią reikšmę. Formaliai užrašant, laikysime, kad $\pi > \pi'$, tik tada ir tada, kai:

$$v_\pi(s) < v_{\pi'}(s)\ arba\ q_\pi(s,a) < q_{\pi'}(s,a)\ \forall s\ \in S, \forall a \in A$$

Tada Bellmano lygybė optimaliam sprendiniui rasti galime išreikšti per formulę:

$$\begin{aligned} v_*(s) &= \max_a q_\pi(s,a) = \max_a E[G_t|s_t = s, a_t = a] \\ &= \max_a E[R_{t+1} + \gamma G_{t+1}|s_t = s, a_t = a] \\ &= \max_a E[R_{t+1} + \gamma v_*(s_{t+1})|s_t = s, a_t = a] \\ &= \max_a \sum_{s',r} p(s', r|, s, a)[r + \gamma v_*(s')] \end{aligned} \tag{8}$$

Optimalią Bellmano lygybę $q_*(s,a)$ galime išsireikšti per $v_*(s)$:

$$\begin{aligned} q_*(s,a) &= \mathrm{E}[R_{t+k+1} + \gamma v_*(s_{t+1}, a')\ |s_t = s, a_t = a] \\ &= \sum_{s',r} p(s', r|, s, a)[r + \gamma \max_a q_*(s', a')] \end{aligned} \tag{9}$$

*Formulių šaltiniai: [*DRL*] Vertės funkcijos ir Dinaminis programavimas*

2.4 Aplinkos tyrinėjimo problema

Viena iš problemų, kuri iškyla tik skatinamajam mokyme yra aplinkos tyrinėjimo problema. Kad gauti didelį atlygį, SM agentas turėtų rinktis veiksmus kuriuos jis jau yra išbandęs ir žino, kad už juos gaus gerą atlygį. Bet kad agentas žinotų, kad renkasi tikrai gerus veiksmus prieš tai jis turi būti išbandęs daug veiksmų. Kitaip tariant, agentas turi išnaudoti jau turimas žinias, kad pasiektų gerą atlygį, bet taip pat turi ir tyrinėti aplinką tikėdamasis rasti geresnius veiksmus. Tai yra dilema, nes vienu metu abiejų dalykų vykdyti negalime, taip pat stochastinėse aplinkose įvertinti būseną ar būsenos ir veiksmo porą yra gan brangu, nes ją reikia pasirinkti daugelį kartų. Šią problemą spręsti yra taikomi keli metodai.

Epsilon godus metodas[RLI], (angl epsilon greedy method), kiekvienoje būsenoje agentas rinksis veiksmą pagal dabar jam žinomą optimalią strategiją arba veiksmą, kurio yra dar nebandęs pagal tikimybę $\varepsilon \in (0,1)$. SM tai yra gan svarbus parametras, nes pasirinkus per didelę reikšmę, agentas negalės pilnai sekti jokios strategijos ir visada tyrinės aplinką, pasirinkus per mažą reikšmę, agentas per mažai ištyrinės aplinką ir dėl to gali nerasti optimalios strategijos. Šią problemą yra patogu spręsti, mokymą pradedant su didele reikšme ε, kad agentas tyrinėtų aplinką

ir laikui bėgant ją mažint iki tam tikros ribos, tarkim $\varepsilon \sim 0.01$. Toks metodas leidžia agentui pradžioj daugiau ištyrinėti aplinką, o vėliau pradėti ją išnaudoti.

Kitas metodas, su kuriuo galime paskatinti agentą tyrinėti yra Lambda reguliarizavimo metodas, su kuriuo kiekviena strategijos $\pi(a|s)$ turi tokią formuluotę

$$\pi(a|s) = \frac{b}{c}, \qquad \pi_\lambda = \frac{b + \lambda}{c + \lambda * |A|}$$

Kur A – galimų veiksmų aibė. Šis metodas, neleidžia agentui, nustatyti 0 vertės tikimybę bet kokiam veiksmui, tai pat kuo didesnis λ – tuo labiau didelės tikimybės yra sumažinamos o mažos padidinamos. λ kaip ir ε – godus metodas, kuris pradžioje dažniausiai yra didelis, kad agentas būtų skatinamas tyrinėti aplinką ir laikui bėgant yra sumažinamas.

2.5 Konvoliuciniai neuroniniai tinklai

Standartiniai DNT architektūrai yra sunku tiesiogiai dirbti su vaizdo medžiaga. Ši problema kyla, nes vaizdo formatas yra $(n_h,\ n_w, 3)$, o į DNT tinklą pateikiama įvestis yra ištiesinto vaizdo formato $[n_h * n_w * 3]$, kur n_h, n_w – yra vaizdo įvesties aukštis ir plotis, o skaičius – 3 yra RGB spalvos kanalų skaičius. Net ir vidutinės kokybės vaizdas kuris yra $(640\ x\ 640)$ pikselių dydžio gali sukelti mokymosi problemų, nes pats pirmas sluoksnis turės 1,228,800 parametrus. Jei ir kiti sekantys sluoksniai turės panašų skaičių parametrų, mokymosi procesas gali būti labai ilgas.

Konvoliucinai neuroniniai tinklai[ICD], toliau KNT, yra gilaus mokymosi struktūra, kuri gali priimti vaizdinę informaciją ir priskirti svarbą skirtingoms vietoms vaizde. Šis procesas yra vykdomas naudojantis konvoliucijos sluoksniu, kuris yra sudarytas iš filtrų rinkinio. Filtrai yra praleidžiami per kiekvieną vaizdo spalvos kanalą. Pats filtras yra dažniausiai (f, f), dydžio parametrų gardelė, kuri bando išgauti požymių žemėlapį (angl. feature map), pagal kurį galėtų identifikuoti vaizdą. Taip pat šio proceso metu vaizdo formatas irgi yra sumažinamas priklausomai nuo filtro dydžio. Sekantis konvoliucinis sluoksnis veikia tokiu pat principu. Kai vaizdas tampa pakankamai mažas, jis yra ištiesinamas ir tada įvedamas į standartinę DNT architektūrą, kad išgauti atsakymą.

2.6 Gilus skatinamasis mokymas

Skatinamasis mokymas vadinamas giliu[DRL], kai agento veiksmų pasirinkimo strategiją diktuoja dirbtiniai neuroniniai tinklai ar būsenos vertės funkcija yra įvertinama naudojantis jais. Prie kiekvienos funkcijos prisideda DNT svoriai θ, formaliai visas iki nagrinėtas funkcijas žymėsime: $v_*(s;\theta)$, $q_*(s,a;\theta)$ $\pi(a|s;\theta)$. Kadangi svoriai tampa atsakingi už agento veiksmų

pasirinkimus, tai pats mokymo procesas yra jų optimizavimas. Gilus skatinamojo mokymosi modeliai, skirtingai nei paprasti skatinamojo mokymosi modeliai, gali mokytis ir iš vaizdo įvesties. Naudojantis konvoliuciniais neuroniniais tinklais galima mėginti iš mokyti agentą atlikti užduotį aplinkoje jam nežinant aplinkos dinamikos.

2.7 Problematika

Pastaruoju metu tokio tipo mokymasis tapo labai populiarus ir aktualus mašinų mokyme, nes buvo pasiekti dideli proveržiai. Vienas iš jų yra gebėjimas žaisti kompiuterinį žaidimą „Dota 2“. „OpenAI“ komandai pavyko iš mokyti agentą, kuris gebėtų laimėti prieš profesionalių žaidėjų komandą[OAB]. „Dota 2“ yra komandinis „MOBA“ tipo žaidimas, kur dvi komandos varžosi tarpusavyje, kuri greičiau užkariaus priešingos komandos bazę. Tai yra sudėtingas uždavinys GSM, dėl šių priežasčių:

- Ilga žaidimo juosta – žaidimo trukmė yra apie 45 minutes, 30 kadrų per sekundę. Daugelis veiksmų, kurie yra pasirenkami žaidime neturi didelės įtakos, pavyzdžiui ėjimas į priekį su herojų, o kai kurie turi didelę įtaką, kaip priešininko komandos nario nugalėjimas. Taip pat veiksmai atliekami pirmosiomis žaidimo minutėmis gali turėti didelę įtaką likusiam žaidimui.
- Dalinė aplinkos informacija – agentas mato aplink save tam tikru atstumu, didžioji dalis aplinkos jam yra nežinoma, kad gerai žaisti šį žaidimą reikia daryti spėjimus ir prielaidas, kur yra priešininko komanda.
- Didelė būsenų, veiksmų aibė – kiekvienu laiko momentu agentas turi pasirinkimą atlikti labai daug veiksmų, todėl įvertinti būsenos ar būsenos ir veiksmo vertę yra gana sunku.

Kad agentas galėtų įveikti profesionalius žaidėjus, jis turėjo sužaisti labai daug žaidimų, 180 metų trukmės žaidimų per dieną, du mėnesius naudojant superkompiuterį. Toks eksperimentas turi labai didelius kaštus ir juos gali atlikti tik labai didelį kapitalą turinčios įmonės.

2.8 Žinių pernešimas.

Žinių pernešimas – kai mašinų mokymo modelis iš mokytas spręsti vieną problemą yra naudojamas kaip pradinis taškas mokytis spręsti kitą problemą. Formaliai tai yra vieno DNT modelio svorio parametrų θ naudojimas kitame modelyje. Tai dažnai naudojama metodika prižiūrimajame mokyme, kompiuterinės regos (angl. computer vision) ir natūralios kalbos apdorojimo (angl. natural language processing) užduotyse. Tai yra daroma, nes pirmieji neuroniniai tinklo sluoksniai gerai gali atpažinti ir rasti vaizdo savybes o vėlesni sluoksniai iš mažų

detalių atpažinti objektą. Toks sprendimas taikomas, kai turime labai daug duomenų spręsti vienai problemai, o kitai per mažai. Tokiu būdu taip pat galime sutrumpinti mokymosi laiką. Šią metodika ir mėginsime pritaikyti giliam skatinamajam mokymui.

3. Gilaus skatinamojo mokymo optimizavimas

Šiame skyriuje atliksime eksperimentus. Mėginsime optimizuoti gilaus skatinamojo mokymo procesą taikant žinių pernešimą. Tai darysime mokindami agentus žaisti žaidimus ir lyginsime juos tokiu principu:

1. Iš mokinsime agentą $A1$ žaisti žaidimą $M1$
2. Iš mokinsime agentą $A2$ žaisti žadimą $M2$
3. Apkeisime agentų $(A1, A2)$ žaidimus, tai reiškia agentas $A1$ mėgins išmokti žaisti $M2$, o agentas $A2$ žaisti $M1$. Ir palyginsime jų mokymosi procesą. Ar agentai greičiau išmoko žaisti atitinkamą žaidimą kai pradėjo žaisti nieko nežinodami ar prieš tai išmokę žaisti kitą žaidimą.

Testus atliksime, aplinkose kur agentas turi mokytis iš vaizdinės aplinkos ir kur yra pateikiama aplinkos informacija skaitine verte S. Taip pat išmėginsime keletą skirtingų algoritmų ir palyginsime atskirai kokią įtaką turi žinių pernešimas kiekvienam iš jų, bei padarysime bendrą išvadą kaip pavyko optimizuoti GSM naudojantis žinių pernešimą.

3.1 Karkasai kurti dirbinius neuoroninių tinklų modelius

Konstruojant DNT modelius naudosime karkasus, kurie palengvina kūrimo procesą. Naudosime Python 3.7 kalbos versiją ir Keras[INK], PyTorch[TRC] karkasus. Keras karkasas pateikia aukšto lygio DNT modeliavimo abstrakcijas, kurios remiasi PyTorch ar Tensorflow karkasų pagrindais ir dažniau yra naudojamas smulkesnėms užduotims bei prototipų testavimui. O PyTorch yra platesnį funkcionalumą pateikiantis karkasas, kuriuo jau atliekami ilgesnis ir sudėtingesni tyrimai. Abu karkasai yra atviro kodo, kurių tikslas pateikti abstrakcijas per kurias tyrinėtojas gali greitai ir nesunkiai apsibrėžti DNT modelį, jį treniruoti bei testuoti. Karkasai skirti darbui su daugiamačiais duomenimis ir yra pritaikyti atlikti skaičiavimus naudojantis vaizdo plokšte. Tą daro pasinaudodami NVIDIA kompanijos kurta biblioteka cuDNN. Kadangi daugelis DNT modelio skaičiavimų yra gerai iš lygiagretinami, tai skaičiavimų perkėlimas į vaizdo plokštę labai paspartina mokymosi procesą.

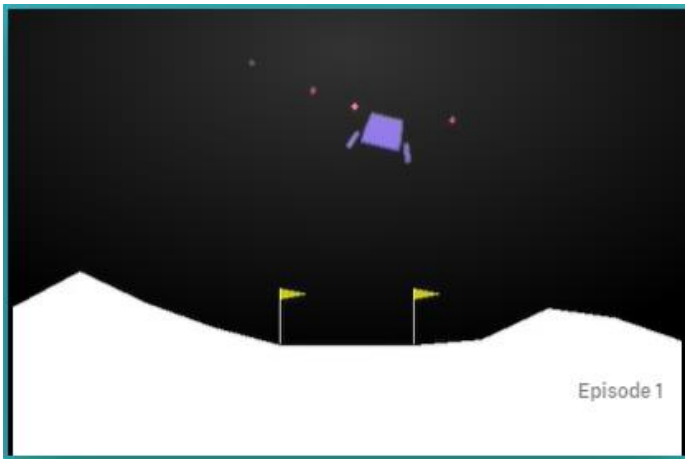
3.2 Mokymosi aplinkos „OpenAI Gym“ ir „Arcade learning environment“

„OpenAI“ – Elono Muso ir Samo Altmano įkurta ne pelno siekianti kompanija, su tikslu globaliai vystyti dirbtinį intelektą. „OpenAI Gym“[OAG] yra šios įmonės atviro kodo sukurtas įrankis, kurio dėka programuotojai gali testuoti skatinamojo mokymo algoritmus, treniruodami

agentus žaisti įvarius žaidimus arba atlikti užduotis. Šis įrankis pateikia jau suformuluotas (S, A, R) aibes ir parametrą parametrą, d kuris nurodo ar baigėme žaisti epizodą.

„Arcade learning environment“[ALE], toliau ALE – karkasas skirtas palengvinti skatinamojo mokymo agentų treniravimą, žaisti „Atari 2600“ žaidimus. „Atari 2600“ yra 1977 metų sukurta žaidimų konsolė, kuriai buvo sukurta daugiau nei 500 populiarių žaidimų, kaip „Space invaders“, „Breakout“ ir kiti. ALE yra sukurtas naudojantis „Stella“ emuliatoriumi, kuris suformuluoja abstrakcijas su kuriomis dabartiniai kompiuteriai gali perduoti įvestį ir gauti žaidimo atvaizdį. Kiekvieno žaidimo dydis yra 210 pikselių dydžio ir 160 pikselių ilgio bei turi 128 skirtingų spalvų paletę. Mūsų patogumui „OpenAI Gym“ turi integraciją su ALE, tai formaliai naudosimės tik viena agentų mokymo aplinka.

3.2.1 „LunarLander-v2“



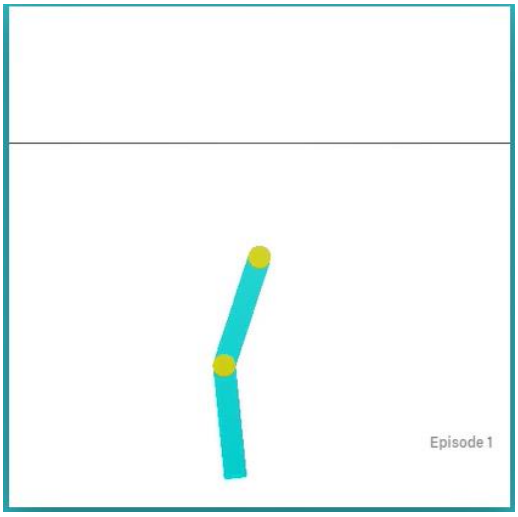
Paveiksliukas 2. Šaltinis [OAIG]. „LunarLander-v2“ aplinkos vaizdas.

Agento tikslas yra saugiai nuleisti ant Mėnulio erdvėlaivį. Tai jis turi padaryti naudodamas 3 įrengtus variklius su kuriais gali valdyti erdvėlaivio poziciją. (S, A, R) aibės.

- S: $[x, y, v_x, v_y, \theta, v_\theta, l, r]$
  - x, y – erdvėlaivio koordinatės
  - v_x, v_y – erdvėlaivio greitis x ašimi, y ašimi.
  - θ, v_θ – erdvėlaivio pasvirimo kampas ir kampinis greitis
  - l, r – erdvėlaivio kairės ir dešinės kojos sąlytis su žeme
- A: [a_0, a_1, a_2, a_3]
  - a_0 – nieko nedaryti
  - a_1, a_2, a_3 – naudoti kairinį, pagrindinį ar dešinį variklį.

- R – Judant arčiau link nusileidimo zonos v_x, v_y arti nulio nusileidimo greičiu suteikiama nuo 100 iki 140 taškų, žaidimo pabaigoje už sėkmingą nusileidimą suteikiama 100 bei 10 taškų už kiekvieną koją kuri turi sąlytį su žeme. Laikoma, kad agentas išsprendė šią užduotį kai $G_T \geq 200$.

3.2.2 „Acrobot-v1“



Paveiksliukas 3. Šaltinis [OAIG]. „Acrobot-v1“ aplinkos vaizdas.

„Acrobot-v1“ – dviejų dalių svyruoklė, kur galima judinti tik vidurinį sraigtą. Agento tikslas yra iškelti antrą jungiamąja dalį iki tam tikro aukščio ribą. (S, A, R) aibės.

- S: $[\cos(\theta_1), \sin(\theta_1), \cos(\theta_2), \sin(\theta_2), v_1, v_2]$
  - $\cos(\theta_1), \sin(\theta_1)$ – pirmosios svyruoklės dalies kampo išraiška pradinėje būsenoje reikšmės $[1,0]$
  - $\cos(\theta_2), \sin(\theta_2)$ – antrosios svyruoklės dalies kampo su pirmąja dalimi išraiška, pradinėje būsenoje reikšmės $[1,0]$
  - v_1, v_2 – kampiniai greičiai pirmosios ir antrosios svyruoklės dalies
- A: $[a_0, a_1, a_2]$ – suteikti jėgos momentą į sraigtą. $[-1,0, +1]$
- R – už kiekvieną veiksmą duodama -1 atlygis, išskyrus kai antroji dalis pasiekia tam tikro aukščio ribą, už kurią gaunamas 0 atlygis. Laikoma, kad agentas išsprendė šią užduotį, kai epizodo atlygis yra $-95 \leq G_T$

3.2.3 „Space invaders-v4“

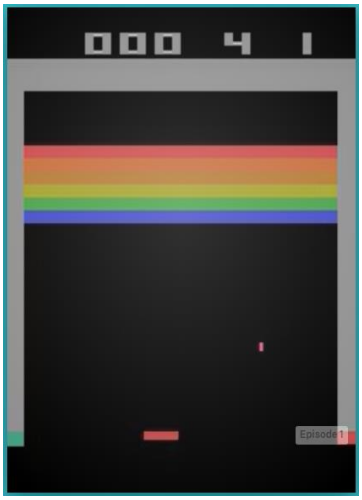


Paveiksliukas 4. Šaltinis [OAIG]. „Space invaders-v0“ aplinkos vaizdas.

Agento tikslas yra įveikti visus ateivius. Tą padaryti agentas turi tris bandymus. Mirus daugiau nei 3 kart žaidimas pradedamas iš naujo. (S, A, R) aibės.

- $S -$ (210,160,3) RGB formato vaizdas.
- A: [$a_0, a_1, a_2, a_3, a_4, a_5$]
    - a_0 – nieko nedaryti
    - a_1 – šauti
    - a_2, a_3 judėti į dešinę ar kairę
    - a_4, a_5 judėti į dešinę ar kairę ir kartu šauti
- R – Už kiekvieną nušautą ateivį gaunamas atlygis, už pirmos eilės $+5$, antros eilės $+10$, trečios $+15$ ir t.t, o už paskutinės eilės , $+30$.

3.2.4 „Breakout-v4“



Paveiksliukas 5. Šaltinis [OAIG]. „Breakout-v0“ aplinkos vaizdas.

Šis žaidimas dar žinomas kaip „bricks“, kur žaidėjo ir šiuo atveju agento tikslas yra sunaikinti visas plyteles, tą padaryti jis turi 5 bandymus.

- $S-$ $(210,160,3)$ RGB formato vaizdas.
- A: $[a_0, a_1, a_2, a_3]$
  - a_0 – nieko nedaryti
  - a_1 – šauti („OpenAI Gym“ aplinkos klaida (angl. bug))
  - $a_2, a_3,$ – judėti į dešinę arba kairę
- R – už kiekvieną sunaikintą plytelę $+1$

Treniruojant agentus žaisti „Atari 2600“ žaidimus transformuosime gaunamą vaizdą. Naudosime kadrų praleidimo strategiją[DFD]. Abi aplinkos generuoja 60 kadrų per sekundę. Mūsų agentas žiūrės tik į kas ketvirtą kadrą, tai yra 20 kadrų per sekundę, taip pat mokymo procese įvestis bus 4 paskutiniai kadrai. Taip yra daroma, nes žiūrint į kas ketvirtą kadrą nėra prarandama daug informacijos ir mokymo proceso laikas sutrumpėja ženkliai, taipogi paduodami 4 paskutinius kadrus suteikiame agentui galimybę išsiaiškinti į kurią pusę judą objektai. Veiksmų atlikimo principas dėl tokios strategijos irgi truputį keičiasi, pasirinktą veiksmą mes kartosime 4 kadrus iš eilės. Taip pat pakeisime dydį kiekvieno kadro, pritaikysime pilką filtrą vietoj RGB spalvų ir apkarpysime vaizdo dalis, kurios neturi svarbios informacijos, pavyzdžiui kaip viršui esanti rezultato skaičiuoklė ir sumažinsime paveikslėlį iki $(84,84)$ arba $(80,80)$ pikselių dydžio.

Taip pat toliau eksperimentus su „OpenAI Gym“ aplinkomis vadinsime EKSGYM, o eksperimentus su ALE – EKSALE.

3.3 Q-mokymasis

Q-mokymosi algoritmas, toliau QM, kuris renkasi veiksmą a_k būsenoje s_t, kuris suteikia didžiausią žinomą atlygį. Tai yra algoritmas, kuriam nebūtina žinoti aplinkos dinamikos modelio. QM priskiria būsenom reprezentacines skaitines vertes, kurios nusako kokia yra būsenos vertė ir mokymosi procese stengiasi rinktis būsenas kurios turi didžiausią vertę. Būsenoms priskiriamos vertės tokiu yra sudedama dabartinės būsenos vertė su kitos būsenos verte, kuri yra didžiausia iš visų galimų į kurias agentas gali patekti, taikant nuolaida. Formaliai tai galime užrašyti taip:

$$Q(s_t, a) = (1-\alpha) * Q(s_t, a) + \alpha(r_t + \gamma * \max_a Q(s_{t+1}, a)) \quad (10)$$

$\alpha \in (0,1)$ yra mokymosi greitis, o $\gamma \in (0,1)$ nuolaidos parametras. Tai yra standartinė formulė, bet kadangi taikysime DNT mokymosi procesui, šiuo atveju $Q(s_t, a)$ atrodys taip:

$$y_i = \mathrm{E}_s[r_i + \gamma * \max_a Q(s_t, a; \theta_{i-1})] \quad (11)$$

$$L_i(\theta_i) = \mathrm{E}_{s,a}[\left(y_i - Q(s,a;\theta_i)\right)^2] \quad (12)$$

Kur θ_i yra DNT svoriai, y_i – kitos būsenos tikėtinas atlygio įvertis. Kad sumažinti atlygių pasiskirstymą, y_i parametrus rinksimės iš „atminties“ masyvo M. Kiekvieną agento žingsnį įsiminsime tokiu principu (s, a, r, s', d), kur d – reiškia ar baigėme žaidimą, ir kai norėsime atnaujinti DNT svorį, atsitiktinai pasirinksime 64 žingsnius iš M ir atnaujinsime DNT tinklą remiantis formule $L_i(\theta_i)$. Naudosime atskiras DNT architektūras ir parametrus EKSGYM-1 ir EKSALE-1 eksperimentams atlikti. Turėdami šią informaciją galime apsirašyti pseudo-kodą ir architektūras:

1. Inicializuojame DNT tinklo svorius θ su atsitiktinėmis reikšmėmis
2. Inicializuojame „atminties“ masyvą M
3. Pradedame žaisti epizodą:
   3.1. Pasirenkame vieną iš galimų veiksmų a_t būsenoje s_t, pagal DNT svorius θ_i, arba pagal tikimybę ε atsitiktinį veiksmą a_t
   3.2. Žengiami žingsnį aplinkoje ir gauname r_{t+1}, s_{t+1}, d būseną
   3.3. Įsimename šį žingsnį (s, a, r, s', d) į masyvą M
   3.4. Pasirenkame 64 atsitiktinius žingsnius iš atminties ir treniruojame DNT svorius θ_i
   3.5. Jei d nurodo, kad epizodas nebaigtas nustatome s', kaip dabartinę būseną s ir grįžtame į 2.a punktą
4. Patikrinam darbo baigimo sąlygas – ar jau sužaidėm N epizodų. Jei sąlyga yra patenkinta baigiama darbą, jei ne grįžtame į antrą punktą.

Formulių šaltiniai: [PDR] Skyrius 2 biografijos faktai

***EKSGYM* neuroninių tinklų architektūra**	
Sluoksnis	**Parametrai**
Įvesties	Aplinkos būsenos aibės dydis $\|S\|$
Pilnai sujungtas	512 neuronų
Pilnai sujungtas	128 neuronų
Pilnai sujungtas	64 neuronų
Išvesties sluoksnis	Veiksmų būsenos aibės dydis $\|A\|$

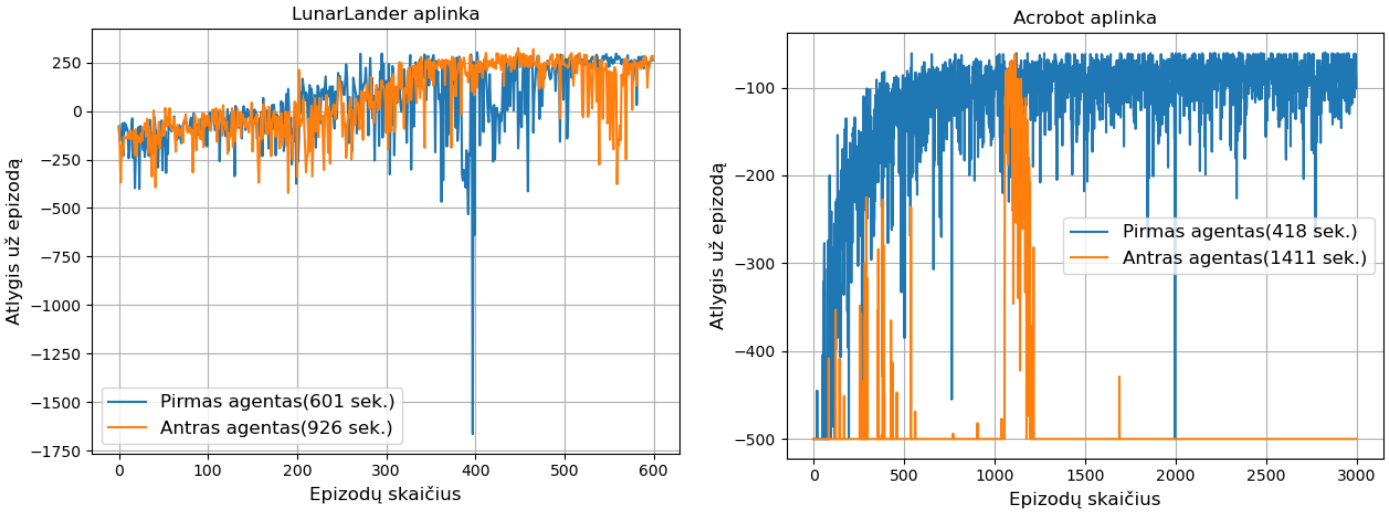
Parametrai *EKSGYM* aplinkoms	
Parametras	**Reikšmė**
Mokymosi greitis	0,01
Nuolaida	0,99
Epsilon pradinis	1
Epsilon galutinis	0,1
Atminties dydis (masyvo dydis)	250.000

Lentelė 1. EKSGYM-1, Elementari pilnai sujungtų neuroninių tinklų architektūra ir parametrai

Pirmas agentas – A1 žaidžia M1, o Antras agentas – A2 jau išmokęs žaisti M2, žaidžia M1. Iš „LunarLander“ aplinkos matome gan gerus rezultatus. Atlygių svyravimas už epizodą yra bendrai mažesnis antro agento, tai pat jeigu stabdytume mokymo procesą ties 400 epizodu Antras

agentas jau būtų pilnai galėjęs įveikti šią užduotį. Antro agento mokymosi trukmė bendrai yra trečdaliu ilgesnė, bet tai yra todėl, nes agentas išmokęs saugiau nuleisti erdvėlaivį ilgiau tai daro, kas ir bendrai sudarė ilgesnį mokymosi laiką.

Iš „Acrobot“ aplinkos matome priešingą rezultatą. Naujai mokinamas pirmas agentas ženkliai geriau sprendžia šios aplinkos užduotį. Tuo tarpu antras agentas sunkiai sugeba gauti didesnį atlygį nei -500, kas reiškia, kad jam mokytis nepavyksta. Mokymo trukmė yra apie 3.5 karto ilgesnė, bet tai yra todėl, nes pirmas agentas išmokęs spręsti užduotį ją greitai išsprendžia ir baigia epizodą, kol antras agentas tą daro visą epizodą. Taip pat galime padaryti išvadą, kad pirmas agentas žaisdamas šį žaidimą blogai išmoko apibendrinti įvestis (problema vadinama angl. „overfitting“)



Paveiksliukas 6. EKSGYM-1 eksperimento rezultatai

Dabar pamėginsime tokį pat eksperimentą atlikti ir su ALE aplinkomis. Nauja DNT architektūra ir parametrai atrodys taip:

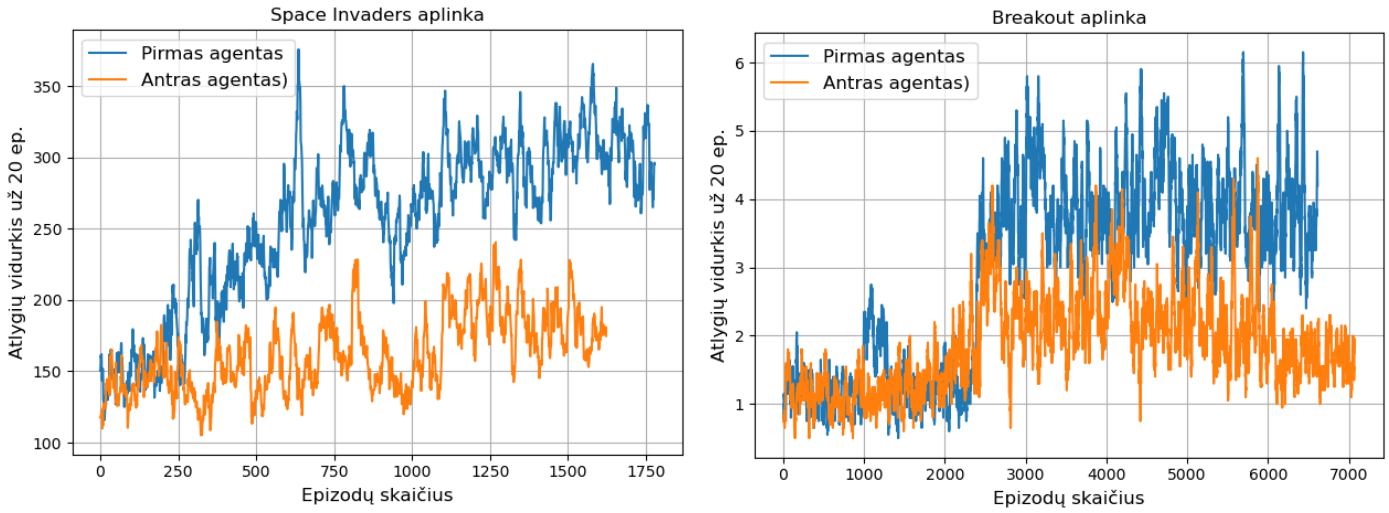
***EKSGYM-1* neuroninių tinklų architektūra**	
Sluoksnis	**Parametrai**
Įvesties	28.224 pikselių reikšmės
Konvoliucinis	32 filtrai dydžio - (8,8)
Konvoliucinis	64 filtrai dydžio - (4,4)
Konvoliucinis	64 filtrai dydžio – (3,3)
Konvoliucinis	512 neuronų
Išvesties sluoksnis	Veiksmų būsenos aibės dydis $\|A\|$

Parametrai *EKSGYM-1* aplinkoms	
Parametras	**Reikšmė**
Mokymosi greitis	0,0025
Nuolaida	0,9
Epsilon pradinis	1
Epsilon galutinis	0,01
Atminties dydis (masyvo dydis)	1.000.000

Lentelė 2. EKSALE-1 Konvoliucinė neuroninių tinklų architektūra ir parametrai

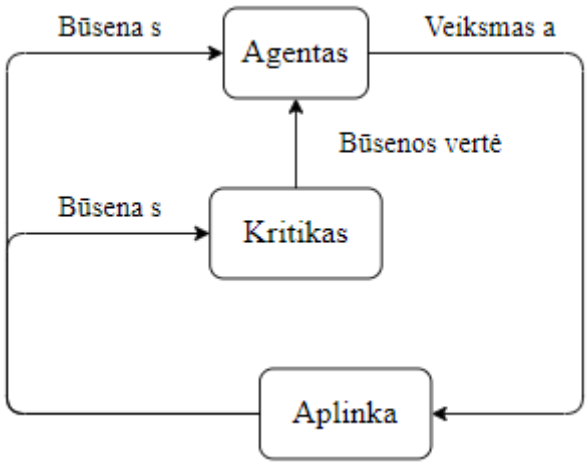
Antro eksperimento metu taip pat truputį pakoreguosime Q-mokymosi algoritmą. Turėsime papildomą kintamąjį, vadinkime jį taikiniu, kuriame saugosime agento modelio kopiją. Kurią atnaujinsime su agento svoriais θ, kas N žingsnių. Šis metodas turėtų sumažinti atlygių pasiskirstymą, nes N žingsnių, mes turėsime taikinį, kuris nesikeičia, taip pat grafuose atlygius atvaizduosime paskutinių 20 epizodų vidurkiu. Kadangi mokymasis gali trukti labai ilgai, apsiribosime laikotarpį – 10 valandų vienam agentui ant kažkurios aplinkos.

Atlikę eksperimentą matome kad „Space Invaders“ aplinkoje agentas ženkliai geriau mokosi nuo pradžių, nei su žinių pernešimu. Tokia pat situaciją galima įžvelgti ir „Breakout“ aplinkoje. Atlygiai atrodo gan triukšmingai abiejų aplinkų atžvilgu. Galime daryti prielaidą, kad žinių pernešimas Q-mokyme sudėtingesnėse ALE aplinkose nėra geras metodas optimizuoti procesą, nes per tą patį laikotarpį agentas nuo nulio išmoko geriau spręsti duotą užduotį aplinkoje.



Paveiksliukas 7. EKSALE-1 eksperimento rezultatai

3.4 Aktoriaus ir kritiko mokymasis



Paveiksliukas 7. A2K mokymosi schema

Aktoriaus ir kritiko mokymasis, toliau A2K, susideda iš dviejų algoritmų. Aktoriaus, kuris pagal strategiją $\pi_\theta(a|s)$ renkasi veiksmus ir maksimizuoja atlygį ir kritiko kuris vertina būsenas kurias renkasi agentas.

Aktoriaus algoritmas šiuo atveju bus REINFORCE[MLG], kuris remiasi Monte Carlo metodais ir stengiasi maksimizuoti veiksmo a pasirinkimo tikimybę būsenoje s, kad galutinis atlygis G_T būtų kuo didesnis. Tą daro maksimizuodamas šio gradiento reikšmę:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \tag{13}$$

θ – yra DNT svoriai. REINFORCE algoritmas sužais vieną epizodą pagal turimą strategiją ir pagal ją atnaujins svorius, tai pseudo-algoritmą galime užrašyti taip:

1. Sužaidžiam epizodą naudodamiesi turimą strategiją.
2. Išsaugome strategijos pasirinktus veiksmus būsenas ir atlygius.
3. Atnaujiname svorius
4. Patikriname ar mokymo sąlygos patenkintos, jei taip baigiam darbą, jei ne grįžtame į pirmą punktą.

Sprendžiant sudėtingesnes problemas, kur būsenų aibė yra didelė, REINFORCE algoritmas susiduria su didele tikimybių variacija ir galutiniai epizodų atlygiai būna labai triukšmingi. Taip pat jeigu $G_t = 0$, mūsų agentas neįsimins dalies veiksmų. Šiai problemai spręsti mes galim pasitelkti kritiką kuris nustatys bazinę funkcijos vertę. Aktoriaus ir kritiko metodai[PPO] gali naudoti skirtingus algoritmus veiksmų pasirinkimo strategijai optimizuoti ir būsenai įvertinti variantus. Šiuo atveju naudosime pranašumo aktoriaus ir kritiko variantą[18] (angl. advantage actor-critic). Pranašumo funkciją apibrėšime taip:

$$A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t) \tag{14}$$

Kur A nurodys koks yra pranašumas rinktis konkretų veiksmą a_t, būsenoje s_t, Q_w – būsenos ir vertės funkcija, V_v – bendra būsenos vertė imant visų galimų veiksmų teikiamų atlygių vidurkį. Indeksas v, reiškia, kad būsena įvertinama dirbtinio neuroninio tinklo. Naudojantis Bellmano lygybe, galime šią formulę apsirašyti taip:

$$\begin{aligned} Q(s_t, a_t) &= E[r_{t+1} + \gamma V(s_{t+1})] \\ A(s_t, a_t) &= E[r_{t+1} + \gamma V(s_{t+1})] - V(s_t) \end{aligned} \tag{15}$$

Taigi turėdami tokį atlygio įvertį galime apsibrėžti funkciją kurią maksimizuosime:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t) \tag{16}$$

A – įvertinti galima naudoti tuos pačius dirbtinio neuroninio tinklo parametrus arba turėti atskirą modelį, kuris tai darys. Šiame eksperimente turėsim vieną bendrą tinklą su kuriuo atliksime svorio atnaujinimo operacijas iš aktoriaus ir kritiko pusės. Taigi apsibrėžia mokomąja funkciją galime užrašyti A2K algoritmo psedo-kodą bei architektūrą su parametrais.

1. Inicializuojame DNT tinklo svorius θ su atsitiktinėmis reikšmėmis
2. Sužaidžiam epizodą pagal $\pi_\theta(a_t|s_t)$ veiksmų pasirinkimo strategiją ir išsisaugome:
   2.1. Būsenas į kurias patekome
   2.2. $\pi_\theta(a_t|s_t)$ reikšmes ir kokį veiksmą pasirinkome būsenoje s_t
   2.3. Gautus atlygius už kiekvieną žingsnį
3. Iš kritiko gauname $V_\theta(s_t)$ ir $V_\theta(s_{t+1})$ būsenų vertes.
4. Pagal $\nabla_\theta J(\theta)$ atnaujiname DNT tinklo svorius
5. Patikrinam darbo baigimo sąlygas – ar jau sužaidėm N epizodų. Jei sąlyga yra patenkinta baigiama darbą, jei ne grįžtame į antrą punktą.

Formulių šaltiniai: [MLG]REINFORCE skaidrė, [AMD] – Asinchroninis gilaus skatinamojo mokymo karkasas, [PPL]

***EKSGYM* neuroninių tinklų architektūra**	
Sluoksnis	**Parametrai**
Įvesties	Aplinkos būsenos aibės dydis $\|S\|$
Pilnai sujungtas	64 neuronų
Pilnai sujungtas	32 neuronų
Pilnai sujungtas	16 neuronų
Išvesties (Kritiko)	1 neuronas
Išvesties (Aktoriaus)	Veiksmų būsenos aibės dydis $\|A\|$

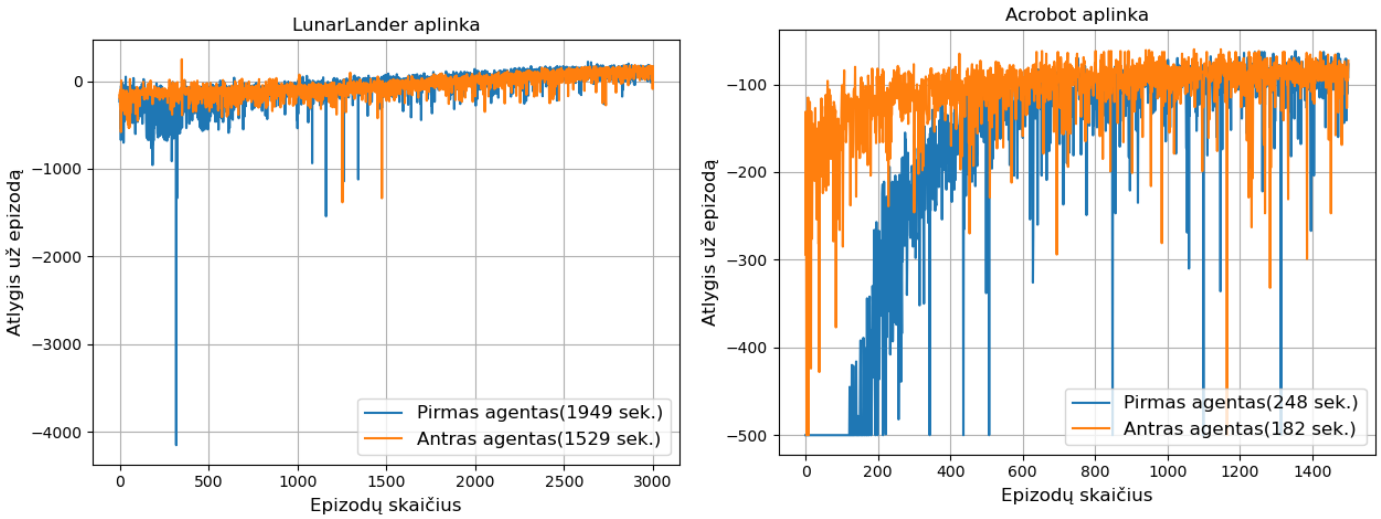
Parametrai *EKSGYM* aplinkoms	
Parametras	**Reikšmė**
Mokymosi greitis (Kritiko)	0,0005
Mokymosi greitis (Aktoriaus)	0,0001
Nuolaida	0,99

Lentelė 3. EKSALE-2, Elementari pilnai sujungtų neuroninių tinklų architektūra ir parametrai

Iš „LunarLander“ aplinkos matome panašų rezultatą kaip ir Q-mokymosi algoritme. Pirmo agento atlygio pasiskirstymas už epizodą yra didesnis, taip pat svyravimas irgi truputį didesnis. Tik šiuo atveju antras agentas greičiau įveikė užduotį. Bendras epizodų skaičius yra keletą kart didesnis nei Q-mokymosi, tai yra todėl, nes A2k mokosi ne epizodo metu, bet sužaidęs epizodą, taip pat eksperimento atveju, reikėjo praleisti daugiau laiko ieškant optimalių arba bent jau pakankamai gerų DNT architektūrų, nes pats algoritmas yra labai jautrus bet kokiems pasikeitimams.

Iš „Acrobot“ aplinkos matome skirtingą rezultatą nei Q-mokymosi algoritme. Pirmam agentui prireikė daugiau laiko kol jis pradėjo mokytis aplinkoje ir ją išmoko, tuo tarpu antras agentas ganėtinai greit priartėjo prie optimalios strategijos ir mokymosi procesą galima buvo stabdyti daug ankščiau.

Panašu, kad A2K algoritmai geba geriau apibendrinti užduotį ir jų apmokintas dirbtinis neuroninis tinklas yra universalesnis bei galimai taikomas kitoms užduotims.



Paveiksliukas 8. EKSGYM-2 eksperimento rezultatai

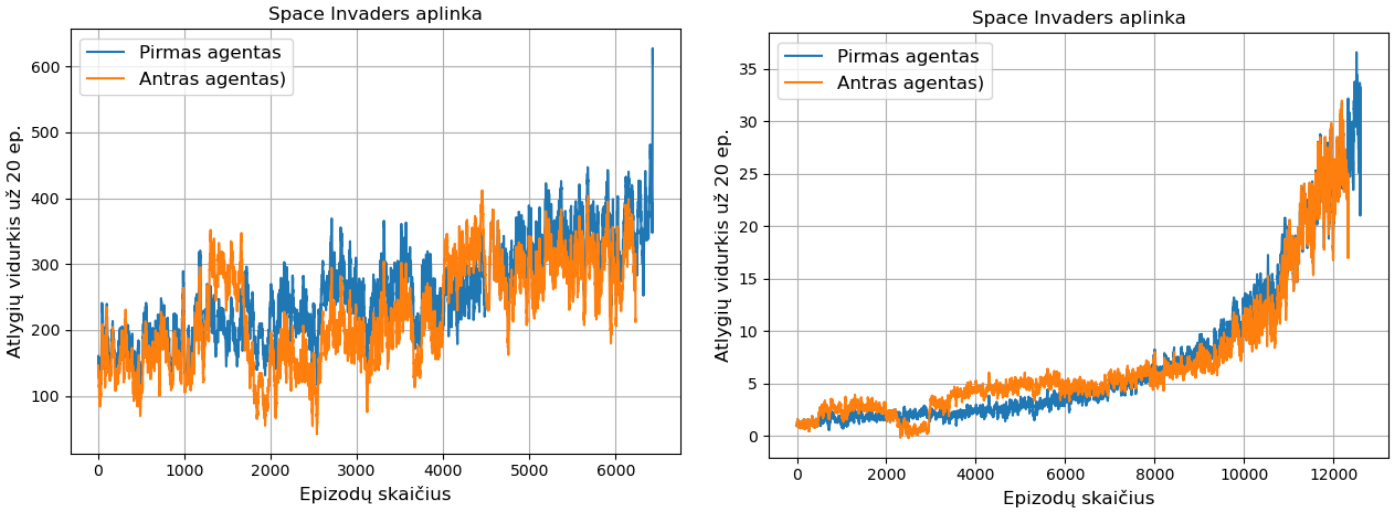
Antru eksperimentu truputį pakoreguosime A2K algoritmą, kad paspartinti mokymosi procesą. Kadangi mokymasis vyksta ne epizodo metu, bet sužaidus epizodą, mes galime naudoti asinchroninę šio algoritmo versiją, kurią toliau vadinsime A3K. Pagrindinis skirtumas nuo A2K yra, kad vienu metu žaidžia *k* aktorių. Tas leidžia greičiau ištyrinėti aplinką ir bendrai pagreitinti mokymo procesą.

***EKSGYM-1* neuroninių tinklų architektūra**	
Sluoksnis	**Parametrai**
Įvesties	25.600 pikselių reikšmės
Konvoliucinis	32 filtrai dydžio - (3,3)
Konvoliucinis	32 filtrai dydžio - (3,3)
Konvoliucinis	32 filtrai dydžio – (3,3)
Konvoliucinis	512 neuronų
Išvesties (Kritiko)	1 neuronas
Išvesties (Aktoriaus)	Veiksmų būsenos aibės dydis $\|A\|$

Parametrai *EKSGYM-1* aplinkoms	
Parametras	**Reikšmė**
Mokymosi greitis (Kritiko ir aktoriaus)	0,001
Nuolaida	0,99
Epsilon pradinis	1
Epsilon galutinis	0,01
Atminties dydis (masyvo dydis)	1.000.000

Lentelė 4. EKSALE-2 Konvoliucinė neuroninių tinklų architektūra ir parametrai

Matome panašų rezultatą kaip ir Q-mokyme, agentas, kuris vykdo žinių pernešimą mokosi truputį prasčiau arba labai panašiai į agentą, kuris mokosi nuo pradžių. Tą pastebime tiek „Space Invaders“ tiek „Acrobot“ aplinkose. Taip pat atlygių pasiskirstymas irgi yra gan mažesnis, ypač „Breakout“ aplinkoje tiek pas pirmą agentą, tiek pas antrą agentą. Pranašumo aktoriaus kritiko metodai panašu, kad geriau sugeba apibendrinti aplinkoje problemą, nes šių eksperimentų metu žinių pernešimas arba turėjo teigiamą įtaką arba neturėjo beveik jokios.



Paveiksliukas 9. EKSALE-2 eksperimento rezultatai

3.5 Eksperimentų išvados

Matome, kad optimizuoti skatinamąjį mokymą naudojantis žinių pernešimu ne visada įmanoma. Eksperimentų rezultatai rodo, kad žinių pernešimas paspartino agento mokymąsi tik „LunarLander“ aplinkoje iš 4 eksperimentų. Čia naudojant skirtingus algoritmus matome, kad atlygių pasiskirstymas tapo mažesnis naudojant Q-mokymąsi ir pranašumo aktoriaus kritiko algoritmus. Taip pat mokymą galėjome stabdyti anksčiau ir būtume optimizavę mokymą ir laiko atžvilgiu. „Acrobat“ aplinka rodo priešingus rezultatus, agentas kuris pirma išmoko žaisti „LunarLander“ nesugeba per duotą epizodų skaičių išmokti žaisti „Acrobot“ problemą. Tas galimai kyla dėl atlygių skirtumo, nes „LunarLander“ aplinkoje keletą būsenų yra labai naudingos, tokios kaip lėtas nusileidimas. Kad saugiai ir lėtai nusileisti reikia atlikti daugiau veiksmų. Agentas mėgindamas sekdamas šią strategiją „Acrobot“ aplinkoje gauna didelį neigiamą atlygį nes renkasi įvykdyti daugiau veiksmų. Tuo tarpu agentas mokintas „Acrobot“ aplinkoje sėkmingai tvarkosi ir „LunarLander“ aplinkoje. EKSGYM-1 eksperimente matome, kad Q-mokymosi procese žinių pernešimas tik sulėtina mokymąsi, nes iš mokinti agentai per tą patį laiko tarpą pasiekia blogesnį

rezultatą, nei nuo pradžių mokomas agentas. Tas aiškiau matoma „Space Invaders“ aplinkoje. Tuo tarpu A3K algoritmas nepagerina situacijos, bet galima ir nepablogina jos. Abu agentai per tą patį laiko tarpą pasiekia panašius rezultatus. Taip pat pats mokymo procesas A3K yra ženkliai geresnis, nei QM, nes per tą patį laiką buvo sužaista beveik dvigubai daugiau epizodų ir pasiekti geresni rezultatai.

Iš eksperimentų galima daryti prielaidą, kad skatinamojo mokymo algoritmai yra linkę į persimokymą (angl overfitting), tai reiškia, kad jie labai gerai gali spręsti tik tą problemą ant kurios jie mokėsi ir bet kokie nauji duomenys nebus teisingai generalizuojami. QM tai yra ypač būdinga, jei atlygių pasiskirstymas yra didelis. Tuo tarpu A3K tas neturi įtakos ir agentas tiesiog iš naujo ieško naujos optimalios strategijos. Visa tai pastebime mokydami agentus spręsti EKSGYM aplinkos užduotis, tai nutinka, nes tinklą treniruojamas spręsti konkrečią užduotį daug ilgiau ir dėl to galima pertreniruoja svorius, kad gauti kuo optimalesnį galutinį atlygį.

Išvados

Darbo pagrindinis tikslas buvo optimizuoti dirbtinius neuroninius tinklus taikant žinių pernešimą. Po atliktų eksperimentų galime daryti išvadą, kad optimizuoti nepavyko. Nepaisant to, kad kai kuriose paprastesnėse aplinkose optimizavimo procesas pavyko, bendru atveju vis tiek optimizacija nepavyko. Tokia išvadą pateikiame dėl to nes pagrindinė problema, kurią mes norėjome išspręsti optimizuojant gilų skatinamąjį mokymąsi naudojantis žinių pernešimu yra mokymosi laiko sutrumpinimas. Kadangi paprastose aplinkose mokymas trunka trumpai bet kuriuo atveju, optimizavimas jose nėra labai naudingas, nes jis sutrumpina vidutiniškai $\frac{1}{3}$ mokymosi laiką. Tai yra iš ~ 30 minučių į ~ 20. O sudėtingesnėse aplinkose, kur pasiekti geresnį rezultatą reikia treniruoti ilgą laiko tarpą ~ 10 valandų mes gavome arba tokius pat rezultatus, kaip mokant agentą nuo pradžių arba net pablogėjusį mokymo procesą.

Kitas svarbus dalykas yra skaičiavimo galios (angl.computing power) turėjimas, mėginti atlikti eksperimentus su procesorium yra bloga idėją, ypač su Q-mokymosi algoritmu, buvo pastebėta, kad tą darant ant vaizdo plokštės pagreitėjimas yra apie 5 kartus. Tas mažiau pastebima naudojant A3K, bet bendru atveju atlikti EKSGYM tipo eksperimentus, be tinkamos aparatūros sukeltų galimai daug keblumų, nes prieš pradedant eksperimentą reikia nemažai laiko optimizuoti hiper-parametrus, kuriuos įvertinti reikia treniruoti agentą, bent pora valandų.

Aplinkos dinamika turi daug įtakos mokymosi procese. Kadangi agentas yra skatinamas tyrinėti aplinką pradžioje jis susirenka daug atsitiktinių būsenų ir yra galimybė, kad esamas rinkinys agentą pasuka į ne optimalios strategijos pusę. Pavyzdžiui „SpaceInvaders“ aplinkoje agentas kartais išmokdavo tiesiog judėti į šoną ir šaudyti net nemėgindamas slėptis nuo priešininko šūvių. Tokiu būdu jis gan greit surinkdavo nemažai taškų, bet taip pat ir greit pralaimėdavo žaidimą. Išmokus tokią strategiją agentui nesisekė toliau mokytis optimalesnės strategijos. Tokių strategijų radimas gali turėti įtakos ir žinių pernešimui bei optimizavimo procesui, nes tokio tipo strategija yra tikėtinai tinkama tik tai konkrečiai aplinkai.

Darbui atlikti buvo pasirinkti du skirtingi mokymosi principai. Tas turėjo daugiau įtakos nei tikėtasi, pirmas principas – kur agentas bando įvertinti būsenos ir veiksmo porą skaitine verte ir pagal tai rinktis kiekvieno būsenoje veiksmą, su kuriuo ši vertė būtų didžiausia - Q-mokymais. Antras principas, bandyti rasti veiksmų pasirinkimo strategiją, kuria vadovaujantis galutinis epizodo rezultatas būtų didžiausias. Pasirodo agentui lengviau apibendrinti problemą, kai jis ją sprendžia bandydamas išmokti tikimybių skirstinį, nei būsenos ir veiksmo vertes. Ta pastebime iš

eksperimentų. Aktoriaus kritiko metoduose žinių pernešimas blogiausiu atveju nepadarė jokios įtakos mokymosi procesui, o kitais atvejais jį paspartino, priešingai nei Q-mokymasis.

Galiausiai apžvelgiant literatūrą ir bendrus pasiekimus šioje mokslo šakoje pastebėta, kad nauji algoritmai, kaip „MuZero“ inkorporuoja abu mokymosi principus, kas galimai leistų geriau pritaikyti žinių pernešimą. Taip pat DNT architektūrų principas keičiasi, vietoj pilnai sujungtų sluoksnių naudojami ilgalaikės-trumpalaikės atminties sluoksniai (angl. Long-Short Term Memory, LSTM). Tokie pakitimai irgi galimai pateiktų kitokius rezultatus, jei mėgintume taikyti žinių pernešimą. Bet šiuo metu galime teikti, kad pernešti žinias iš vieno skatinamojo mokymo agento į kitą siekiant optimizuoti mokymosi procesą yra nerekomenduotina.

Šaltiniai

- [INK] A. Gulli, S. Pal „Deep Learning with Keras“ Implement neural Networks with Keras on Theano and Tensorflow
- [ICD] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep
- [DFD] A. S. Lakshminarayanan, S. Sharma, B. Ravindran „Dynamic Frame skip Deep Q Network“
- [OAG] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba „OpenAI Gym“
- [DRL] Y. Li, Deep Reinforcement Learning
- [PPO] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov „Proximal Policy Optimization Algorithms“
- [ALE] M. G. Bellemare, Y. Naddaf, J. Veness, M. Bowling „The Arcade LEarning Environment: An Evaluation Platform for General Agents“
- [DSI] N. Desai, A. Banerjee „Deep Reinforcement Learning to play Space Invaders
- [TRC] R. Collobert, K. Kavukcuoglu and C. Farabet, "Torch7: matlab-like environment for machine learning,"
- [RLI] R. S. Sutton and A. G. Barto, Reinforcement Learning An Introduction second edition
- [MAA] S. Gredanus „Mastering Atari with an Actor-Critic model“
- [MLG] T. Wang, Machine Learning Group, University of Toronto „Learning Reinforcement Learning by Learning REINFORCE“
- [PDR] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller „Playing Atari with Deep Reinforcement Learning“
- [AMD] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, K. Kavukcuoglu „Asynchronous Methods for Deep Reinforcement Learning“
- [PPL] Coursera kursai Pratical Reinforcement Learning
- [OAB] OpenAI 5 blog - https://openai.com/blog/openai-five/
- [UOG] Understanding OpenAI Gym https://medium.com/@ashish_fagna/understanding-openai-gym-25c79c06eccb
- [LIN] Linkedin - https://learning.linkedin.com/blog/top-skills/the-skills-companies-need-most-in-2020and-how-to-learn-them
- [COD] – Kodo repositorija https://github.com/Martis467/Code