Vilniaus Gedimino Technikos Universitetas
Fundamentinių mokslų fakultetas
Matematinės statistikos katedra

Justina Marcinkėnaitė

STATISTINĖ INKSTŲ VEIKLOS SUTRIKIMO ANALIZĖ

The statistical analysis of renal dysfunction

Baigiamasis bakalauro darbas

Taikomosios statistikos ir ekonometrijos studijų programa, valstybinis kodas 612G31001
Statistikos studijų kryptis

Vilnius, 2022

Vilniaus Gedimino Technikos Universitetas
Fundamentinių mokslų fakultetas
Matematinės statistikos katedra

TVIRTINU
Katedros vedėjas

_____
(Parašas)

_____
(Vardas, pavardė)

_____
(Data)

Justina Marcinkėnaitė

STATISTINĖ INKSTŲ VEIKLOS SUTRIKIMO ANALIZĖ

The statistical analysis of renal dysfunction

Baigiamasis bakalauro darbas

Taikomosios statistikos ir ekonometrijos studijų programa, valstybinis kodas 612G31001
Statistikos studijų kryptis

Vadovas _____ _____ _____
(Pedagoginis vardas, vardas, pavardė) (Parašas) (Data)

Vilnius, 2022

Turinys

Iliustracijų sąrašas

Lentelių sąrašas

Įvadas

Šių laikų žaidimų industrijoje sukurti žaidimo agentą, kuris realistiškai reaguoja į aplinką, yra sudėtinga užduotis reikalaujanti daug laiko bei pastangų. Taip pat neretai prie laiko kaštų prisideda ir tai, kad prieš kuriant žaidimo agentą tam reikia tinkamai paruošti aplinką – sukurti pagalbinius įrankius leidžiančius supaprastinti programavimo procesą. Tačiau net įdedant pastangas programuojant agentus, rezultatai yra dažnai prasti, jie nėra pakankamai protingi, žaidėjui nėra didžiulio iššūkio juos įveikti. Tai priveda prie didžiausio žaidėjų nepasitenkinimo – agentu sukčiavimo. Norint agentus padaryti protingesniais jiems suteikiama žaidimo informacija, kuri yra neprieinama įprastiems žaidėjams.

Per pastarąjį dešimtmetį įvykus neuroninių tinklų proveržiui buvo dedama daug vilčių, kad jie sugebės sukurti protingesnius žaidimo agentus. Tačiau neuroniai tinklai tėra funkcijos aproksimacijos metodas ir sunkiausia šio išukio dalis greitai atsiskleidė - kaip sukurti funkciją, kuri padėtų agentams tobulėti, tapti protingesniams? Šis klausimas buvo nagrinėtas per pastaruosius 70 metų, įtraukiant tokias mokslo šakas kaip dinaminis programavimas, markovo modeliai, neuromokslai. Atradimai minėtose šakose privedė prie skatinamojo mokslo gimimo, kuomet 1987 metais šio mokslo pradininkai Sutton, R. bei Barto, A. moksliniame straipsnyje [1] pirmieji sukūrė optimizavimo funkciją, pavadinta **?**Temporal difference**?** (toliau TD), padedančia agentams mokytis iš praeities rezultatų, siekiant pasirinkti tokius ateities veiksmus, kurie pagerintų ateities rezultatą. Funkcijos principas remiasi skatinimo metodais, kaip siunčiant paskatinimo impulsus agentui, jei jis atliko teisinga veiksmą, bei nubaudžiamas jei atliko neteisingą veiksmą. Tokiu principu agentas sukuria pozityvias bei negatyvias asociacijas su galimais veiksmais, kuriuos jis gali atlikti. Šį principą puikiai iliustruoja pavlovo šunų eksperimentas. Sekančius dešimtmečius sekė šios funkcijos gerinimas, optimizavimas, bandymas išspręsti didelės dimensijos problemą - kuo didesnė dimensija, tuo ilgiau užtrunka skaičiavimai. Ši problema buvo išspresta pasitelkiant neuroninius tinklus, kurie aproksimuodavo TD funkciją. Šio metodo kulminacija įvyko 2016 metais, kuomet Google AlphaGo sugebėjo įveikti stalo žaidimą Go, kurio nesugebėjo įveikti nei viena komanda prieš tai dirbusi prie pirmųjų algoritmų, įveikusių šachmatų čempionus. Šiuo metu skaitinamieji metodai aprėpia daugybe sričių: automobilių autonominis vairavimas, logistikos grandžių optimizavimas, robotikos judėjimo problemos ir dar daugybe kitų sričių.

Šiame darbe bus išanalizuojamas skatinamasis mokymasis bei įgyvendintas vienas iš šio mokslo metodų - TD3. Teorinėje dalyje bus analizuojama šio metodo veikimo principai. Tai atlikus praktinėje dalyje šis metodas bus pritaikytas išspręsti roboto voro fizinio judėjimo aplinkoje problemą.

Darbo aktualumas. Dirbtiniui intelektui sugebėjus įveikti šachmatų čempionus buvo manoma, kad žmonių dominavimas tradiciniuose stalo žaidimuose pasibaigė. Tačiau programuotojų ambicijos išblėso, kuomet min-max paremti šachmatų algoritmai nesugebėjo įveikti

kiniečių tradicinio stalo žaidimo – Go. To pagrindinė priežastis – neapskaičiuojamai didelė žaidimo būsenų erdvė. Kompiuteriai užtrukdavo per ilgai planuoti ėjimus ir jam neužtekdavo atminties išanalizuoti visus galimus ėjimus. Todėl iki šiol šis stalo žaidimas buvo neįveikiamas iki kol Google korporacija išleido skatinamojo mokymosi paremto algoritmo, kuris 2016 metais sugebėjo įveikti šių laikų geriausią Go žaidėją. Nuo to laiko skatinamųjų mokymosi metodų aktualumas vis labiau plito įtraukiant sritis kaip autonominiai automobiliai, robotika, finansai bei vis naujai populiarėjanti sritis – žaidimų industrija.

Darbo tikslas Pritaikyti TD3 skaitinamojo mokymosi modelį, kuris apmokytų simuliuojamą vorą vaikščioti 3D aplinkoje.

Uždaviniai. Maecenas eget erat in sapien mattis porttitor:

1. Išanalizuoti gilių neuroninių tinklų struktūrą.
2. Įgyvendinti Python bei Unity duomenų sinchronizaciją.
3. Išanalizuoti skatinamojo mokymosi modelio principus.
4. Išanalizuoti Belmano, Q-mokymosi, TD funkcijas.
5. Sukurti unity voro aplinką bei agentą, kuris bus apmokomas vaikščioti.
6. Įgyvendinti TD3 mokymosi modelį bei apmokyti vorą vaikščioti sukurtoje aplinkoje.

1. Teorinė dalis

Skatinamojo mokymo metodai remiasi neuroninių tinklų aproksimacijom. Todėl šioje darbo dalyje nagrinėsiu giliųjų neuroninių tinklų veikimo principus. Tai atlikęs toliau nagrinėsiu skatinamojo mokymosi metodus bei kaip neuroniniai tinklai padeda juos įgyvendinti. Pagrindinis nagrinėjamas skatinamojo mokymosi modelis bus TD3. Tai yra modelis kuris padeda agentui tolydžioje aplinkoje pasirinkti veiksmus, kurie suteike dydžiausią apdovanojimą. Galiausiai bus nagrinėjama aplinka, kurioje šis mokymo metodas bus įgyvendintas.

1.1. Neuroninių tinklų klasifikatorių tipai

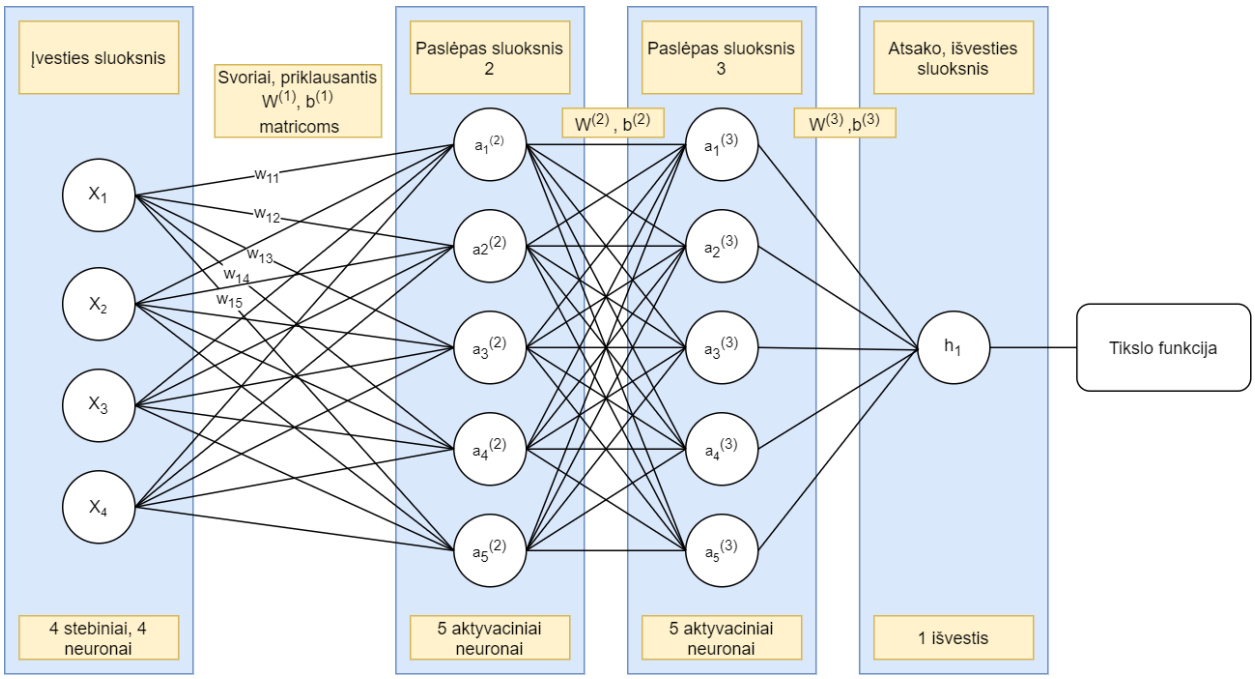
1.1.1. Neuroninio tinklo struktūra

Neuroninių tinklų sandarą galimą žiūrėti kaip į daugybės logistinių funkcijų veikimą. Kitaip tariant vieną neuroną galima traktuoti kaip logistinę funkciją, kuri išsiskaido į dvi dalis: tiesinę bei aktyvacinę. Tiesinėje dalyje suskaičiuojame tiesinę lygtį ir ją įvedame į aktyvacijos funkciją, kuri logistinėje regresijoje yra sigmoido funkcija. Tai yra identiška logistinei regresija, kurios išvedimus buvau pateikęs praeitoje ataskaitoje. Aktyvacinės funkcijos gali būti skirtingos, nebūtinai sigmoido. Pagrindinė skirtingų aktyvacinių funkcijų priežastis yra spartesnė svorių konvergacija į optimalią reikšmę. Dažnai naudojama Relu aktyvacinė funkcija, kuri neturi sigmoido nykstančio gradiento problemos. Ši problema pasireiškia kuomet sigmoido funkcijos reikšmė yra labai didelė arba maža, tuomet gradientas tampa beveik nulinis ir mokymosi procesas sustoja. Pats neuroninis tinklas susideda iš logistinių funkcijų (jei naudojame ne sigmoid aktyvacinę funkciją tai jau nebėra logistinė funkcija) sluoksnių, kurie prasideda įvesties sluoksnių, viduryje paslėptuoju sluoksniu ir galiausiai išvesties sluoksniu. Visi klasifikatoriai turi vieną bendrą panašumą – įvesties bei paslėptuosius sluoksnius. Tačiau priklausomai nuo to ar atsakas yra klasifikacinis ar regresinis, kinta išvesties sluoksnis. Kuriant python aplinkoje neuroninius tinklus buvo pasitelkta vektorizacija. Visus įvestis, svorius, išvestis išsireiškus matriciniu pavidalu gaunamas ryškus neuroninių tinklų paspartėjimas. Neuroninio tinklo bendra naudota struktūra yra pateikta 1 pavekslėlyje.

Vieno sluoksnio įvesties transformacijos vektorizaciją yra taip skaičiuojama:

$$Z^l = W^{l^T} * A^{l-1} + b^l$$

Žymėjimas: A^l yra l sluoksnio įvestis (matrica, kuri susideda iš a^l elementų) gauta po aktyvacinės funkcijos (jei l yra lygus įvesties sluoksniui, tai A^l tampa duomenų įvesties vektorius X), o W^T yra to sluoksnio svoriai. Šios transformacijos matricos eilutės yra skirtingų neuronų tiesinės transformacijos reikšmės, o stulpeliai žymi skirtingus mokymosi duomenis. Šią transformaciją įkėlus į aktyvacijos funkciją gauname galutinę perceptrono reikšmę. Norint, kad neuroninis tinklas būtų tikslus, mes turime optimizuoti svorius. Tam pasitelkiame



1 pav. Gilaus neuroninio tinklo bendra struktūra

gradientinį nuolydį. Gradientinis nuolydis padeda surasti svorius su kuriais tikslo funkcija įgyja mažiausią paklaidą. Tikslo funkcija sudaro tolygų paviršių, kurio skiringos ašys yra svorių reikšmės bei viena iš jų yra tikslo funkcijos rezultatas. Pateiktame 2 pavekslėlyje apatines ašis x bei y galima traktuoti kaip svorius, o viršutinę ašį kaip tikslo funkcijos paklaidą.

Judandami svorius priešingą gradiento puse, mes judame ties tokiais svoriais, kurie mažiną tikslo funkcijos paklaidą. Šią trajektoriją atspindi 2 pavekslėlyje esanti raudona tiesė, kuri kiekvienos iteracijos metu (mėlyni taškai) po truputi slenka žemyn, optimizuoja svorius.

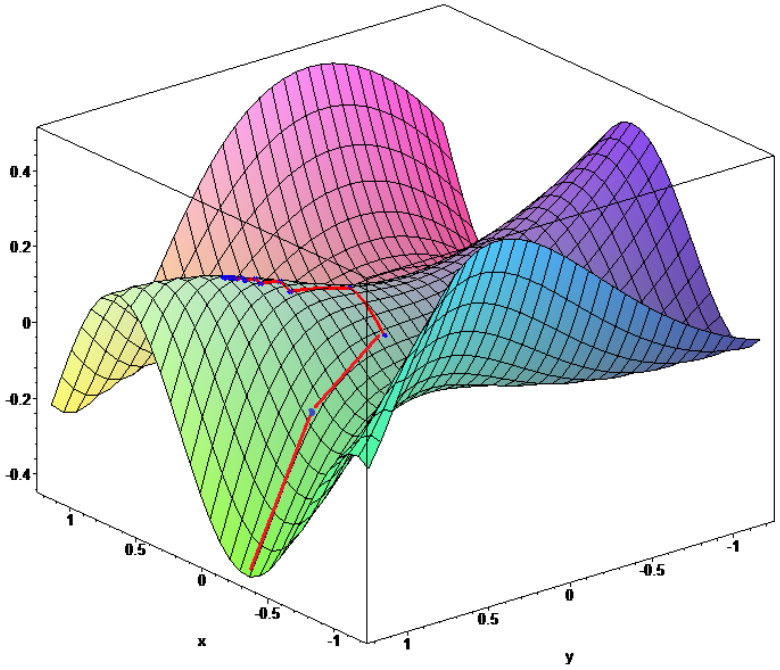
Vėlgi gradientinio nuolydžio vektorizacija yra tokia pati tiek įvesties, tiek paslėptuose sluoksniuose, bet priklausomai nuo analizuojamos problemos - skirtinga išvesties sluoksnyje. Gradiento vektorizacija:

$$dZ^l = W^{l+1^T} dZ^{l+1} * g^l\left(Z^l\right) \tag{1}$$

Žymėjimas: $g^l(Z^l)$ yra dabartinio sluoksnio aktyvacinės funkcijos išvestinė. Ji yra sudauginama su visais matricos nariais. Gavus perceptronų išvestines dZ^l toliau galime gauti svorių gradientus:

$$\begin{aligned} dW^l &= \frac{1}{m} dZ^l A^{(l-1)^T} \\ db^l &= \frac{1}{m}\left(dZ^l \text{ matricos stulpeliu susumavimas }\right) \end{aligned} \tag{2}$$

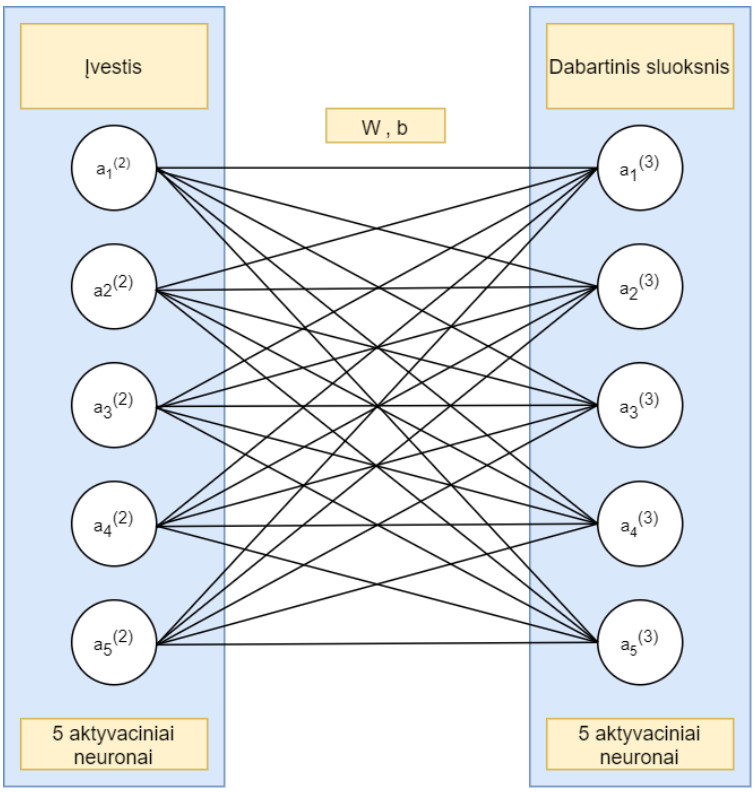
Žymėjimas: stulpelių skaičius arba mokymosi duomenų kiekis yra žymimas m. Toliau bus aptariamos priekinio, atgalinė propogacija bei tikslo funkcijos, kurios sudaromos atitinkamai pagal turima klasifikacijos problemą.



2 pav. Gradientinio nuolydžio pavyzdys

1.1.2. Neuroninio tinklo priekinė propogacija

Įgyvendinti gilaus neuroninio tinklo struktūra pasirinktoje programavimo kalboje nėra sunku, kuomet pastebimas visiems sluoksniams bendras pasikartojantis skaičiavimo bruožas. Skaičiuojant priekinę propogaciją (judame neuriniu tinklu iš kairės į dešinė) į neuroninį tinklą galima žiurėti kaip į atskirus blokus susidedančius iš įvesties bei išvesties, kaip tai matosi 3 paveikslėlyje.

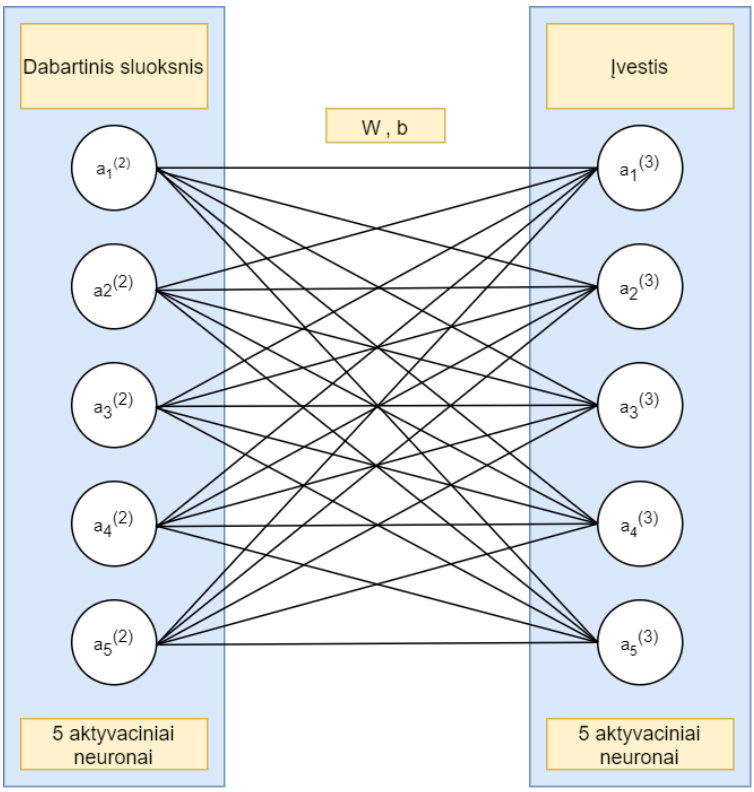


3 pav. Priekinės propogacijos vieno neuroninio sluoksnio apskaičiavimo grafas

Tokia formą išskaidžius neuroninį tinklą, jį tampa labai paprasta apskaičiuoti, tereikia iteruoti kiekvienu sluoksniu iš kairės į dešinę. Kiekvienos iteracijos metu praeitą sluoksnį traktuojame kaip įvesties sluoksnį ir naudojantis 2 formulėmis apskaičiuojame dabartinio sluoksnio aktyvacinius neuronus. Šios operacijos metu svarbu išsaugoti tarpinius skaičiavimus, kaip tiesinės funkcijos bei aktyvacinę funkcijos apskaičiavimo rezultatą. Jos bus reikalingos apskaičiuoti gradientinį nuolydį, kaip tai matosi 1 bei 2 formulėse. Paskutinės iteracijos metu, atlikus aktyvacinės funkcijos apskaičiavima, lieka tik apskaičiuoti tikslo funkcijos rezultatą. Šį struktūra pasižymi formulių universalumu, jos tinka neuroniniams tinklams, kurie yra negilieji bei gilieji, kitaip tariant - nepriklauso nuo sluoksnių kieko. Atlikus priekinę propgaciją ir suskaičiavus tikslo funkcijos klaidą, galime pradėti ją mažintį, judėdami priešinga gradiento linkme. Tai atlieka sekanti potemė - atgalinė propogacija.

1.1.3. Neuroninio tinklo atgalinė propogacija

Atgalinės propogacijos struktūra yra vos ne identiška priekiniai propogacijai. Pagrindiniai skirtumai yra sukeistos vietos apskaičiuojamo sluoksnio bei įvesties sluoksnio. Tačiau kaip ir priekinės propogacijos atveju, mes atliekame panašias įteracijas, tik šį kart pradedam nuo tiklso funkcijos ir judame link pirmo sluoksnio, iš kairės į dešinę. Atliekant šias svorių atnaujinimo operacijas, kaip tai matosi 1 bei 2 formulėse, pasitelkiame priekinės propogacijos metu išsaugotais duomenimis.



4 pav. Atgalinės propogacijos vieno neuroninio sluoksnio apskaičiavimo grafas

1.1.4. Binarinis klasifikatorius

Tai klasifikatorius, kuris vos ne identiškas logistinei regresijai. Klasifikuoja binarinius atsakus. Išvesties sluoksnis turi tik vieną neuroną bei jo reikšmės transformacija sutampa su logistinės regresijos tikslo funkcija:

$$\log(L) = \ell = \sum_{i=1}^{n} \left[y_i \log\left(p_i\right) + \left(1 - y_i\right) \log\left(1 - p_i\right) \right] \tag{3}$$

Šios funkcijos neurono gradientas yra susiprastiną į šią formą:

$$dZ^l = A^l - Y, \tag{4}$$

kur l yra paskutinis sluoksnis, Y yra tikroji atsako reikšmės, o A^l yra neuroninio tinklo gauti atsakai. Turint dZ^l galima gauti svorių gradientus bei neuronų gradientus, kaip tai buvo parodyta prieš tai. Turint svorių gradientus atnaujiname svorius:

$$W^l = W^l - \alpha dW^l \tag{5}$$

kur α nurodo gradientinio žingsnio dydį.

1.1.5. Daugiau nei viena klasė

Klasifikuojant daugiau nei vieną klasę yra naudojamas entropijos tikslo funkcija. Tikslo funkcijos forma:

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i * \log(\hat{y}_i), \tag{6}$$

kur y yra tikroji reikšmė, o $\hat{y}$ yra aproksimuota.

Kuomet paskutinio sluoksnio aktyvacinė funkcija yra „soft-max“, šios funkcijos gradientas labai gražiai susiprastina ir paskutinio sluoksnio gradientas atrodo ši taip:

$$dZ^l = (Y - \hat{Y}) \tag{7}$$

kur l yra paskutinio sluoksnio indeksas, o dydžiosios Y yra vektorizacija daugybės stebinių į vieną matricą.

Iš šio gradiento galime seniau aptartais metodais gauti W bei b gradiento reikšmes. Galima naudoti ir didžiausių kvadratų metoda, tačiau gradientinis nuolydis konverguotų lėčiau. Taip pat pagrindinis skirtumas tarp binarinio klasifikatoriaus yra aktyvacinė funkcija. Šį kart tai nėra sigmoido funkcija, o „soft-max“ normalizacija. Jos metu kiekvieno neurono reikšmės apskaičiuojamos naudojantis eksponento funkcija e^z, visos reikšmės padalinamos iš šių reikšmių sumos ir gautas rezultatas perduodamos į tikslo funkciją. Taigi šis metodas yra dažnai naudojamas klasifikuoti reikšmes, kurios turi daugiau nei vieną klasę.

1.1.6. Tiesinės regresijos atsakas

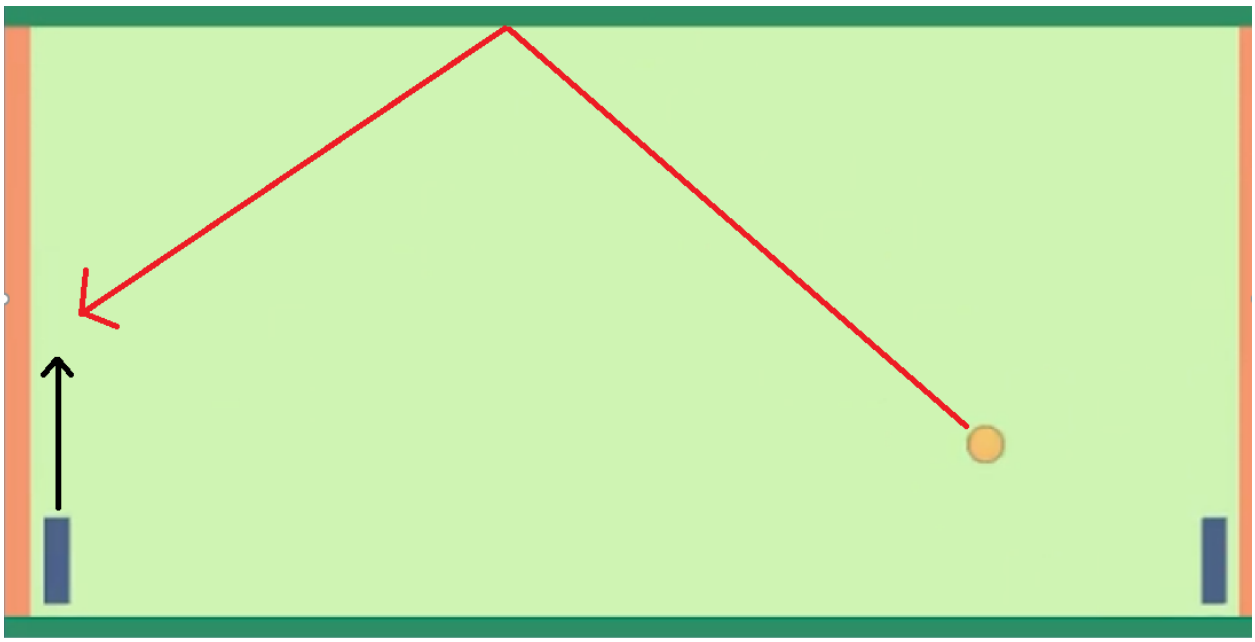
Jei atsakas yra regresinis, tuomet tikslo funkcija tampa didžiausių kvadratų optimizacijos tikslo funkcija ir paskutinis neuronas neturi aktyvacijos funkcijos. Tikslo funkcijos dydžiausių kvadratų sumos vidurkio formulė:

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{8}$$

kur y yra tikroji reikšmė, o $\hat{y}$ yra aproksimuota.

Neuroniniai tinklai su regresinę tikslo funkciją yra puikus agentų apmokymo pavyzdys, įrodantis kad priklausomai nuo užduoties, nebūtina imtis sudėtingų skatinamojo mokymosi metodų, norint sukurti protingą agentą. Su šiuo metodu buvo optimizuojamas stalo teniso žaidimas, kuriame atsakas buvo stalo teniso raketės pozicijos optimali padėtis atmušti atskriejanti kamuoliuką, kaip tai matosi 5 paveikslėlyje. Tai buvo paprastas gilus neuroninis tinklas su dveis paslėptais sluoksniais po 12 neuronų. Buvo pasitelktas stochastinis gradientinis nuolydis, kuomet kiekvieno žaidimo kadro metu buvo nusiunčiami neuroniniam tinklui dabartinės raketės padėtis bei kamuoliuko trajektorija. Šiam uždaviniui neuroninio tinklo nereikia, pakaktu paprasto algoritmo ar tiesinės regresijos, tačiau tai gerai iliustruoja jo veikimo principus. Pradžioje raketės juda padrikai, bet bėgant laikui, vykstant gradientiniui apsimokymui, rakečių judėjimas vis tikslėja.

5 Paveikslėlyje raudona trajektoriją yra neuroninio tinklo įvestis, apskaičiuojanti kamuoliuoko galutinę padėtį. Juoda trajektorija yra neuroninio tinklo išvestis, raketės greičio



5 pav. Sukonstruota stalo teniso simuliacija

vektorius, simbolizuojantis pozicija, kurioje raketė turi atsirasti

1.1.7. Tiesinė regresija su daugybe atsako kintamųjų

Identiška tiesinės regresijos funkcijai, tačiau šiuo atveju paskutinis sluoksnis turi daugiau nei vieną neuroną. Tuomet tikslo funkcija atrodo šitaip:

$$L = \frac{1}{m * 2} * \sum \left(Z^l - Y\right)^2$$

Vektorizacija identiška tiesinės regresijos atveju, tik atsako kintamasis Y turi ne vieną eilutę, o daugiau. Eilutės žymi pasirinkto sluoksnio neuronų reikšmes. Su šiuo metodu buvo optimizuotas automobilio kontroleris, kurio tikslas buvo apvažiuoti trasą. Įvestis buvo normalizuota automobilio sensorių informacija, kurie pateikinėdavo trasos barjero atstumą. Išvestis buvo automobilio stabdymo ir greičio pedalo stiprumai bei vairo pozicija. Apmokius neuroninį tinklą rezultatai buvo tragiški, automobilis važiuodavo tik tiesiai. Tačiau pritaikius Adamo gradientinį nuolydį ir taip sumažinus modelio klaidą, buvo pasiekti tenkinami rezultatai. Automobilis sugebėdavo apvažiuoti trasą.

1.2. Neuroninių tinklų reguliarizacija

Neuroninių tinklų reguliarizacija reikalinga norint išvengti modelio prisitaikymo prie mokymosi duomenų. Ji padeda pagerinti testavimo duomenų tikslumą. Toliau nagrinėsiu metodus, kurie tai padeda pasiekti.

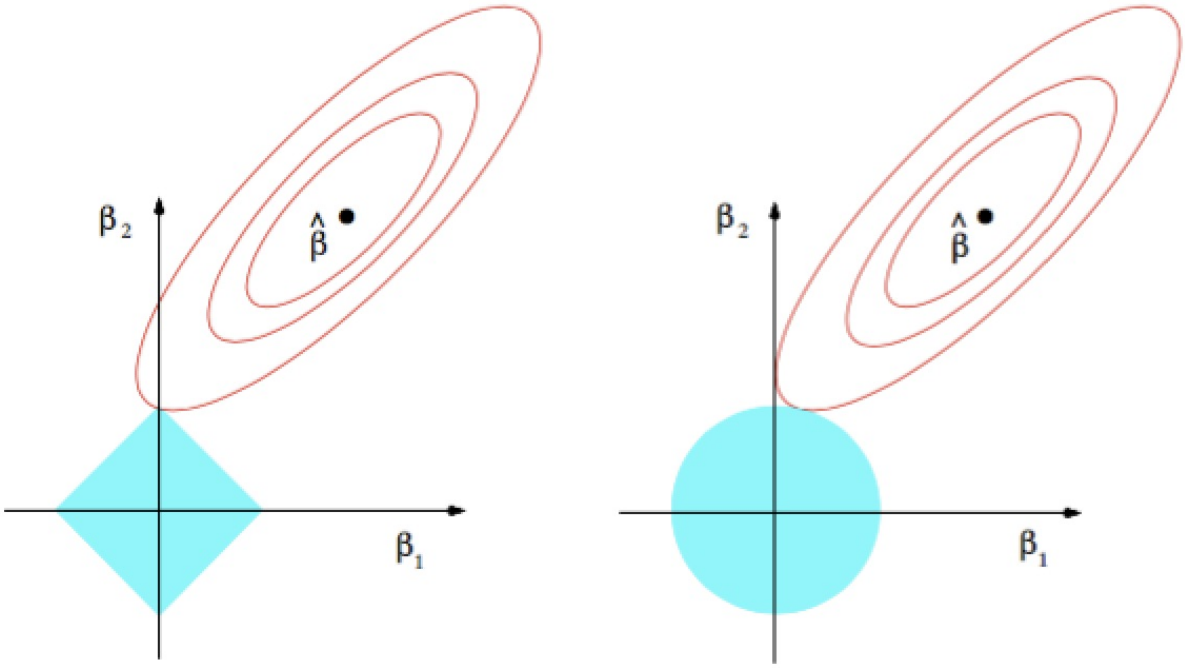
1.2.1. L1 bei L2 reguliarizacija

Didžiuliai svoriai retai gerai generalizuoja modelį, jie numuša testavimo imties tikslumą. Todėl svarbu neleisti svoriams tapti milžiniškais. L1 bei L2 reguliarizacija tai padeda pasiekti. Prie tikslo funkcijos lygties mes pridedame Lagrandžo svorių apribojimus L2 atveju:

$$J(W, b, X, y) = \frac{1}{m} \sum L(y, \hat{y}) + \frac{\lambda}{2 * m} \sum \left\| w^l \right\|^2 ,$$

kur λ yra lagrandžo daugiklis ir taip pat hiperparametras.

Matome, kad prie pagrindinės tikslo funkcijos prisideda papildoma bauda, kuo didesni svoriai, tuo didesnė bauda. L1 atveju mes vietoj ilgio kvadrato, naudojame absoliutinę reikšmę. L2 skiriasi nuo L1 tuo, kad L2 padeda nunulinti tuos svorius, kurie turi mažai įtakos gradientiniam nuolydžiui, kaip tai matosi 6 paveikslėlyje. Tačiau abiejais atvejais mes nepasieksime optimalios reikšmės ir išmainysime „variance“ į „bias”, taip pasiekdami tikslesnį testavimo duomenų klasifikavimą, jei modelis yra per daug prisitaikęs prie mokymo duomenų. Gradientą šiai funkcijai lengva surasti, kadangi matome, kad J funkcija išsiskaido į dvi dalis. Telieka surasti išvestines L2 formai, kas yra labai paprasta.



6 pav. Reguliarizacija - po kaire L1, po dešine L2

1.2.2. Atvirkštinis išmetimas

Šio metodo esmė, išjunginėti kiekvieno sluoksnio neuronus su tam tikra tikimybe. Juos išjungus, kiti neuronai turės perimti jų darbą ir papildomai apsimokyti. Tai padeda išvengti atveju, kuomet vienas neuronas persimoko. Tačiau išmetimo atveju vidutiniškai neuronų išvestis sumažėja išmetimo tikimybės procentu. Todėl atvirkštiniu išmetimu, mes po išmetimo

operacijos kiekvieną neurono Z reikšmę padaliname iš išmetimo tikimybės taip padidindami neurono išvestį ir gražindami to sluoksnio tikėtiną reikšmę.

1.2.3. Duomenų augmentacija

Šis metodas taikomas, kuomet neturime pakankamai duomenų. Jei duomenys susideda iš nuotraukų, tuomet augmentacijos metu mes galime nuotraukas apsukti, priartinti, karpyti ir panašius metodus taikyti, norint padidinti mokymosi duomenų imti.

1.2.4. Įvesties normalizacija

Gradientinio nuolydžio konvergacijos greitis priklauso nuo paviršiaus išsidėstimo. Paviršius gali būti labai susipaudęs bei banguotas, kas neleidžia atlikti didžiulių gradientinių žingsnių. Tokiais atvejais dideli žingsniai, gali šokinėti aplink optimalią reikšmę bei tokiu budu niekados nekonverguoti. Taip pat svoriai nėra proporcingai panašus, vieni turi žymiai didesnę įtaka nuolydžiui, kiti mažesne. Tokiu atveju paimti dideli gradiento žingsniai gali vėlgi nekonverguoti. Tačiau normalizavus įvesties duomenis, paviršius išsilygina bei yra lengviau optimizuoti. Normalizavimas paremtas paprasčiausiu statistiniu Z normalizavimu. Jis atliekamas, kuomet įvesties duomenis pasižymi skirtingais duomenų intervalais – vienos įvesties intervalas [0,1], kitos [0,1000].

1.3. Neuroninių tinklų parametrų inicializacija

1.3.1. Gradiento sprogimas bei nykimas, aktyvacinių funkcijų parametrų priskirimas

„Forward propogation“ neuroninio tinklo žingsnyje, kuomet skaičiuojame išvestį galime pastebėti tokį neuroninio tinklo bruožą

$$\hat{y} = W^l * W^{l-1} * W^{l-2} * \ldots * X$$

Matome, kad jei W^l parametrai priskiriami su didelėm reikšmėm, turint daugybe sluoksnių mūsų tiek neurono tiek svorių reikšmės ekponentiškai išauks. Tuo tarpu jei parametro reikšmės W^l priskiriamos su mažesnėms nei vieneto reikšmėm, svoriai greitai tampa nuliais. Norint to išvengti visų neurono sluoksnio aktyvacijų reikšmių vidurkis turi būti lygus nuliui bei variacija visuose sluoksniuose išlikti vienoda. Šių tikslų padeda pasiekti svorių priskirimo metodai. Dažniausiai svoriai yra paimami iš $\mathcal{N}(0,1) * 0.001$. Šie svoriai nėra blogi jei turime tik pora sluoksnių tačiau didėjant sluoksniams, kaip matėme prieš tai, neuronų reikšmės taps nuliais. Todėl priklausomai nuo sluoksnio aktyvacijos naudojame specifinius metodus priskiriant svorius. Kad šie metodai veiktų, įvestis privalo būti normalizuojama Z-norm principu.

1. Jei aktyvacijos funkcija yra tanh, svoriai imami iš $\mathcal{N}(0, \frac{1}{n^{l-1}})$, kur n yra neuronų skaičius sluoksnyje, šiuo atveju tai būtų praeito sluoksnio neuronų skaičius.
2. Jei aktyvacijos funkcija yra Relu, svoriai imami iš $\mathcal{N}(0,1) * \frac{2}{\sqrt{n}}$.
3. Jei aktyvacijos funkcija yra sigmoidas, svoriai imami iš $\frac{\mathcal{N}(n^{l-1}, n^l)}{\sqrt{n}}$.

1.4. Mokymosi greičio didinimas

Jei turime labai daug duomenų, matricų operacijos užima daug laiko. Todėl naudojame tolimesnius metodus, kurie paspartina mokymosį procesą.

1.4.1. Mažos mokymosi imties metodas

Metodo principas labai paprastas – iš visos duomenų imties paimame dalį skirstinio ir su šia gauta maža mokymosi imtimi apmokome modelį. Kiekviena tokia dalis negražins optimalaus gradiento vektoriaus, tačiau kadangi duomenis yra iš to pačio pasiskirstimo mes vis tiek judėsime į optimalias svorių reikšmes ir galiausiai konverguosime į optimalius svorius.

1.4.2. Stochastinis nuolydis

Beveik toks pats kaip praeitas metodas, tik dar ekstremalesnis atvejis – gradientas gaunamas iš vieno imties elemento, stulpelio. Vektorizacija šiuo atveju praranda visą savo greitį, taip pat gradiento vaikščiojimas neatrodo tolygus, juda vos ne į visas puses. Tačiau bendra gradientų krypties tendencija juda link tikslo funkcijos klaidos mažinimo. To priežastis yra išlieka tokia pati, kaip praeito metodo: kadangi duomuo yra iš to pačio pasiskirstimo mes vis tiek judėsime į optimalias svorių reikšmes. Šis metodas yra nuolatos taikomas skaitinamuosiuose mokymosi metoduose.

1.5. Gradientinio nuolydžio optimizavimas

Norint pasiekti greitesni svorių konvergavimą yra sukurta skirtingų gradientinio nuolydžio metodų, kurie netik pagreitina konvergaciją bet ir padeda kovoti su didelėmis gradiento problemomis – keliavimas plokščiu paviršiu bei lokalus minimumai.

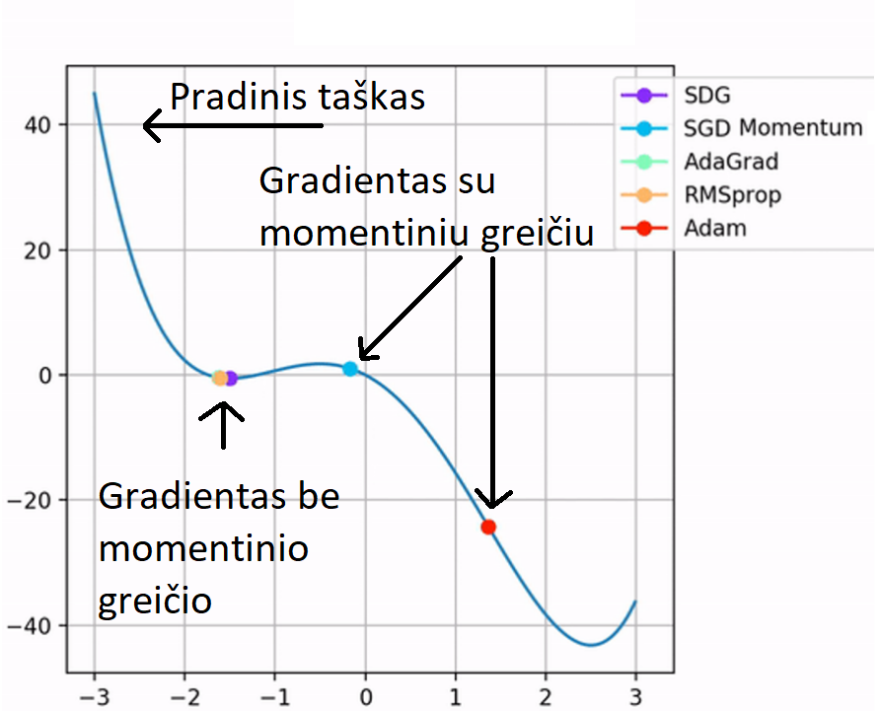
1.5.1. Momentinis gradientinis nuolydis bei eksponentinškai pasvertų svorių vidurkis

Pagrindinis šio optimizavimo principas remiasi momentiniu nuolydžiu, kuomet gradientas įgyja pagreiti ties dažniausiai pasikartojančiu vektoriumi. Primena įsibėgėjusi automobilį. Šis įsibegėjimas padeda išvengti lokalių minimumu, kaip tai matosi 7 paveikslėlyje. Jame pavaizduoti skirtingi gradientinio mokymosi algoritmai, kurie pradeda tame pačiame taške bei kurių dauguma sustoja mokytis ties lokaliu minimumu. Tačiau momentinio mokymusi

paremti gradientinio nuolydžio metodai praskrieja lokalųjį minimumą, kuris tik trumpam sumažina gradiento greitį. Taigi momentinio greičio intuicija labai paprasta: jei gauname naują gradiento reikšmę, kuri nurodo judėti priešinga linke, ji yra atsveriama gradieno įsibėgėjimo greičiu ir gradientas tik truputi suletėja. Šio gradiento formulė primena eksponentiškai pasvertų svorių vidurkio formulę ir išsireiškia tokiu pavidalu:

$$Vdw = \beta_1 * Vdw + (1 - \beta_1) * dw \tag{9}$$

Kur β_1 yra parametras nuo 0 iki 1, dažniausiai būnantis 0.9. Kaip matome naujausi gradientai prie galutinės gradiento reikšmės prisidės tik $(1-\beta_1)$ dydžiu, kas suteikia momentinio greičio pavidalą. Šis pasvertų vidurkių principas yra plačiai naudojamas skatinamuosiuose mokymosi metoduose ir jį galima pastebėti visuose pagrindinėse formulėse. Skatinamuosiuose mokymosi metoduose šios formulės principą galima traktuoti kaip gradientinį nuolydį - po truputi konverguojame ties optimaliomis reikšmėmis.



7 pav. Gradientinė optimizacija su ir be momentinio greičio. SGD momentinis bei Adam konverguoja į globalų minimumą, like konvergavo į lokalų minimumą

1.5.2. Normalizuotas gradientinis nuolydis

Niekur nenaudojamas gradientinio nuolydžio metodas, kurio metu gradiento dydis normalizuojamas. To pasekoje net ir plokštumoje greitis nėra nulinis. Tačiau mokymosi procesas yra labai lėtas, nes gradiento smarkus nuolydis yra normalizuojamas.

1.5.3. RMSProp

Tai gradientinio nuolydžio metodas, kurio tikslas pagreitinti gradientą, kuomet judame lygia plokštuma. Tačiau kaip matome iš 7 RMSProp kenčia nuo momentinio greičio neturėjimo ir sustoja ties lokaliu minimumu. RMSProp forma labai panaši į momentinio gradiento:

$$Sdw = \beta_2 * Sdw + (1 - \beta_2) * (dw)^2 \tag{10}$$

Ir svorių atnaujinimo formulė:

$$W = W - \alpha \frac{dw}{\sqrt{Sdw}} \tag{11}$$

Empiriškai galime pastebėti, kad didžiulios gradiento reikšmės sumažės, tačiau mažos padidės. Todėl šis metodas padeda išspręsti lėtą judėjimą lygumomis. Rekomenduojamas β_2 dydis yra 0.99.

1.5.4. Adam

Šis metodas yra vienas populiariausių gradientinio nuolydžio metodų. Jo tikslas sumažinti dvi problemas - lokalius minimumus bei judėjima plokštuma. Tai pasiekiame sukombinuojant momentininį greitį bei RMSProp:

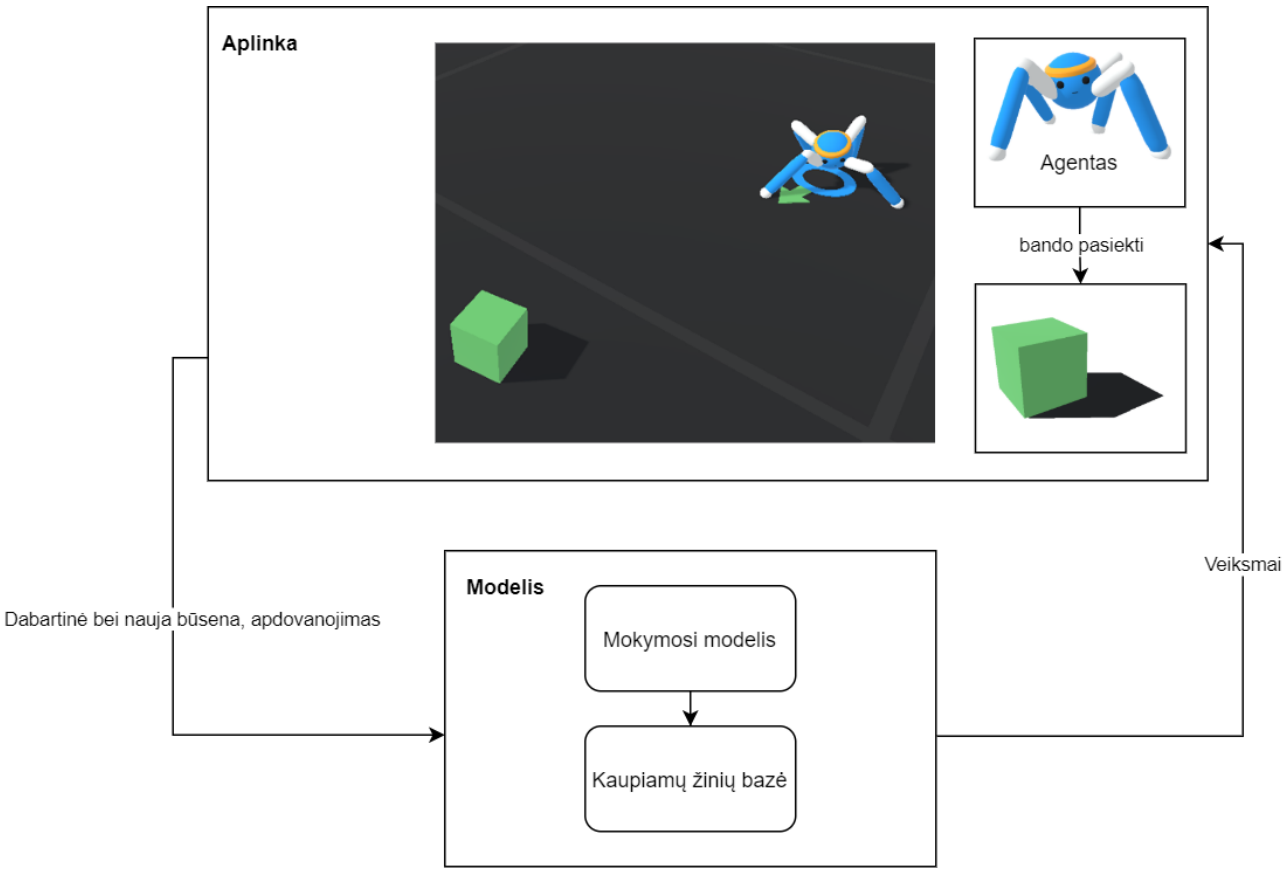
$$W = W - \alpha \frac{Vdw}{\sqrt{Sdw}} \tag{12}$$

Vdw suteikia momentinį greitį kiekvienam svoriui. Bet jei momentinis greitis yra didelis Sdw jį sumažiną. Tačiau jei gradientas užstringa lygumoje, Sdw padeda atstrigti. Šį metodą pritaikiau minėtame automobilio kontrolerio pavyzdyje. Jis padėjo pasiekti trygubai mažesnį klaidos dydį ir taip pagerino automobilio kontrolerį.

1.6. Skatinamasis mokymasis

Skatinamojo mokymosi metodai yra mašinio mokymosi metodai, kurie apdovanoja pageidaujamus veiksmus bei baudžia nepageidautinus veiksmus. Naujuosiuose skatinamuosiuose modeliuose apdovanojimo bei baudos dydis tėra vienintelė informacija, kurią modelis težino. Jei tiesinėje regresijoje mes nuolatos turėdavome tikrąsias reikšmes, ties kuriomis bandėme pritaikyti modelį, šiuo atveju tokia informacija yra nepasiekiama. Todėl mūsų modelis turi nuolatos tyrinėti aplinką, kaupti naujas patirtis. Šiuos duomenis gauname iš aplinkoje esančio roboto ar agento, kuris duotoje aplinkoje stengesi atlikti optimalius veiksmus ir pagal juos susidaryti geriausių siūlomų veiksmų modelį.

Šią veiksmų eigą apibendrina bendrasis skatinamųjų modelių grafikas, pateiktas 8 paveikslėlyje. Jis iliustruoja praktinės dalies pagrindinės užduoties įgyvendinimą, voro aštuonių galunių judinima, kuriomis bandoma kuo arčiau priartinti vorą ties žaliu kubeliu.

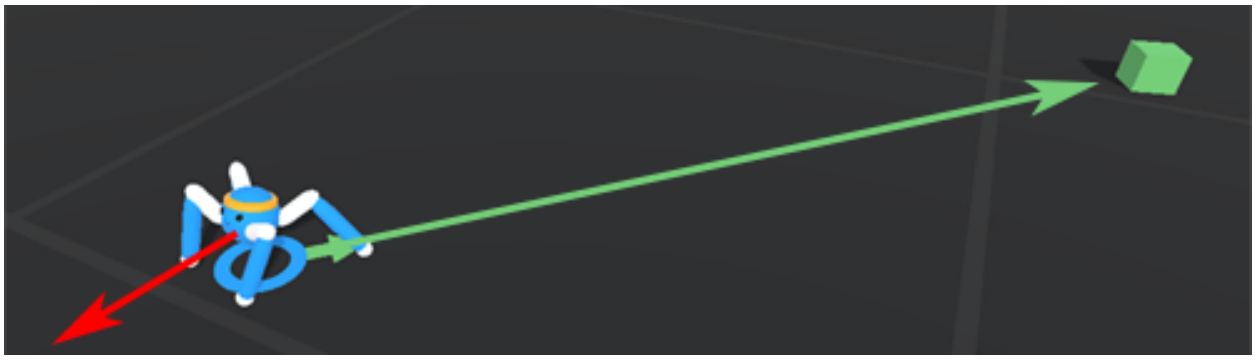


8 pav. Bendras skatinamojo mokslo modelis

Šio modelio veikimo principai išsiskaido į šias dalis:

1. Aplinka. Aplinkoje egzistuoja mūsų agentas, kuris priiminėje modelio sugeneruotus veiksmus. Pateiktame pavyzdyje tai būtų voro galūnių judėjimo trajektorijos. Atlikes duotus veiksmus, voras įgauną naujas būsenas, pasistumį į naują poziciją. Nauja įgyta būsena bei prieš tai buvusi, nusiunčiama modeliui. Taip pat modeliui nusiunčiame apdovanojimo dydį. Jis apskaičiuojamas naudojantis naujai sugeneruotos būsenos naudingumu. Naudojantis voro pavyzdžiu tai atitiktų voro greičio vektorio laipsnio sutapimu su norimos krypties vektoriumi, kaip tai matosi 9 pavekslėlyje. Šiuo atveju apdovanojimas būtų nulinis, nes vektorių trajektorijos visai nesutampa. Atlikus veiksmą bei apskaičiavus jo naudingumą, tokiu budu gaunamas ryšys tarp veiksmo, kuris sugeneruoja naują būseną, bei įgyto apdovanojimo, naujoje būsenoje. Šis ryšys tarp veiksmo bei gauto rezultato yra neseniai atgimusio skatinamojo moklso pamatas, kuriuo moderniausi algoritmai ne per seniausiai pradėjo vadovautis, atradus apdovanojimų taisyklių gradientą bei pradėjus taikyti neuroninius tinklus.
2. Agentas. Agento sluoksnis susideda iš mokymosi modelio bei žinių bazės, kurioje kaupiami gauti aplinkos rezultatai, susidedantis iš vektorių su šiomis reikšmėmis - praeita būsena, atliktas veiksmas, nauja būsena ir apdovanojimas. Mokymosi modelis naudojasi sukauptų žinių baze gerinti savo modelio rezultatus. Modelio tikslas yra pateikti tokias veiksmų sekas, kurios suteiktų dydžiausią įmanoma apdovanojimą. Voro pavyz-

džiu tai būtų sėkmingas žalio kubelio pasiekimas. Modelio apmokymui yra sukurta daugybe įvairių metodų ir šiame darbe bus nagrinėjamas vienas iš jų - TD3



9 pav. Apdovanojimo pavyzdys, pagal voro greičio (raudona rodyklė) bei norimo judėjimo (žalia rodyklė) vektorių atitikimu.

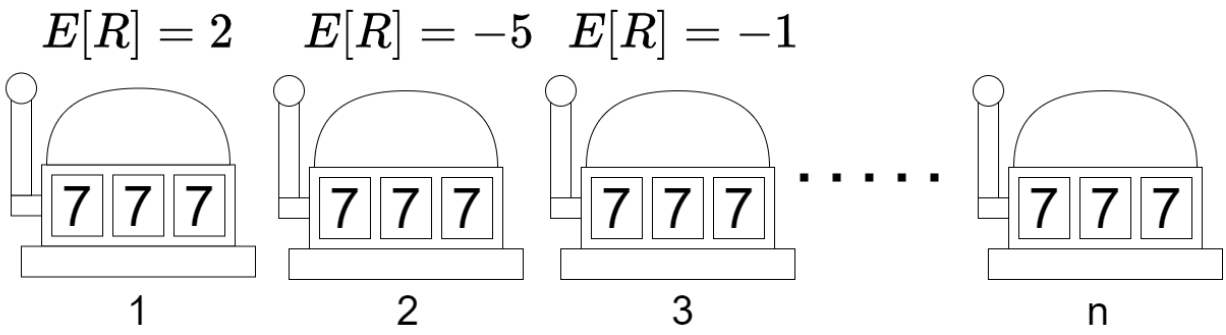
Taigi visas skatinamasis mokslas paremtas tik vienu kintamuoju - apdovanojimo gausumu. Šis apdovanojimas apdovanoja pageidautina elgesį bei baudžia neigiamą elgesį. Tokiu budu agentas skatinamas siekti dydžiausio ilgalaikio nuopelno, kas veda ties optimaliu sprendimu. Tolimesniuose skyriuose bus aptarema kaip iš šio vieno kintamojo yra konstruojamas visas mokslas bei įvairiausi modeliai.

1.6.1. Skatinamojo mokslo pagrindiniai kintamieji, stacionariojo pasiskirstymo problema

Kad ir koki skatinamojo mokslo uždavinį spręstume, mes visados sutiksime šiuos kintamuosius:

1. t - $\mathbb{N}$ laiko žingsnis. Kiekviena kart atlikus veiksmą bei apskaičiavus jo naudingumą, laiko žingsnis pajuda vienetu į priekį
2. A_t - atliktas veiksmas t laiko žingsnyje.
3. R_t - gautas apdovanojimas atlikus A_t veiksmą.
4. $Q(a)$ - vidurkis, tikėtina apdovanojimų reikšmė pasirinkus veiksmą a. Kitaip tariant $Q(a) = \mathbb{E}[R_t|A_t = a]$. Ši formulė ypatingai svarbi, kadangi iš jos vos ne visi mokymosi metodai yra išvedami.

Dažniausiai pateikiamas pavydzys, kuris iliustruoja šio mokslo principą bei padeda geriau suprasti šiuos kintamuosius, yra lošimo aparatu, pavaizduotų 10 paveikslėlyje, laimėjimų optimizavimas. Šiame uždavinyje iš n aparatų agentas bando surasti lošimo aparatą, kuris suteikia dydžiausią įmanomą apdovanojimą. Kiekvienas aparatas duoda atsitiktinį laimėjimą iš $\mathcal{N}(\mu = \mathcal{N}(0,2),\sigma = 1)$ pasiskirstimo. Žaidimo sesija, dažniausiai vadiname episodas, nėra ribojima, agentas gali atlikti begalybe t žingsnių, lošimo aparatų pasirinkimų. Pasirinkimą galima traktuoti kaip veiksmą a ir gautą laimėjimą kaip r. Agentas bando vienos



10 pav. N lošimo aparatų, su skirtingais vidutiniais laimėjimais

žaidimo sesijos metu įgyti dydžiausią sukauptą laimėjimą. Jis pritaiko viena iš populiariausiu skatinamojo mokslo metodu $\epsilon - greedy$. Šis metodas visados renkasi lošimo aparatą, kurio aproksimuotas tikėtinas laimėjimas, žymimas $Q(a)$, yra dydžiausias, kaip tai išreikšta 13 formulėje.

$$A_t = argmax_a Q_t(a) \tag{13}$$

Tačiau su maža ϵ tikimybe, agentas pasirenka atsitiktinį lošimo aparatą. Taigi kiekviename t žingsnyje agentas renkasi lošimo aparatą, jį aktyvuoja ir gauną naują apdovanojimą, kuri talpina į to aparato apdovanojimų sarašą. Šis procesas ypatingai svarbus, jis dominuoja visuose mokymosi modeliuose. Agentas stengiasi netik gauti dydžiausia įmanoma tikėtina laimėjimą, tačiau kartu tirinėja aplinką, ieško optimalesniu lošimo aparatu, nei surastas dabartinis geriausias. Šis procesas vadinasi apdovanojimų-tirinėjimo kompromisu. Lošimo aparato tikėtinas laimėjimas tėra to aparato gautų laimėjimų vidurkis, kaip tai matosi 14 formulėje.

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i * \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \tag{14}$$

Kitaip tariant $Q_t(a)$ yra apdovanojimų suma, kuomet pasirinkome a veiksmą (arba lošimo aparatą), atlikus t žingsnių, padalinti iš kiek kartų buvo a veiksmas (arba lošimo aparatas) pasirinktas. Kuomet t artėja link begalybės, pagal didžiųjų skaičių dėsniu, $Q(a)$ priartėja prie tikrosios reikšmės. Visi naujausi modeliai yra paremti panašiu imties traukimo metodu - simuliuojame skirtingus veiksmus ir stebime gautus rezultatus. Nors pavyzdys atrodo primitivus, tačiau visi naujausi modeliai sprendžia tą pačią užduoti - kokį pasirinkti veiksmą a, kuris suteiktų dydžiausią apdovanojimą.

Pagrindiniai skirtumai atsiranda dėl kompiuterio resursų optimizavimo. Viena iš problemu, su kuria iškarto susiduriame, yra duomenų kaupimas. Kiekvienam lošimo aparatui reikia kaupti jo gautus apdovanojimus. Didėjant problemai bei jos duomenų dimensijai, kompiuteriui kaupti duomenis tampa fiziškai nebeįmanoma, todėl yra sukurta daugybe esminių metodų šią problemą mažinti bei išvengti. Vienas iš metodų, kurio formą galima

pastebėti TD formulėje, yra rekursyvus vidurkio apskaičiavimas, pateiktas 15 formulėje. Ši formule padeda išvengti apdovanojimų kaupimo, kaip tai matėme lošimo pavyzdyje.

$$\begin{aligned} Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\ &= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\ &= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\ &= \frac{1}{n}\left(R_n + (n-1)Q_n\right) \\ &= \frac{1}{n}\left(R_n + nQ_n - Q_n\right) \\ &= Q_n + \frac{1}{n}\left[R_n - Q_n\right] \end{aligned} \tag{15}$$

Lošimų pseudokodas, kuris palaipsniui apskaičiuoja skirtingų veiksmų imties vidurkius, pateiktas toliau:

Algoritmas 1: Lošimo aparato pseudo algoritmas

$Q(a) \leftarrow 0$;
$N(a) \leftarrow 0$;
for ∞ **do**

$$A \leftarrow \begin{cases} \arg\max_a Q(a) \text{ su tikimybe } 1-\varepsilon \text{ (lygiąsias reikšmes pasirenkame atsitiktinai)} \\ \text{atsitiktinis veiksmas su tikimybe} \varepsilon \end{cases}$$

$R \leftarrow$ LošimoAparatas (A) ; `// Pasirenkame veiksma A ir gauname apdovanojimą R`
$N(A) \leftarrow N(A) + 1$; `// Kiek kartų kiekvienas veiksmas buvo pasirinktas`
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$;

Rezultatas: Gautos optimalios, tikrosios Q(a) reikšmės. Jas turint žinosime, kuris aparatas duoda dydžiausią laimėjimą

Paskutinė 15 formulės išraiška yra įpatingai svarbi, nes ji yra ištaka tiek TD, tiek Q-mokymosi, tiek bellmano formulėms, kurios yra skatinamojo mokslo pamatas. Apibendrinant formulę ją galima šitaip išreikšti:

$$\text{Naujas vidurkis} = \text{Senas vidurkis} + \frac{1}{n}(\text{Nauja reikšmė - Senas vidurkis}) \tag{16}$$

Jei tęstume formules apibendrinimą išriškėtu bellmano funkcija, kuri yra viena svarbiausiu šiame moksle ir į kuria gilinsimes tolimesniuose skyriuose:

$$\text{Naujas įvertis} = \text{Senas įvertis} + \frac{1}{n} * (\textbf{Tikslas - Senas įvertis}) \tag{17}$$

Paryškinta dalis yra vadinama TD paklaida, ties kuria gilinsimes tolimesniuose skyriuose.

Kolkas nagrinėta problema pasižymi apdovanojimo skirstinio stacionarumu, jis nekinta. Tačiau nagrinėjant sudėtingas problemas netik mūsų apdovanojimų pasiskirstimas kinta, tačiau mum idomios tik paskutinės 10 -100 reikšmės bei aukštos dimensijos problemose mum neužtenka kompiuterio resursų išsaugoti $N(A)$ reikšmių, ką mes atliekame 1 pseudokode. Todėl įgyvendinamas dar vienas šios formulės optimizavimas

1.6.2. Ne stacionarus pasiskirstymas

Galima pastebėti daugybe problemų su 15 formule:

1. Toliau plėtojant duotą pavyzdį, galima įsivaizduoti situaciją, kurioje, kas tam tikrą laiko intervalą, kiekvieno aparato apdovanojimo funkcija pakinta ir įgyja naują $\mathcal{N}(\mu = \mathcal{N}(0{,}2), \sigma = 1)$ pasiskirstymą.
2. Kuomet vieno lošimo aparato apdovanojimų R imtis yra labai didelė, naujai gautos reikšmės turi labai mažai įtakos vidurkio pokyčiui dėl $\frac{1}{n}$ formulėje esančios išraiškos.
3. Kiekviena $Q(a)$ reikšmė turi kaupti $N(A)$. Aukštosiose dimensijos neužtenka kompiuterio resursų išsaugoti šias reikšmes.

Šios problemos sprendimas jau buvo pateiktas Adamo algoritmo įgyvendinime, mes toliau naudojame judančio vidurkio principus:

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha\left[R_n - Q_n\right] \\ &= \alpha R_n + (1-\alpha)Q_n \\ &= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\ &= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\ &= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} + \\ &\quad \ldots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\ &= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i \end{aligned} \tag{18}$$

Ši 18 formulė pakeičia $\frac{1}{n}$ į $\alpha \in [0{,}1]$. Dažniausia α reikšmė yra 0.9, su kuria apytiksliai apskaičiuojame paskutinių gautų 10-ties narių vidurkį.

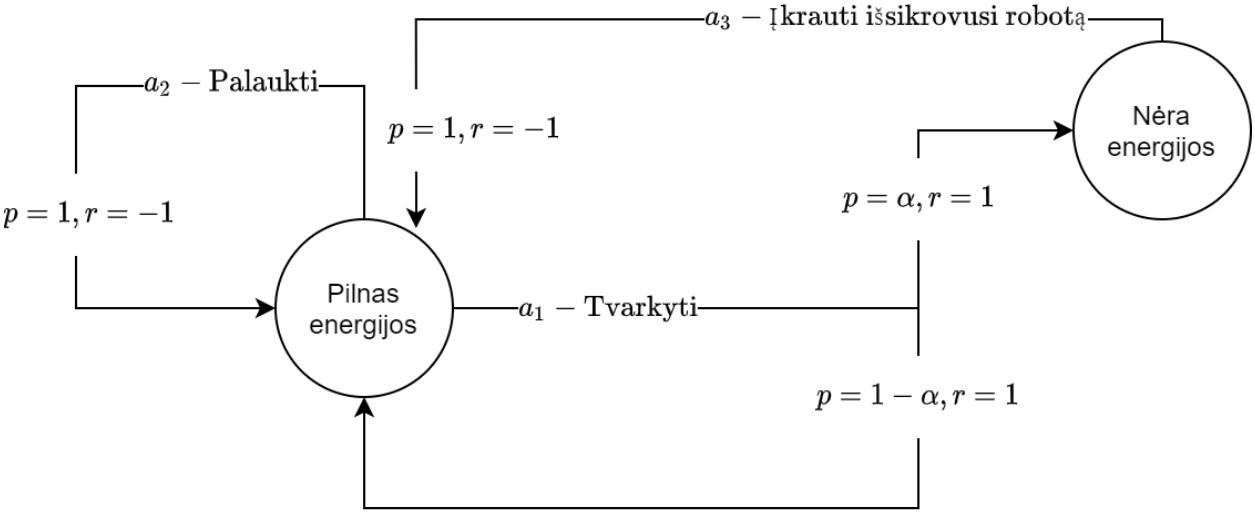
1.6.3. Markovo grandinės

Dauguma mažos dimensijos skatinamojo mokslo problemų galima išreikšti markovo grandinių pavidalu. Toks išreiškimas padeda geriau suvokti keturias pagrindines skatinamojo mokslo savokas:

1. s - esama būsena.
2. a - pasirinktas veiksmas.
3. r - esamosios būsenos pasirinkto veiksmo apdovanojimas.

4. s' - nauja būsena, kuri gaunama atlikus a veiksmą.

Nors dažniausiai būsenos būna daugiau nei vienos dimensijos, pavyzdžiui būsenos vektorius susidedantis iš 2 reikšmių - pozicijos bei greičio, lengviausiai jas yra suprasti atvaizduojant vienos dimensijos markovo grandinių pavidalu, kaip tai matosi 11 paveikslėlyje.



11 pav. Dulkio siurblio markovo grandinė

Jei dulkio siurblio robotas yra būsenoje $s =$?Pilnas energijos?, jis gali pasirinkti a_1 arba a_2 veiksmus. Pasirinkęs a_1 veiksmą, robotas su α tikimybe gali patekti į būsena $s' =$?Nėra energijos? ir su tikimybe $1 - \alpha$ į būseną $s' =$?Pilnas energijos?. Už abu šiuos veiksmus veiksmus robotas yra apdovanojamas $r = 1$. Šis procesas galioja ir kitoms būsenoms.

Priklausomai nuo esančios būsenos, mes visa laika norime pasirinkti tokį veiksmą, kuris duoda dydžiausią tikėtiną apdovanojimą:

$$r(s, a) \doteq \mathbb{E}\left[R_t \mid S_{t-1} = s, A_{t-1} = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p\left(s', r \mid s, a\right) \tag{19}$$

Veiksmų pasirinkimas vadovaujantis dydžiausiu tikėtinu apdovanojimu bus nagrinėjamas sekančiame skyriuje.

Markovo grandinių išraiška taip pat yra svarbi ne tik dėl vizualinio aiškumo. Visos skatinamojo mokymo problemų formuluotės privalomai turi galėti išsireikšti markovo grandinių pavidalu. Šis reikalavimas yra būtinas dėl vienos iš pagrindinių markovo grandinės savybių - ateities įvykio tikimybė nepriklauso nuo prieš tai įvykusių įvykių:

$$P\left(X_{n+1} = x \mid X_0, X_1, X_2, \ldots, X_n\right) = P\left(X_{n+1} = x \mid X_n\right), \tag{20}$$

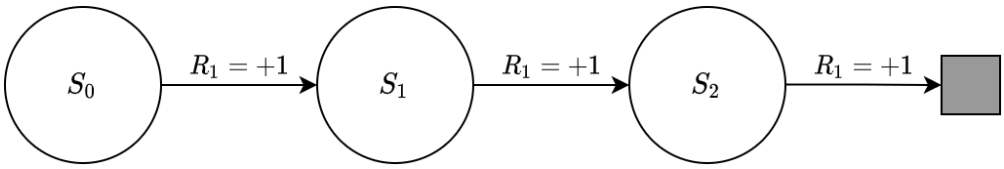
kur X yra prieš tai buvę įvykiai. Ši savybė leidžia apskaičiuoti tikėtiną apdovanojimą pasitelkiant tik turima busena, prieš tai buvusios būsenos neturi įtakos tikėtinam apdovanojimui.

Taigi 11 pavyzdys iliustruoja kaip skatinamajame moksle yra lengva apskaičiuoti naudingiausius veiksmus, kuomet turime sudaryta modelį. Tačiau retu atveju mes jį turime ir dažniausiai mum reikia imtis aplinkos tirinėjimo veiksmų, kurie sudaro modelio aproksimacijas. Taip pat mes ne tik norime apskaičiuoti geriausią sekančio veiksmo apdovanojimą, tačiau geriausią apdovanojimą ilgalaikėje perspektyvoje. Dažnai gyvenime mes pasirenkame mėgautis maloniais dalykais, kuriais vėliau mes gailimes. Būtent šią problemą kitas skyrius ir nagrinės - ilgalaikis apdovanojimas, o ne trumpalaikis.

1.6.4. Apdovanojimai bei tikslai

Adpovanojimų hipotezė: Kad visa tai, ką mes turime omenyje kalbėdami apie tikslus ir uždavinius, gali būti gerai suprantama kaip tikėtinos vertės maksimizavimas, kai gaunama ilgalaikė skalarinio signalo (vadinamo apdovanojimu) suma.

Šią sumą mes vieno epizodo metu stengemes maksimizuoti. Epizodas - būsenų, veiksmų, apdovanojimų seka, kuri užsibaige agentui pasiekus tikslą arba sustojimo sąlygą, kaip tai matome 12 paveikslėlyje. Agentas renkasi skirtingus veiksmus, įgauna naujas būsenas, kaupia epizodo apdovanojimą, kol galiausiai pasieke tikslą, pavaizduotų pilku kvadratu.



12 pav. Vienas agento epizodas

Epizodo galutinis sukauptas apdovanojimas užrašomas tokiu pavidalu:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \tag{21}$$

kur T yra paskutinis epizodo žingsnis. Tačiau jei agentas optimizuoja savo veiksmus aplinkoje, kurios pabaigą neįmanoma identifikuoti arba epizodas trunka nepaprastai daug žingsnių, mes naudojame apdovanojimų proporcionalius mažinimus:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{22}$$

kur γ yra **?**nuolaidos**?** kintamasis, kurio dažniausia vertė yra 0.9. Ši vertė indikuoja, kad 11 žingsnio apdovanojimas bus proporcianalus 0.9^{11} savo vertės, kitaip tariant nuliui. Tačiau ši formulė kaip ir praeitos pasižymi panašiais trūkumais, kaip nuolatinis duomenų kaupimas. Tai galima išspręsti perrašius formulę į rekursinę formą:

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{23}$$

Taip pat ši formulė padeda pasiekti balansą ties ilgalaikio apdovanojimo bei trumpalaikio apdovanojimo optimizavimo.

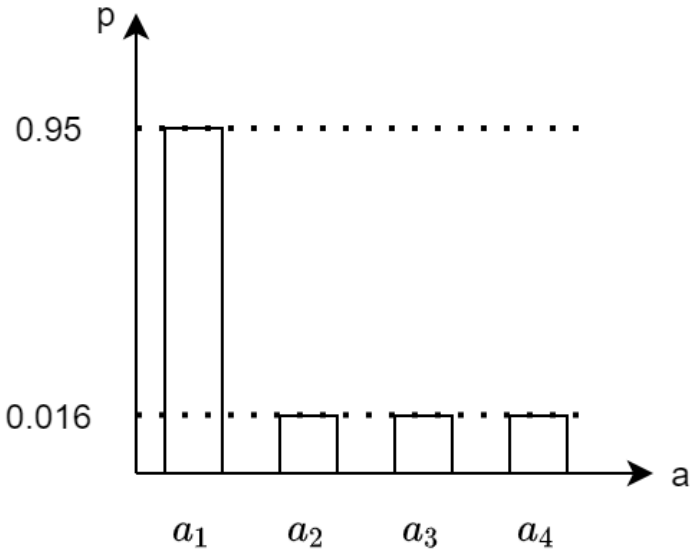
Vienas iš svarbiausių praktinių patarimų, padedančių pasiekti dydžiausią ilgalaikį apdovanojimą yra priskirti apdovanojimus už pasiektą tikslą, o ne skirti apdovanojimus už tai kaip mes norėtume, kad agentas pasiektų tikslą. Agentas mokymo procese turi pats atrasti optimaliausius veiksmus vedančius ties tiek tikslo pasiekimu, tiek dydžiausiu epizodo apdovanojimu. Kitame skyriuje bus apibrežiama, kaip būtent agentas optimizuoja savo elgesį, jo viena iš tikslo funkcijos konceptų, kurį gradientiniai metodai nuolatos optimizuoja.

1.6.5. Vertės bei veiksmų taisyklių funkcijos

Norint pasirinkti veiksmus, vedančius į būsenas, kurios suteikia dydžiausią įmanoma apdovanojimą, mes turime kiekvienai busenai priskirti tam tikrą matą. Todėl kiekviena būsena yra nusakoma pagal savo vertę. Ši vertė yra žymima $v_\pi(s)$ ir nusako pasirinktos s būsenos tikėtiną apdovanojimą:

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{ for all } s \in \mathcal{S} \tag{24}$$

Žinant kiekvienos būsenos tikėtina apdovanojimą, mes visados galime rinktis tik tuos veiksmus, kurie veda į dydžiausią apdovanojimą, dydžiausią $v_\pi(s')$. Šį pasirinkimą nusako veiksmų taisyklės. Tai yra tikimybių pasiskirstimas žymimas $\pi(a|s)$, nurodantis būsenos s visų veiksmų pasirinkimu pasiskirstymą. Kitaip tariant su kokia tikimybe, mes pasirinktume veiksmą a būnant būsenoje s, kaip tai matosi paveikslėlyje 13.



13 pav. Veiksmų taisyklių pasiskirstymas, kur ϵ =5

Užrašymas $v_\pi(s)$ nurodo, kad ši būsenos vertė yra apskaičiuota naudojantis π veiksmų taisyklėmis. Veiksmų taisyklės gali būti įvairios. Dažniausiai naudojame $\epsilon - greedy$ metodą,

kaip tai matėme ties 13 formule bei kurio pritaikymas markovo grandinėms yra pateiktas 1.6.7 formulėje. Šio metodu mes visalaika priskiriame $1-\epsilon$ tikimybę veiksmui, kuris gražina dydžiausią apdovanojimą apskaičiuojama naudojantis $v_\pi(s)$ formule.

$$\begin{aligned}\pi'(s) &\doteq \arg\max_a q_\pi(s,a) \\ &= \operatorname*{argmax}_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) \mid S_t = s, A_t = a\right] \\ &= \operatorname*{argmax}_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right]\end{aligned} \tag{25}$$

Turint kiekvienos būsenos taisyklių pasiskirstymą, galima suskaičiuoti kiekvienos būsenos tikėtiną reikšmę, kaip tai matosi 26 formulėje.

$$\begin{aligned}v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right)\left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\right] \\ &= \sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right], \qquad \text{kiekvienam } s \in \mathcal{S},\end{aligned} \tag{26}$$

Šios formulės paskutinė lygybė yra dar geriau žinoma kaip bellmano lygybė. Ji padeda rekursiškai apskaičiuoti $v_\pi(s)$ reikšmes bei išvengti apdovanojimų kaupimo kompiuterio atmintyje.

Tačiau praktikoje mes naudojame ne $v_\pi(s)$ - būsenos vertės matą, o $q_\pi(s,a)$ - veiksmo vertės matą esant būsenoj s:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \tag{27}$$

Galime pastebėti, kad $v_\pi(s)$ yra tos būsenos s visų veiksmų $q_\pi(s,a)$ proporcinga $\pi(a|s)$ sudėtis bei taip pat išsireiškia bellmano lygybe:

$$\begin{aligned}&q_\pi(s,a) = E_\pi\left[G_t \mid S_t = s, A_t = a\right] = \\ &= \sum_{s'} \sum_r p\left(s', r \mid s, a\right)\left[r + \gamma E_\pi\left[G_{t+1} \mid S_{t+1} = s'\right] = \right. \\ &= \sum_{s'} \sum_r p\left(s', r \mid s, a\right)\left[r \cdot \gamma \sum_{a'} \pi\left(a' \mid s'\right) \cdot E_\pi\left[G_{t+1} \mid S_{t+1'} = s', A_{t+1} = a'\right]\right] \\ &= \sum_{s'} \sum_r p\left(s' r \mid s, a\right)\left[r + \gamma \sum_{a'} \pi\left(a' \mid s'\right) \cdot q_\pi\left(s', a'\right)\right],\end{aligned} \tag{28}$$

kur $v_\pi(s') = \gamma E_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]$.

$q_\pi(s,a)$ Formos dėka mum nereikia apskaičiuoti s būsenos visų veiksmų apdovanojimus norint gauti $v_\pi(s)$. Užtenka rinktis tik tuos veiksmus, kurie yra jau apskaičiuoti bei optimaliausi.

Taigi turint būsenų verčių bei veiksmų taisyklių apskaičiavimo formules galime pradėti iteracini procesą, kurio tikslas yra optimizuoti šias reikšmes.

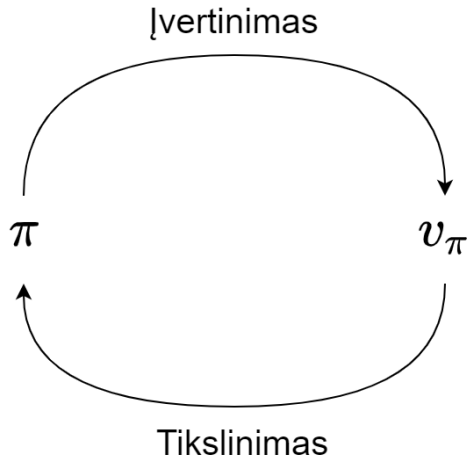
1.6.6. Verčių bei veiksmų taisyklių iteravimas

Optimizavimo metu, mes ieškome tokių būsenų verčių bei optimalių taisyklių, kurios gražintų dydžiausią epizodo apdovanojimą. Yra daugybe būdų tai atlikti, kuriuos galima sugrupuoti į trys kategorijas:

1. Dinaminio programavimo metodas, kuomet modelis žinomas, mes kiekvienos iteracijos metu perskaičiuojame visas $v_\pi(s)$ reikšmes ir atnaujiname veiksmų taisykles. Šį procesą iteruojame, iki konvergacijos pasiekimo. Nenaudojamas metodas, nes labai retai tenkinama šio metodo privaloma salyga - modelio turėjimas. Modelį mes traktuojame kaip markovo grandinių modelį, su visų veiksmų perėjimo tikimybėmis.
2. Monte Carlo metodai - imties metodais apskaičiuojame $q_\pi(s, a)$ reikšmes. Šis metodas taikomas kuomet neturime modelio ir reikia jį aproksimuoti.
3. Neuroninių tinklų aproksimacijos - metodai, kurie aproksimuoja $q_\pi(s, a)$ reikšmes. Jei monte carlo metodai apskaičiuodavo kiekvienos būsenos tikslias reikšmes, funkcijų aproksimacijos metodais mes jas apskaičiuojame tik apytiksliai.

Visos šios kategorijos pasižymi bendru optimizavimo principu:

1. Įvertinimas - Perskaičiuojame $v_\pi(s)$ reikšmes naudojantis atnaujintomis veiksmų taisyklėmis.
2. Tikslinimas - Turint naujas $v_\pi(s)$ reikšmes, perskaičiuojame optimalias veiksmų taisykles $\pi(a|s)$.



14 pav. Viena optimizacijos iteracija

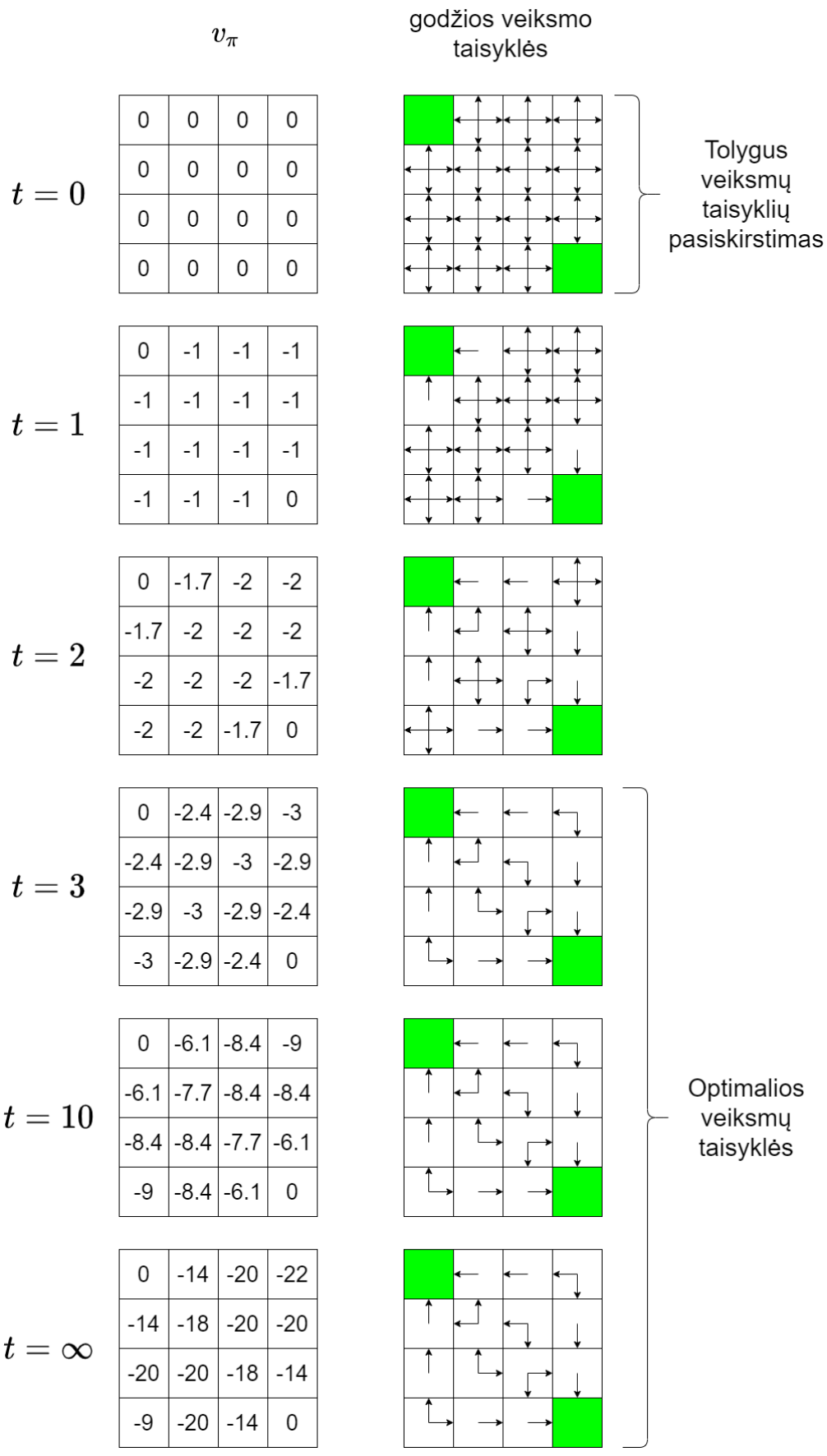
Šiuos žingsnius kartojame tol, kol pasiekiame $v_\pi(s)$ konvergaciją - naujos iteracijos nepakeičia $v_\pi(s)$ reikšmių. Tolimesniuose skyriuose trumpai apžvelgsiu šio proceso įgyvendinimą dinaminiuose bei monte karlo metoduose. Neuroninių tinklų aproksimacijos bus plėtojamos giliausiai, kadangi šiuo metu jie yra labiausiai paplite bei efektyviausi.

1.6.7. Dinaminis programavimas

Įvertinimo bei tikslinimo procesą, pateikta 14 paveikslėlyje, paprasčiausiai iliustruoja dinaminio programavimo metodas pateiktas 15 pavyzdyje. Šiame pavyzdyje mūsų agentas atsiranda viename iš keturiolikos langeliu ir gali eiti į keturias puses, pateiktomis rodyklėmis. Agento tikslas patekti į žalius langelius. Kairėje paveikslėlio pusėje yra pateikti $v_\pi(s)$ įverčiai naudojantis 26 formule, o dešinėje perskaičiuojame optimalias veismų taisykles naudojantis formule. Ypatingai svarbi optimizavimo seka:

1. Įvertinimas - Perskaičiuojame $v_\pi(s)$ reikšmes atnaujinamos naudojantis $t-1$ žingsnio taisyklėmis.
2. Tikslinimas - Perskaičiavus $v_\pi(s)$ reikšmes, perskaičiuojame t žingsnio optimalias veiksmų taisykles $\pi(a|s)$. Pavyzdyje naudojame godų veiksmo taisyklių įvertinima, kuris renkasi tik optimaliausius veiksmus.

Vienas agento žingsnis, žymimas t, susideda iš šios įvertinimo bei tikslinimo sekos. Matome, kad užtenka vos tryjų žingsnių veiksmų taisyklėms tapti optimaliomis. Tačiau $v_\pi(s)$ reikšmėmių konvergacija užtrunka ilgesni laiką. Įvykus konvergacijai, agentui atsiradus ant bet kurio iš galimų baltų langelių, jis, vadovaudamasis veiksmų taisyklėmis, visados optimaliai suras kelią į žalius langelius. Taip pat labai svarbu pastebėti seniau aptarta $v_\pi(s)$ išsiskaidymą į $q_\pi(s,a)$ narių sumą. Šis išsiskaidymas intuityviai matosi pateiktame 15 paveikslėlyje, kuomet kiekvieną rodyklę, galima traktuoti kaip $q_\pi(s,a)$ reikšmę. Šis išskaidymas yra svarbus neuroninių tinklų aproksimacijose, kaip tai bus aptarta tolimesniuose mokymosi modeliuose. Tačiau šis algoritmas yra tinkamas tik apskaičiuoti mažų dimensijų problemas bei iliustruoti intuityvius pavyzdžius. Augant tiek veiksmų, tiek būsenų erdvei šie skaičiavimai tampa neįmanomi. Ši problema plėtojama kitame skyriuje.



15 pav. Dinaminio programavimo pavyzdys

1.6.8. Monte Karlo metodai

Nors specifinius Monte Carlo metodus šiame darbe nenagrinėsiu, tačiau pabrėšiu jų labai svarbų principą, kuriuo visi naujausi metodai remiasi. Monte Carlo metodais mes simuliuojame agento epizodus, gauname veiksmų trajektorijas (kaip tai matosi 12 paveikslėlyje) iš kurių susiskaičiuojame $q_\pi(s, a)$ reikšmes. Kiekvienos iteracijos metu kiekvienai būsenai s bei jos specifiniui veiksmui a mes išsaugome gautą apdovanojimą G_t. Iš šių išsaugotų reikšmių mes susiskaičiuojame vidurkį ir jį priskiriame $q_\pi(s, a)$ nariui. Tai atlikus mes perskaičiuojame veiksmų taisyklių reikšmes, kurios pasirenka naujus optimalius veiksmus. Po daugybes iteracijų, $q_\pi(s, a)$ konverguoja ties specifine, optimalia reikšme. Šio metodo veikimo principai pateikti 2 pseudokode.

Algoritmas 2: Taisyklių bei verčių aproksimavimas Monte Carlo pirmo apsilankymo metodu

Algoritmo parametrai:

$\varepsilon > 0$

Duomenų priskirimas:

$\pi \leftarrow$ atsitiktinė ε-greedy veiksmų taisyklių pasiskirstymas

$Q(s, a) \in \mathbb{R}$ (atsitiktinis priskirimas), visiems $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Vertės $(s, a) \leftarrow$ tuščias sąrašas, visiems $s \in \mathcal{S}, a \in \mathcal{A}(s)$

for ∞ **do**

  Agentas sugeneruoja episodą sekant $\pi : S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  for $t = T-1, T-2, \ldots, 0$ **do**

    $G \leftarrow \gamma G + R_{t+1}$

    if *Jei* S_t, A_t *būsena yra pirmas apsilankymas* $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$ *sekoje* **then**

      Išsaugoti G į Vertės(S_t, A_t) sąraša

      $Q(S_t, A_t) \leftarrow$ vidurkis (Vertės(S_t, A_t))

      $A^* \leftarrow \arg\max_a Q(S_t, a)$

    for *visi* $a \in \mathcal{A}(S_t)$ **do**

      $$\pi(a \mid S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon / |\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon / |\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Rezultatas: Gautos optimalios $Q(s, a)$ bei $\pi(a|s)$ reikšmės.

2 Algoritmo problema, kaip ir prieš tai buvusių metododų, yra kiekvienos būsenos ir jo veiksmo saugojimas bei ilgas konvergavimas. Augant būsenų bei veiksmų dimensijoms, konvergavimas tampa neįgyvendinama užduotimi. Veiksmams bei dimensijoms užtenka būti trečioje dimensijoje ir jau nebeužtenka kompiuterio resursų apskaičiuoti optimalių reikšmių. Taip pat monte karlo metodai pasižymi duomenų kaupimo problema, kuomet neužtenka

atminties juo saugoti. Duomenų kaupimą galima sumažinti pasitelkiant 15 bei 18 formulėmis, kaip tai atlieka pagrindinės šio mokslo formulės - TD ir Q-mokymasis.

1.6.9. TD bei Q-mokymasis

TD formulė sukombinuoja Monte Karlo bei dinaminio programavimo pagrindines idėjas į vieną paprastą 29 formulę.

$$\begin{aligned} V(S_t) \leftarrow V(S_t) + \alpha\left[G_t - V(S_t)\right] = \\ = V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \end{aligned} \tag{29}$$

Šios formulės ištakas galima pastebėti tiek 15 bei 18 formulėse. Šias dvi formules sukombinavus, galime perrašyti 26 formulę, kuri pasitelkia markovo grandinių modeliu, į 29 formulę, be markovo grandinių modelio. TD paklaida yra paklaida tarp $R_{t+1} + \gamma V(S_{t+1})$ bei $V(S_t)$. Kuomet $V(S_t)$ konverguoja, TD paklaida tampa nulinė. Pasiekus nulinę TD paklaidą galima laikyti, kad modelis apsimokė. Vienas iš pagrindinių šio modelio skirtumu lyginant su 2 algoritmu, yra greitesnė konvergacija. Šis metodas nelaukia epizodo galo, o atlieka $v_\pi(s)$ verčių atnaujinimą kiekvieno agento žingsnio metu. TD metodas yra pagrindas SARSA bei Q-mokymosi metodams, kurie atlieka atnaujinimus $q_\pi(s,a)$ vertėms. Šios dvi metodologijos skiriasi savo optimalumo atradimu, SARSA naudojasi šia TD atnaujinimo formule:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]. \tag{30}$$

o Q-mokymasis šia:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right] \tag{31}$$

Nors šios formulės yra vos ne identiškos, tačiau jos skiriasi skirtingomis optimaliomis veiksmų trajektorijomis. SARSA modelio atveju agentas stengesi elgtis saugiai, dažniau vengia būsenų, kurių vertė yra neigiama. Šis modelis tinkamas tokiems eksperimentams, kuriuos yra brangu simuliuoti ir norime sumažinti agento kritines klaidas besimokant, vedančias ties agento fiziniu sugedimu, praradimu. Šio atvejo pavyzdys galėtų būti roboto navigacija ties šlaitu. Kiekvienos simuliacijos metu agentui besimokant jis gali nukristi nuo šlaito ir sugadinti savo įranga. Todėl norint šias brangias klaidas sumažinti mes apmokome robota pasitelkiant SARSA metodu. Bet kadangi dydžiaja laiko dalį mokymasis vyksta simuliuotose aplinkose, mūsų agento kritinių klaidų kaina nėra faktorius. Svarbiausias tikslas mums tampa optimalios verčių suradimas, kuo Q-mokymasis ir pasižymi. Q-mokymasis [2] yra vienas iš populiariausių bei seniausių egzistuojančių metodu. Tačiau jis išpopuliarėjo visai neseniai, kuomet 2014 metais Google Deepmind komanda sugebėjo 31 funkcija aproksimuoti pasitelkiant konvoliucinius neuroninius tinklus. Šiomis aproksimacijomis Deepmind komanda apmokė agentus įveikti Atari žaidimus taip pademonstruojant skatinamojo

mokymosi bei neuroninių tinklų aproksimacijų potencialą. Visi dabartiniai modeliai pasitelkia neuroninių tinklų aproksimacijomis. Aproksimacijos padėjo išspręsti viena dydžiausių skatinamojo mokslo problemų - aukštų būsenų dimensijų neįmanoma konvergavimą. Dėl šios priežasties toliau bus aptariamas gilusisQ - vienas iš pirmųjų ir nors jau nebeefektyviu modeliu, sukėlusių skatinamojo mokslo atgimimą.

1.6.10. Gilusis skatinamasis mokslas, gilusis-Q

Visi naujausi, dabartiniai skatinamojo mokslo modeliai yra aproksimuojami neuroniniais tinklais. Neuroniniai tinklai padeda generalizuoti nepaprastai didžiules būsenų erdves, kaip tai pademonstravo Google Deepmind giliojo-Q konvoliucinis neuroninis tinklas, kurio įvestis buvo kompiuterio ekrano individualus pikseliai.

Gilusis-Q metodas parametrizuoja $q_\pi(s, a, \theta)$ reikšmę. Jo θ parametrai tampa neuroninio tinklo θ parametrai, kurie yra gradientiniais metodais optimizuojami, su tikslu pasiekti tikrąsias optimalias $q_\pi(s, a)$ reikšmes. Gilusis-Q neuroninio tinklo išvestis yra aproksimuota $q_\pi(s, a)$ reikšmė, kuri yra pateikta Q-mokymosi 31 formulėje šiuo $[R + \gamma \max_a Q(S', a)]$ pavidalu. Kaip matome, mus domina tik pateiktos funkcijos tikslo dalis, kaip tai išreikšta 17 formulėje. Šis tikslas yra traktuojamas, kaip naujausia $q_\pi(s, a)$ reikšmė. Dažnai norint tai pabrėžti Q-mokymosi formulė yra pertvarkoma į 32 formulėje pateiktą bellmano pavidalą:

$$Q^{\text{naujas}}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{Sena reikšmė}} + \alpha \cdot \underbrace{(r_t + \gamma \cdot \max_a Q(s_{t+1}, a))}_{\text{Tikslas = nauja } q_\pi(s,a) \text{ reikšmė}} \tag{32}$$

Šis pavidalas yra nekaskitas, o daugybe kartu nagrinėta judančio vidurkio formulė, kur $[R + \gamma \max_a Q(S', a)]$ yra naujausia gauta $q_\pi(s, a)$ reikšmė.

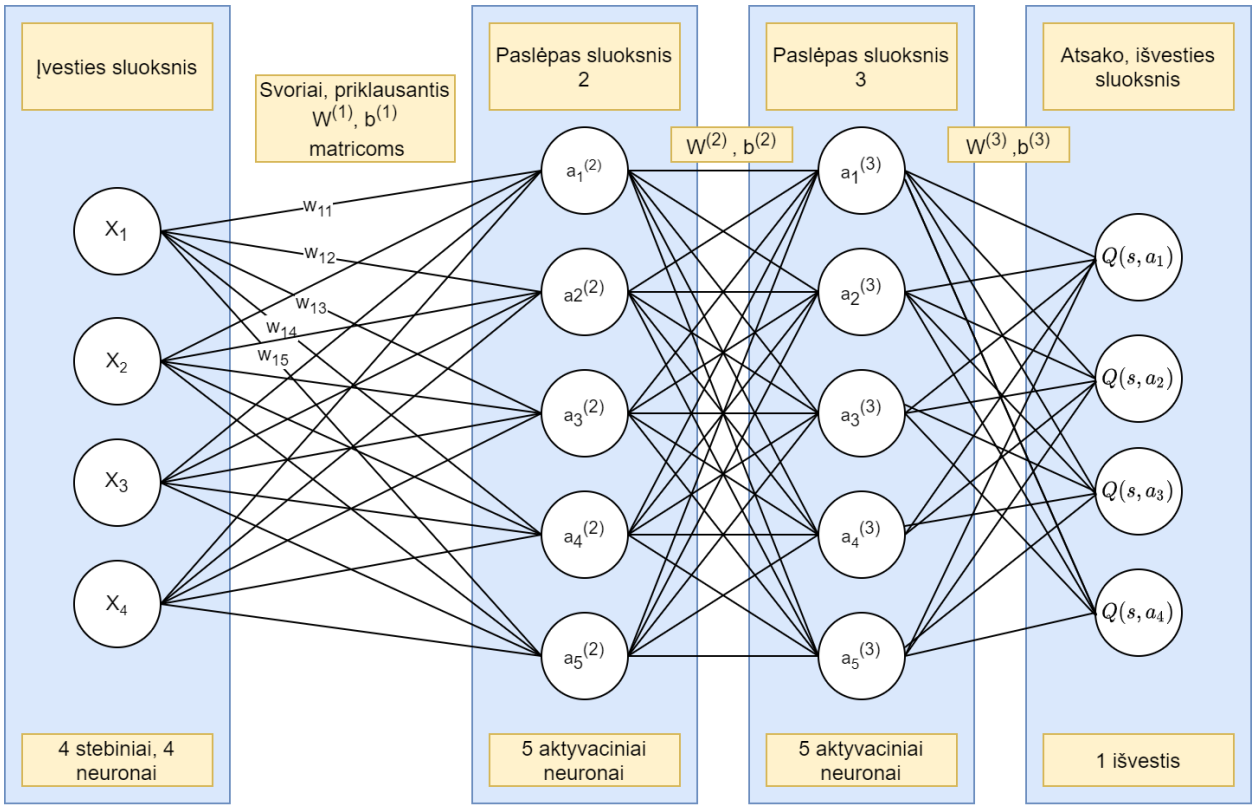
Taigi neuroninis tinklas stengesi aproksimuoti tikslą, kuris geriau žinomas kaip G_t arba naujausias aproksimuotas $q_\pi(s, a)$ narys. Tuo tarpu neuroninio tinklo įvestis yra būsenos vektorius s. Kadangi naudojamas neuroninis tinklas, vektoriaus s dydis nėra svarbus, konvergacija išlieka sparti.

Optimizavimo procesas susideda iš šių tryju dalių:

1. Sukuriama duomenų talpykla bei du neuroniniai tinklai - taikinys bei pagrindinis.
2. Pasirenkame veiksmą vadovaujantis godžiuoju ϵ veiksmų taisyklių metodu
3. Svorių atnaujinimas naudojantis bellmano taisykle, pateikta 32

Patirčių talpyklos bei dviejų neuroninių tinklų sukurimas Agentas tyrinėdamas aplinką, kaupia duomenis patirčių talpykloje, dar kitaip vadinama buferiu. Ši patirčių talpykla yra agento praeities patirtys, kurios padeda optimizuoti neuroninius tinklus. Joje kaupiamos (s,a,r,s') duomenų eilutės, kurios vos ne identiškos SARSA modelio duomenims. Panašumas ties tuo nesibaige, modelis taip pat kaupia duomenis tokiu pačiu principu kaip SARSA modelis, monte karlo budu simuliuojant žingsnius. Esant būsenoje s pagrindinis

neuroninis tinklas aproksimuoja s būsenos visų a veiksmų $q_\pi(s,a)$ vertes, kaip tai matyti 16 paveikslėlyje ir pasirenka judėti ta trajektorija, kurios $q_\pi(s,a)$ vertė yra dydžiausia.



16 pav. Giliojo Q neuroninio tinklo struktūra

Atlikus veiksmą mes apskaičiuojame gautą apdovanojimą r bei naują buseną s', kurioje atsiradome atlikus veiksmą a. Tokiu budu turim vieną duomenų eilutę, kurią išsaugome į patirčių talpyklą. Sukaupus pakankamai daug duomenų galime juos pradėti naudoti neuroninio tinklo apmokymui. Apmokant mes pasitelkiame tik mažą dalį duomenų atsitiktinai ištrauktų iš talpyklos. Svarbu neuroninį tinklą apmokyti tik su duomenimis iš patirčių talpyklos, norint išvengti koreliacijos. Šią korelecijos tendencija lengva pastebėti intuityviai išanalizavus patyrčių talpyklos veikimo principus:

1. Atsitiktinės duomenų eilutės (s,a,r,s') paimamos iš patirčių talpyklos. Dėl atsitiktinio ėmimo, dydžioji dalis patirčių - duomenų eilučių, yra iš skirtingų trajektorijų, epizodų.
2. Su paimtais duomenimis apmokome modelį. Šio proceso metu, dėl atsitiktinio ėmimo, skirtingos būsenos įgyją naujas $q_\pi(s,a)$ aproksimacijas.

Tuo tarpu be patirčių talpyklos, mes apmokytume neuroninį tinklą kiekvieno žingsnio metu. Tačiau kadangi nauji agento žingsniai priklauso nuo senesnių, mūsų duomenis taptų priklausomi. Tai pažeidžia stochastinio gradientinio nuolydžio viena iš pagrindinių reikalavimų - imties duomenis turi būti nepriklausomi. Tai nutikus modelis pradėtų dažniau rinktis pasikartojančias trajektorijas, diverguotų į ne optimalias trajektorijas. Patirties talpykla taip pat turi dar viena papildoma svarbų bruožą - pakartotinas apmokymas su tais pačiais

duomenimis. Kadangi duomenis yra išsaugojami talpykloje, mes turime ne mažą tikimybę apmokyti neuroninį tinklą su tuo pačiu dėmeniu ne vieną kartą. Šis bruožas naudingas apmokyti neuroninį tinklą su retai pasitaikančiomis reikšmėmis. Patirčių talpykla dažniausiai turi fiksuotą dydį. Talpykla yra naudojama visuose algoritmuose, kurie naudoja neuroninių tinklų aproksimacijas.

Patirčių talpykla padeda apsimokyti pagrindiniam neuroniniui tinklui, kurio tikslo funkcija yra pateikta žemiau:

$$L(\theta) = \left[\left(\left(r + \gamma \max_{at+1} Q\left(s_{t+1}, a_{t+1}; \theta^{\text{taikinys}}\right)\right) - Q\left(s, a; \theta^{\text{pagrindinis}}\right)\right)^2\right] \tag{33}$$

Kaip matome θ parametrai atkeliauja iš taikinio bei pagrindinio neuroninio tinklo. Abu neuroniniai tinklai aproksimuoja $q_\pi(s, a)$. Tačiau taikinio neuroninio tinklo parametrai nėra apmokami, jie kas tam tikra agento žingsnių kieki yra periodiškai paimami iš pagrindinio neuroninio tinklo. Tuo tarpu pagrindinio neuroninio tinklo parametrai yra atnaujinami kas kiekviena žingsnį. Ši metodologija buvo pirma karta pritaikyta Google Deepmind komandos, kuriant Atari agentus [4]. Jos tikslas yra sumažinti mokymosi divergacijas bei nestabilias $q_\pi(s, a)$ verčių svyravimus. Šios nepalankios savybės atsiranda iš neuroninių tinklų generalizacijos. Atnaujinus $q_\pi(s, a)$ reikšmes ir pakeitus θ parametrus, kartu pasikeičia kaimyninės $q_\pi(s', a)$ reikšmės, kitaip tariant mūsų taikinio reikšmės, jei naudotume viena neuroninį tinklą. Nuolat kintančios taikinio reikšmės veda prie nestabilaus konvergavimo, mūsų $q_\pi(s, a)$ reikšmės artėja link norimų $q_\pi(s, a)$ reikšmių, kurios dėl kiekvieno atnaujinimo kinta. Todėl yra sukuriamas antras neuroninis tinklas su fiksuotais parametrais, kurie stabilizuoja mokymosi procesą.

Veiksmų pasirinkimas Gilusis Q naudoja goduji ϵ veiksmų pasirinkimui, kur su $1 - \epsilon$ tikimybe pasirinksime veiksmą vedanti į dydžiausią $q_\pi(s, a)$. Šiuos $q_\pi(s, a)$ vertes apskaičiuoja neuroninis tinklas, kaip tai pateikta 16 paveikslėlis. Tačiau su ϵ tikimybe mes pasirinksime neoptimalę vertę. Tai padeda aplankyti naujas trajektorijas, tikintis surasti optimalesnius veiksmus.

Svorių atnaujinimas Atlikus veiksmo pasirinkimą bei išsaugojus duomenis susyjusius su šiuo veiksmu mes įgyvendiname viena agento žingsnį. Po kiekvieno žingsnio mes apmokome agenta tiksliau generalizuoti $q_\pi(s, a)$ reikšmes. Mokymo procese pasitelkiame patirčių talpyklą, kaip tai jau buvo aptarta 1.6.10 skyriuje. Mokymosi procesas gali trukti pora dienų, priklausomai nuo nagrinėjamos aplinkos sunkumo.

Nors Q-mokymasi procesas pademonstravo skatinamojo mokslo potencialą, šis metodas yra nebetaikomas. Tačiau svarbios pademonstruotos idėjos kaip $q_\pi(s, a)$ verčių aproksimavimas neuroniniais tinklais, patirčių talpykla, dviejų neuroninių tinklų stabilizavimas yra pamatas aktoriaus-kritiko šablonui, kuriuo vadovaujasi visi moderniausi modeliai. Norint

suprasti aktoriaus-kritiko šabloną, pradžioje reik įsigilinti į vieną svarbiausių skatinamojo mokslo teoremų - veiksmo taisyklių gradientas.

1.6.11. Veiksmo taisyklių gradientas bei aktorius-kritikas

Veiksmo taisyklių gradientas yra viena svarbiausių teoremų skatinamajame moksle. Ilga laika buvo manoma, kad veiksmo taisyklių gradientas neegzistuoja, todėl buvo taikomi Giliojo-Q mokymosi algoritmai, kurie optimizuoja $q_\pi(s, a)$ vertes ir neliečia veiksmo taisyklių optimizavimo. Tačiau kaip matome 14 paveikslėlyje, optimizavimo procesas įtraukia nuolatinį verčių bei veiksmo taisyklių iteravimą. Atradus veiksmo taisyklių gradientą, pagaliau tapo įmanoma įgyvendinti šį ciklą ir tai pasiekus gimė daugybė galingu modelių kaip PPO, DDPG, TD3, SAC.

Veiksmo taisyklių tikslas yra skatinti tuos veiksmus, kurie vedė ties veiksmų trajektorijomis suteikenčiomis aukštus apdovanojimus ir slopinti veiksmus, kurių apdovanojimų trajektorijos yra mažos. Formaliai tai mes užrašome šiuo pavidalu, nurodančiu tikėtiną veiksmų taisyklių apdovanojimą:

$$J\left(\pi_\theta\right) = \underset{\tau \sim \pi_\theta}{\mathrm{E}}[R(\tau)], \tag{34}$$

kur $R(\tau)$ yra gautų epizodo, trukusio T žingsnių, apdovanojimų suma:

$$R(\tau) = \sum_{t=0}^{T} R_t \tag{35}$$

Tikėtina veiksmo taisyklių apdovanojimą mes išreiškiame per tikimybę įgyti šiuos apdovanojimus:

$$P(\tau \mid \theta) = \rho_0\left(s_0\right) \prod_{t=0}^{T} P\left(s_{t+1} \mid s_t, a_t\right) \pi_\theta\left(a_t \mid s_t\right) \tag{36}$$

$P(\tau \mid \theta)$ yra trajektorijos tikimybė, kur $\rho_0\left(s_0\right)$ yra tikimybė, kad pradėsime s_0 būsenoje. Šia tikimybe sudauginus su apdovanojimų kiekiu, gauname galutinę $J(\pi)$ išraišką:

$$J(\pi) = \int_\tau P(\tau \mid \pi) R(\tau) = \underset{\tau \sim \pi}{\mathrm{E}}[R(\tau)] \tag{37}$$

Mūsų tikslas yra optimizuoti veiksmų taisyklių $J(\pi)$ parametrus taip, kad jie gražintu kuo didesnį epizodų apdovanojimus:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J\left(\pi_\theta\right)\big|_{\theta_k} \tag{38}$$

Deja mes negalime optimizuoti $J(\pi)$ tiesiogiai, kadangi neturint modelio, mes negalime gauti 36 formulės reikšmes, reikalingas apskaičiuoti gradientą. Todėl imamės analitinių išraiškų, padedančių pakeisti gradiento formulę į naują formulę, kurios rezultatas nors ir nesutampa su $J(\pi)$ bet sutampa jo gradientas. Šios formulės išvedimui buvo pasitelkti šie faktai:

1. Log išvestinė:
$$\nabla_\theta P(\tau \mid \theta) = P(\tau \mid \theta) \nabla_\theta \log P(\tau \mid \theta) \tag{39}$$
2. Log trajektorijos tikimybė:
$$\log P(\tau \mid \theta) = \log \rho_0 (s_0) + \sum_{t=0}^{T} \left(\log P(s_{t+1} \mid s_t, a_t) + \log \pi_\theta (a_t \mid s_t) \right) \tag{40}$$
3. Apskaičiuojant gradientinį nuolydį, funkcijos, kurios nepriklauso nuo θ, pranyksta. Kitaip tariant $\rho_0 (s_0), P(s_{t+1} \mid s_t, a_t)$, ir $R(\tau)$ yra lygus nuliui.
4. Šios pranykusios funkcijos supaprastina log trajektorijos tikimybe:
$$\begin{aligned} \nabla_\theta \log P(\tau \mid \theta) &= \nabla_\theta \log_0 (s_0) + \sum_{t=0}^{T} \left(\nabla_\theta \log P(s_{t+1} \mid s_t, a_t) + \nabla_\theta \log \pi_\theta (a_t \mid s_t) \right) \\ &= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t) \end{aligned} \tag{41}$$

Pasitelkus šiuos išvedimus galime išsivesti veiksmo taisyklių gradientą:

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \nabla_{\theta_{\tau \sim \pi_\theta}} \underset{\tau \sim \pi_\theta}{\mathrm{E}} [R(\tau)] \\ &= \nabla_\theta \textstyle\int_\tau P(\tau \mid \theta) R(\tau) \\ &= \textstyle\int_\tau \nabla_\theta P(\tau \mid \theta) R(\tau) && \text{Log išvestinės faktas} \\ &= \textstyle\int_\tau P(\tau \mid \theta) \nabla_\theta \log P(\tau \mid \theta) R(\tau) \\ &= \underset{\tau \sim \pi_\theta}{\mathrm{E}} [\nabla_\theta \log P(\tau \mid \theta) R(\tau)] \\ \therefore \nabla_\theta J(\pi_\theta) &= \underset{\tau \sim \pi_\theta}{\mathrm{E}} \left[\textstyle\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t) R(\tau) \right] && \text{4 fakto pritaikymas} \end{aligned} \tag{42}$$

Turint šią formą mes stochastiniu budu galime pradėti aproksimuoti gradientinį pakilimą. Šio proceso metu mes generuojame naujas imtis bei apskaičiuojame imties vidurkį:

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t \mid s_t) R(\tau), \tag{43}$$

kur $\hat{g}$ yra mūsų gradiento vektorius, o $\mathcal{D}$ yra trajektorijų (imčių) kiekis. Tokiu budu mes galime labai tiksliai aproksimuoti mūsų veiksmo taisyklių parametrus, vedančius ties dydžiausiais apdovanojimais.

Pats metodas savaime nėra efektyvus tačiau sukombinavus su Giliuoju-Q mokymusi mes grįžtame į pagrindinį 14 metodą, kuris buvo taikomas be aproksimacijų. Šį procesą įgyvendina aktoriaus kritiko metodas. Jo pavadinimas kilo iš intuityvios įdėjos, kuria galime pastebėti 43 formulėje. Jei $R(\tau)$ yra teigiamas, pateiktas veiksmas paskatinamas, jei $R(\tau)$ neigiamas, pateiktas veiksmas slopinamas. $R(\tau)$ atlieka kritiko rolę, skatina arba slopina veiksmų taisyklių neuroninio tinklo, vadinamo aktoriumi, pateiktas reikšmes. Tačiau aktoriaus-kritiko metoduose mes vietoj tiesioginės $R(\tau)$ reikšmės, pasitelkiame giliojo-Q neuroninį tinklą kurio $q_\pi(s, a)$ reikšmės aproksimuoja $R(\tau)$. Taigi gilusis-Q pateikia $q_\pi(s, a)$ bei jas optimizuoja, kaip tai jau matėme 1.6.10 skyriuje, ir naudojantis $q_\pi(s, a)$ reikšmėmis mes kartu atnaujiname veiksmo taisyklių neuroninį tinklą naudojantis 43 formule. Šis metodas bei jo pseudokodas bus toliau plėtojamas tolimesniuose skyriuose, kurie remiasi jo struktūra.

1.6.12. DPG bei DDPG

DPG[5] metodas yra paremtas aktoriaus-kritiko metodu. Jis sukombinuoja Gilusis-Q metodą su veiksmo taisyklių gradientu. Tačiau šiuo atveju veiksmo taisyklių tikslo formulė įgauna skirtingą pavidalą:

$$J(\theta) = \int_{\mathcal{S}} \rho^{\mu}(s) Q\left(s, \mu_{\theta}(s)\right) ds, \tag{44}$$

kur $\mu_{\theta}(s)$ yra deterministinė versija $\pi_{\theta}(a|s)$ versija. Kitaip tariant veiksmai parenkami ne iš veiksmų tikimybių pasiskirstymo, o optimaliausias veiksmas yra iškarto funkcijos aproksimuojamas - $a = \mu(s)$. Veiksmų aproksimacijos padeda praplėsti veiksmų dimensiją iš keleto pasirinkimų, kaip tai matemė 15 pavyzdyje, turint tik keturis galimus veiksmus, į begale. Vienas iš praktinių šio metodo bruožų yra jo lengvas gradiento apskaičiavimas:

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \int_{\mathcal{S}} \rho^{\mu}(s) \nabla_a Q^{\mu}(s,a) \nabla_{\theta} \mu_{\theta}(s) \Big|_{a=\mu_{\theta}(s)} ds \\ &= \mathbb{E}_{s\sim\rho^{\mu}} \left[\nabla_a Q^{\mu}(s,a) \nabla_{\theta} \mu_{\theta}(s) |_{a=\mu_{\theta}(s)} \right] \end{aligned} \tag{45}$$

Telieka simuliuoti agento žingsnius ir apskaičiuoti veiksmo taisyklių gradientą, kaip tai matėme 43 formulėje. Praktikoje gradiento apskaičiavimas yra keblesnis, nei 42 metodo, kadangi reikia apskaičiuoti dviejų neuroninių tinklų gradientus bei juos sudauginti. Šis procesas bus giliau aptariamas praktikoje.

Šio modelio dydžiausias trūkumas yra neegzistuojantis agento aplinkos tirinėjimas. Modelis visados renkasi optimalias veiksmų taisykles, netirinėja aplinkos. DDPG[6] modelis pagerina DPG modelį pridėdamas triukšmą prie veiksmų taisyklių pateiktų reikšmių:

$$\mu'(s) = \mu_{\theta}(s) + \mathcal{N}, \tag{46}$$

kur $\mathcal{N}$ yra normalusis pasiskirstymas. Jo parametrai yra pasirenkami priklausomai nuo nagrinėjamos problemos.

1.6.13. TD3

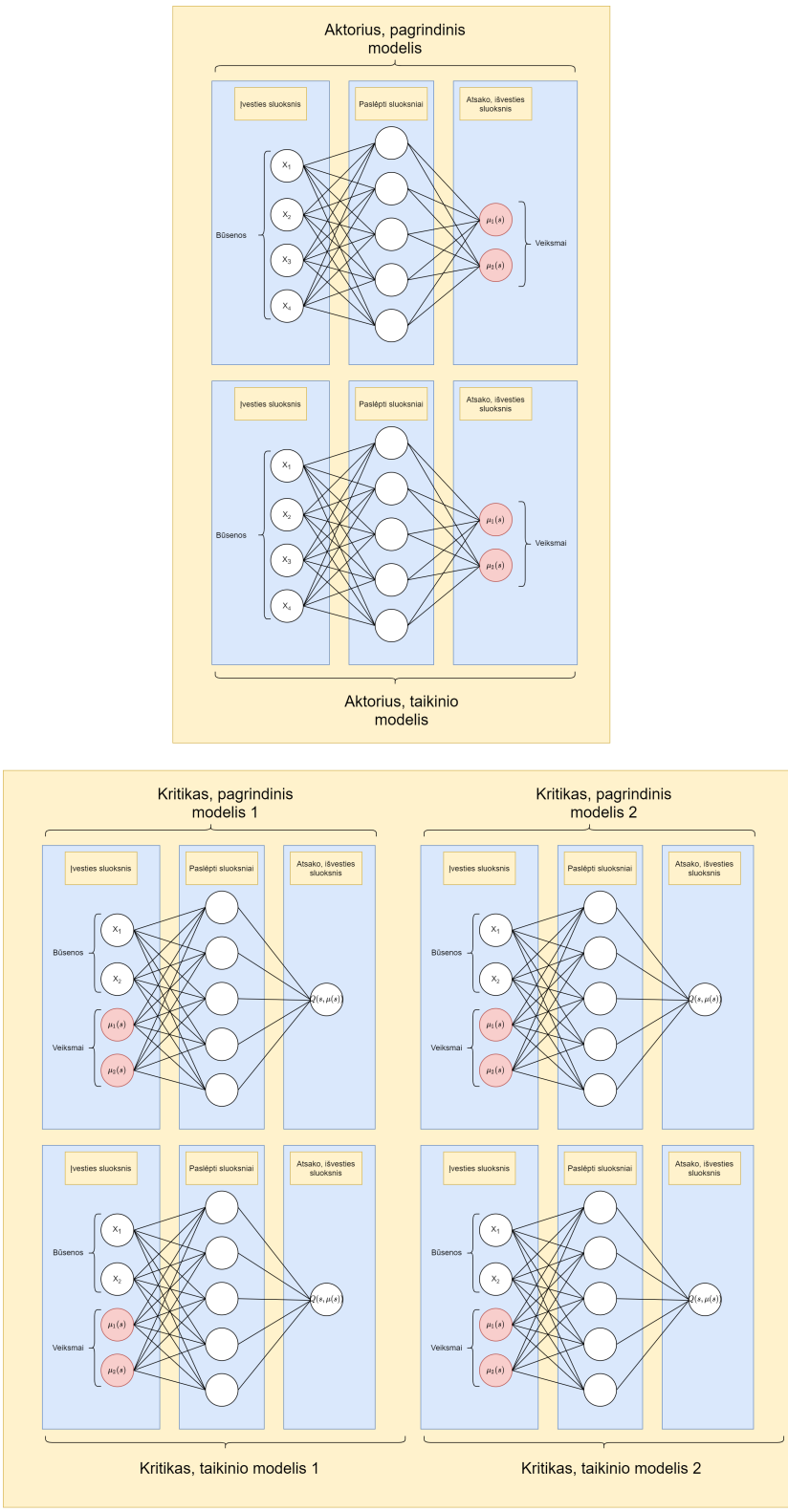
TD3 modelis yra vos ne identiškas DDPG modeliui tačiau turi pora skirtumu:

1. Du gilieji-Q neuroniniai tinklai. Kadangi gilusis-Q neuroninis tinklas dažnai pervertina reikšmes, mes apmokymui naudojame du giliuosius-Q neuroninius tinklus ir apsimokome su mažesne gauta reikšme.
2. Vėluojantis veiksmo taisyklių neuroninio parametrų atnaujinimas. Jei DPG atnaujindavo neuroninį tinklą kas kiekvieną agento žingsnį TD3 autorius rekomenduoja atnaujinimą atlikti kas du žingsnius.
3. Reguliarizacija, pateikta 47 formulėje. Ji padeda išvengti persimokymo, kuomet veiksmų taisyklės per greit prisitaiko prie aukštų gilaus-Q reikšmių. Pagrindinė idėja yra

pridėti maža triukšmą prie veiksmo taisyklių neuroninio tinklo pateiktos reikšmės. Tokiu budu neuroninis tinklas bus apmokytas geriau aproksimuoti kaimynines veiksmų reikšmes.

$$a'\left(s'\right) = \text{clip}\left(\mu_{\theta_{\text{taikinio}}}\ \left(s'\right) + \text{clip}(\epsilon, -c, c), a_{\min}\ , a_{max}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma) \tag{47}$$

TD3 struktūra susideda iš šešių neuroninių tinklų, kaip tai matosi 17 paveikslėlyje. Jie reikalingi tiktais apmokymo metu. Apmokius neuroninį tinklą užtenka vieno veiksmų taisyklių neuroninio tinklo, kuris pateikinėtu optimalius veiksmus agentui.



17 pav. TD3 modelio struktūra

2. Praktinė dalis

Praktinėje dalyje bus įgyvendinmas TD3 metodas. Jo įgyvendinimui bus pasitelkta unity aplinka, kurioje egzistuos apmokamas agentas. Agentas yra aštuonių galūnių voras, kurio tikslas yra nuropoti į užsibrėžtą poziciją, judinant savo galunes. Šis agento tikslas bei aplinka bus plėtojamaa pirmuosiuose praktinės dalies skyriuose. Toliau bus gilinamasi ties komunikacijos sluoksniu, kuriame simuliacijos aplinka bei mokymosi metodas TD3 keičiasi duomenimis. Kadangi TD3 metodas bus įgyvendintas python aplinkoje, o aplinkos simuliacija unity aplinkoje, šios dvi aplinkos negali vienas kitai persiųsti bei gauti duomenis. Todėl yra būtinas komunikacijos sluoksnis tarp šių dviejų aplinkų, padedančiu išsiųsti bei gauti žinutes. Galiausiai bus detaliai aptartas bei įgyvendinamas TD3 metodas.

2.1. Agentas, jo tikslai bei aplinka

Per pastaraji dešimtmeti skatinamieji metodai buvo pradėti taikyti įvairiuose srityse, tačiau viena iš sričių pasižymejųsių dydžiausiu proveržiu - robotų judėjimo kontroleriai. Roboto judėjimo užduotis tradiciniais metodais buvo ne vieną kartą stengiamasi išspręsti, bet rezultatai buvo nepatenkinami. Tačiau skatinamojo mokslo metodai pademonstravo, kad jie yra puikiai pritaikyti kontroliuoti agento galunes, surasti optimalias judėjimo trajektorijas vedančias į užsibrėžto tikslo pasiekimą. Butent šią užduoti ir įgyvendinsiu praktinėje dalyje. Bus pasitelktas unity aplinkoje sukurtas voras, kurio tikslas nuropoti iki specifinės kordinatės, kaip tai matome 8 paveikslėlyje. Pagrindinis užduoties sunkumas kyla iš fizikinio judėjimo. Labai dažnai judėjimo agentai unity aplinkoje nejuda pagal fizikinius dėsnius, o paprasčiausiai animacijų pagalba jų judesiai yra įrašyti ir atkartojami. Tokios animacijos nėra paremtos fizika, kiekviena agento galūnė kiekvieno kadro metu įgyja įrašytas pozicijų kordinates. Toks judėjimas neprisitaiko prie kintančios, dinaminės aplinkos ir roboto galūnės dažnai kerta objektus kiaurai. Šias problemas išsprendžia fizika paremtas galūnių judidinimas, kurio programiškai iki šiol įgyvendinti buvo neįmanoma. Tik neseniai pasitelkus statistiniais mokymosi metodais kaip TD3 buvo pasiektas fizikinis agentų judėjimas.

Taigi kiekvieno epizodo metu voras yra sukuriamas aukštai ore. Jis turi sugebėti nusileisti ant savo galūnių bei nuropoti ties žaliu kubeliu. Sėkmingai jį pasiekus epizodas nesibaige, yra sukuriamas naujas žalias kubelis, kurį voras vėl turi pasiekti. Šis budas skatina vorą aplankyti naujas verčių būsenas susijusias su voro apsisukinėjimu. Epizodas baigiasi kuomet pasiekeme epizodo maksimalų žingsnį, kurį galime traktuoti kaip laiko intervalą. Taip pat epizodas gali baigtis ankčiau, jei yra pasiekta terminuojanti būsena. Voro aplinkos atveju tai nutinka jei pagrindinė kūno dalis arba viršutiniai sanariai pasiekę žemę. Kuo ilgiau voras išgyveną bei kuo arčiau jis yra žalio kubelio, tuo agentas gauna didesnį epizodo kaupiamąjį apdovanojimą. Tai ir yra agento pagrindinis tikslas, gauti kuo didesnį apdovanojimą.

2.2. Mokymosi seka

Voro agento mokymosi procesas yra identiškas aptartiems mokymosi metodams ir sudarytas iš šių iteracinių žingsnių:

1. TD3 modelis turi dabartines voro būsenas. Turint šias būsenas TD3 pateikia naujus veiksmus.
2. Voras gauną TD3 modelio pateiktą veiksmą bei jį atlieką.
3. Atlikus veiksmą gaunama nauja būsena bei apdovanojimas. Šios vertės nusiunčiamos TD3 modeliui.
4. TD3 gavęs pateikto veiksmo rezultatą - naują būsena bei apdovanojimą, pradeda modelio apsimokymą. Tai atlikus naujos būsenos tampa dabartinėmis ir pradedama naują iteracija. Dažniausiai apsimokymas būna atidedamas ir jis įvyksta pasibaigus agento epizodui arba paralelizuotams agentams atlikus tam tikrą kiekį žingsnių.

Svarbu pastebėti, kad apdovanojimų bei būsenų duomenys nusiunčiami TD3 modeliui ne kiekvieno simuliacijos kadro metu, o tik kas penkis unity fizikos simuliacijos žingsnius. Kitaip tariant agentui atlikus veiksmus jis laukia penkis unity fizikos atnaujinimus, per kuriuos jo galūnės pakeičią poziciją penkis kartus po labai maža atstumą. Tai atlikus agentas tik tada užfiksuoja naują įgytą būseną. Per šiuos penkis žingsnius jo apdovanojimas yra kaupiamas - kas žingsnį pridedame po apskaičiuota apdovanojimą. Šis penkių kadrų atnaujinimo atidėjimas padeda pasiekti ryškesni naujo veiksmo ir jo pasekmės ryšį. Taip pat tai sumažiną duomenų kiekį bei pagreitiną mokymą.

2.3. Aplinka

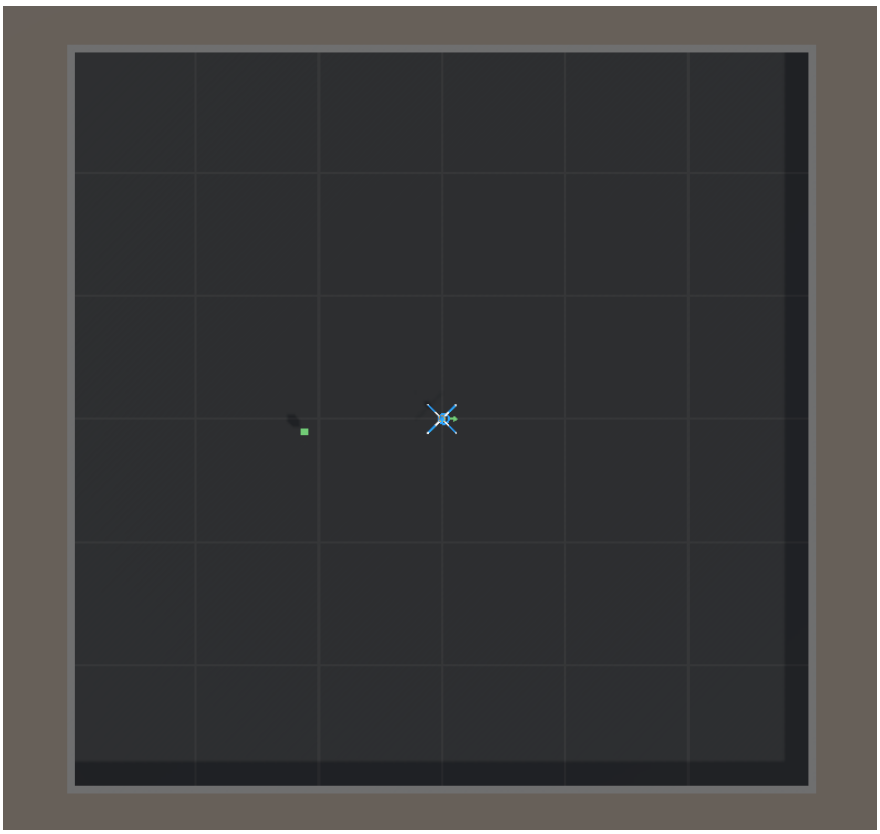
Voro aplinka susideda iš tryjų elementų:

1. Voras.
2. Tuščia, lygi arena, su keturiom, areną apsupančiom, sienom.
3. Žalias kubelis, iki kurio vorui reikia nuropoti.

Aplinkos atvaizdavima matome 18 paveikslėlyje. Joje agentas praleidžia visą savo mokymosi laiką. Aplinka bei pats fizikiniais dėsniais paremtas voras yra jau sukurtas unity programuotojų. Jie pasitelkė skaitinamojo mokslo PPO metodus apmokyti vorą ropoti bei pasiekė įspudingus rezultatus. Mano mokymo metu bus naudojamas TD3 metodas, kuris yra specifiškai pritaikytas deterministiniams veiksmams bei panašiuose užduotyse dažniau pasiekę optimalesnius rezultatus.

Arena įgyjo šį specifini išsidėstymą, dėl šių tolimesnių priežaščių

1. Arena lengva padvigubinti, sukurti daugybe jos kopijų. Tai naudinga paraleliam mokymuisi, kuris bus aptartas tolimesniuose skyriuose.



18 pav. Voro aplinka

2. Lygioje arenoje, be įdubimu bei iškilimu, yra lengviau apmokyti agentą. Nereikia programuoti sudėtingu metodu įdubimams aptikti bei informuoti agenta apie jų egzistavimą. Tai sumažina būsenų erdvę.
3. Žalias kubelis, iki kurio vorui reikia nuropoti. Tai yra paprasčiausias atskaitos taškas, naudojamas įvertinti agento optimalumą.
4. Keturios sienos, neleidžiančio vorui nukristi už simuliacijos ribų.

Taigi ši arena yra papračiausia įmanoma aplinka, kuri padeda įsitikinti mokymosi metodo efektyvumu. Jei pasirinktas mokymosi metodas sugeba susidoroti su šia aplinka, ją galima pradėti palaipsniui sunkinti bei pritaikyti savo reikmėms.

2.4. Duomenys

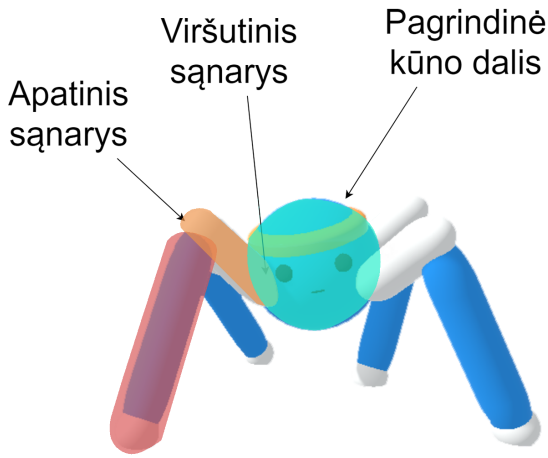
Kaip matome 17 paveikslėlyje, TD3 mokymosi metodų neuroninių tinklų įvestis yra būsenų erdvė bei aproksimuoti veiksmai. Tuo tarpu pagrindinė išvestis yra veiksmų taisyklės - kokius tolimesnius veiksmus turėtų atlikti agentas. Šias veiksmų reikšmes padeda apskaičiuoti apdovanojimai, taip pat viena iš įvesčių reikšmių, apskaičiuojanti taikinio reikšmes, kurias neuroninis tinklas tūrėtų aproksimuoti. Taigi toliau visi šie duomenys bus aptariami detaliau.

2.4.1. Įvestis - būsenos

Tiek aktoriaus modelis, aproksimuojantis veiksmų taisykles, tiek kritiko modelis, aproksimuojantis būsenų vertes, priima būsenų reikšmes. Voro būsenų reikšmės susideda iš 127 reikšmių. Jas galima suskaldyti į šias kategorijas:

1. Voro galūnės. Kiekviena voro koja susideda iš dviejų sąnarių: viršutinio bei apatinio sąnario.
2. Voro pagrindinis kūnas. Viršutiniai voro sąnariai susijungia su šia dalimi.

Kaip matomo 19 paveikslėlyje apatinis sąnarys kontroliuoja koją, užtušuota rausvai. Viršutinis sąnarys kontroliuoja koją užtušuota oranžine spalva. Ir galiausiai visi viršutiniai sąnariai susijungia į pagrindinę kūno dalį užtušuota žalsvu atspalviu.



19 pav. Voro galūnės.

Visos voro galūnės kaupia ne konkrečią informaciją, bet santykinę. Kitaip tariant nėra kaupiama tiksli voro pozicija, bet atstumo vektorius nuo voro iki taikinio, žalio kubelio. Tokiu budu voras neprisitaiko prie konkrečios aplinkos, agentas gali lengvai ropoti įvairiose arenose. Tai ypatingai svarbu paralelizuojant mokymosi procesą. Taigi visi sanariai kaupia šias būsenas:

1. Ar galūnė liečia žemės paviršių. Priklausomai nuo to ar liečia, simuliacija gali būti nutraukiama ir įvardijama kaip nepasisekusi. Ši sąlyga galioja viršutiniam sąnariui, kadangi nenorime, kad jis liestų žemę, jis nuolatos turi būti virš jos.
2. Atstumo vektorius tarp sąnario greičio vektoriaus bei agento-kubelio atstumo vektoriaus. Mes norime, kad atstumas būtų lygus nuliui, kitaip tariant abu vektoriai žiūri į tą pačia trajektoriją.
3. Laipsnių atstumas tarp sanario pozicijos vektoriaus bei agento-kubelio atstumo vektoriaus. Turi panašumu su praeitu punktu, tačiau duomenys išreikšti laipsniais.

Pagrindinė kūno dalis kaupia šią informaciją:

1. Ar liečia žemės paviršių. Jei taip, simuliacija yra nutraukiama.
2. Pagrindinės kūno dalies atstumas iki žemės paviršiaus. Kadangi agentas turi sugebėti stovėti su savo kojomis ant žemės paviršiaus, voras turi žinoti kaip arti jis yra iki žemės.
3. Atstumo bei laipsnių vektoriai, gaunami tokiu pat principu kaip sąnariuose.

Taip pat kaupiamas vidutinis agento galūnių greitis. Kuo didesnis greitis, tuo jis gauna didesnį apdovanojimą. Tokiu budu mes skatiname agentą jūdėti sparčiai.

Šios visos būsenos supakuojamos į vieną realiųjų skaičių masyvą ir nusiančiamos TD3 mokymosi modeliui. Taip pat nusiunčiame apdovanojimą, kuris yra apskaičiuojamas unity aplinkoje.

2.4.2. Apdovanojimas

Duomenų talpykloje kaupiamos ne tik būsenos bet ir apdovanojimai. Jie atkeliauja iš unity aplinkos. Vorui atlikus TD3 modelio pateiktus veiksmus, jis apskaičiuoja atliktų veiksmų naudą. Vadovaujantis 1.6.4 skyriaus svarbiu patarimu, mes turime suformuoti apdovanojimo funkciją taip, kad jį neformuotų užduoties sprendimo būdo, o tik apdovanotų už pasiektą tikslą. Todėl apdovanojimo funkcija susideda iš šių tryju funkcijų:

1. Skaliarinė sandauga tarp pagrindinės kūno dalies greičio vektoriaus bei normalizuoto krypties vektoriaus nuo pagrindinio kūno į žalią kubelį. Šie vektoriai pavaizduoti 9 paveikslėlyje, tačiau pateiktas žalias vektorius nėra vizualiai normalizuotas. Jei šie vektoriai sutampa, mes gauname vieneto vertę, žyminčią dydžiausią įmanomą apdovanojimą. Tokiu budu mes nurodome agentui judėti link žalio kubelio, bet nepasakome kaip konkrečiai atlikti šią užduoti.
2. Skaliarinė sandauga tarp agento krypties vektoriaus bei normalizuoto krypties vektoriaus nuo pagrindinio kūno į žalią kubelį. Kitaip tariant jei agentas yra atsisukęs į žalią kubelį, jis gauną maksimalų apdovanojimą - vienetą.
3. Apskaičiuojame ar vidutinis visų galūnių greitis atitinka mūsų pateiktą maksimalų greitį. Jei atitinką, priskiriame papildoma vieneto apdovanojimą, jei visai nesutampa, apdovanojimas tampa nulis. Tarpinės reikšmes pateike sigmoido funkcija. Ši apdovanojimo funkcija buvo pridėta vėliausiai iš visų apdovanojimo funkcijų, kuomet buvo pastebėtas voro lėtas slinkimas į žaliąjį kubelį. Agentas išmokęs stabiliai stovėti ant kojų ir iš lėto ropoti link žaliojo kūbelio po truputi pastebi, kad kuo greičiau juda, tuo didesnį pasiekę apdovanojimą.

Šie apdovanojimai, kiekvienos fizikos atnaujinimo metu yra apskaičiuojami bei sudedami į vieną kaupiamąjį apdovanojimą. Jis, kartu su būsenomis, nusiunčiamas TD3 modeliui. Nusiuntus apdovanojimą jis tampa nuliu ir jo skaičiavimo procesas prasideda iš naujo.

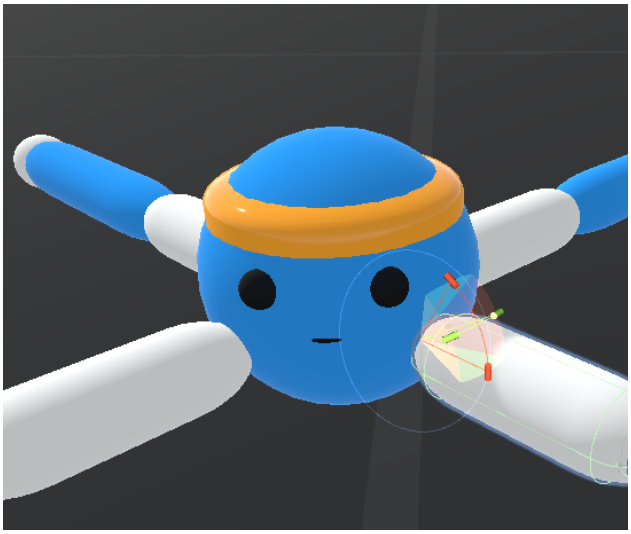
Kiekvieno epizodo metu mes kaupiame epizodo galutinį apdovanojimą. Jis judančio vidurkio metodu apskaičiuoja paskutinių penkiasdešimties epizodų apdovanojimo vidurkį. Šis kiekvieno epizodo metu kintantis matas yra pagrindinis kriterijus pagal kurį vizualiai vertiname modelio efektyvumą. Mokymuisi vykstant jis turi kilti. Tai parodo vis dydėjanti epizodų apdovanojimą.

2.4.3. Išvestis - veiksmai

TD3 aktoriaus neuroninis tinklas priėmes būsenas išveda veiksmus. Iš viso yra apskaičiuojami dvidešimt veikmų verčių, kuriuos voras gauną. Šios veiksmų vertės kontroliuoja voro viršutinęs bei apatinęs galunes. Toliau bus vardyjama galūnių veiksmų vertės:

1. Viršutinis voro sąnarys gali sukinėtis aplink x bei y kordinačių ašį, tačiau ne z. Veiksmų vertės nurodo galutinę sąnario poziciją laipsniais bei kiek jėgos bus suteikta šiai pozicijai pasiekti.
2. Apatinio sąnario veiksmų vertės yra identiškos, tačiau šis sąnarys gali sukiotis tik aplink x ašį.

Kadangi TD3 aktoriaus neuroninio tinklo išvestis yra tanh funkcija, mes nuolatos turime konvertuoti $[-1,1]$ intervalo reikšmes į konkrečias laipsnių bei jėgų reikšmes. Taigi agentas gauna naujas galūnių judėjimo reikšmes ir tokiu budu jis jas po truputi krutina.



20 pav. Viršutinio sąnario judėjimo trajektorijos

2.5. Komunikacijos sluoksnis

TD3 modelis bei voro aplinka yra įgyvendinta skirtinguose programuose. Voro aplinka yra įgyvendinta unity programoje, o TD3 modelis pythono aplinkoje. Kadangi unity variklis negali kompiliuoti python kodo, teko įgyvendinti šias dvi aplinkas atskirose programose. Tai

sukėlė pastovių kėblumu, kadangi duomenys tarp šių dviejų programų turi nuolatos bei užtikrintai keistis, kas dažnai neįvygdavo. Taigi buvo sukurtas komunikacinis sluoksnis, kuris užtikrindavo pastovų žinučių siuntima iš TD3 modelio į unity agento aplinką. TD3 atliko serverio rolę, nuolatos pirmutinis siusdavo duomenis agento aplinkai bei iš jo laukdavo atsakymo. Žinutėms siųsti buvo pasirinktas json formatas dėl dviejų priežasčių: abi skirtingos aplinkos turi json nuskaitymo bibliotekas bei į json lengva įrašyti norimą informaciją.

Buvo įgyvendintos dvi esminės žinutės: naujo epizodo bei duomenų rinkimo. Šios dvi žinutės pasižymi panašia duomenų forma - išsaugome tekstinę komandą bei pridedame norimus duomenis. Toliau pateiktos dvi skirtingos siunčiamos komandos:

1. Naujo epizodo komanda. Serveris praneša agentui reikalavimą pradėti naują epizodą. Jį gavęs agentas inicializuoja vorą į pradinę epizodo poziciją ir išsiunčią būsenos duomenis į serverį. Serveris gavęs šiuos duomenis gali pradėti mokymosi procesą.
2. TD3 gavęs voro būsenas aproksimuoją naują veiksmą bei nurodo serveriui išsiųsti jį agentui. Voras gavęs veiksmus juos atlieka. Juos atlikęs atgal išsiunčią serveriui naujai įgytas būsenas, apdovanojimą bei ar ši nauja būsena terminuoja epizodą.

Šis duomenų apsikeitinėjimas pateiktas 21 paveikslėlyje.

2.6. TODO

-black box, kiek užtrunka laiko apmokyti, kodėl pasirinktas paralelus mokymasis. -rezultatai, galime pastebėti, kad agentas bebėgdamas pradeda stabdyti, tikėdamas, kad pasiekus žalia kubeli reikės apsisukti. Pagrindinis modelio gerumo matas

-TD3 metode paaiškinti inference punkta? Ir tada apmokymosi dali?

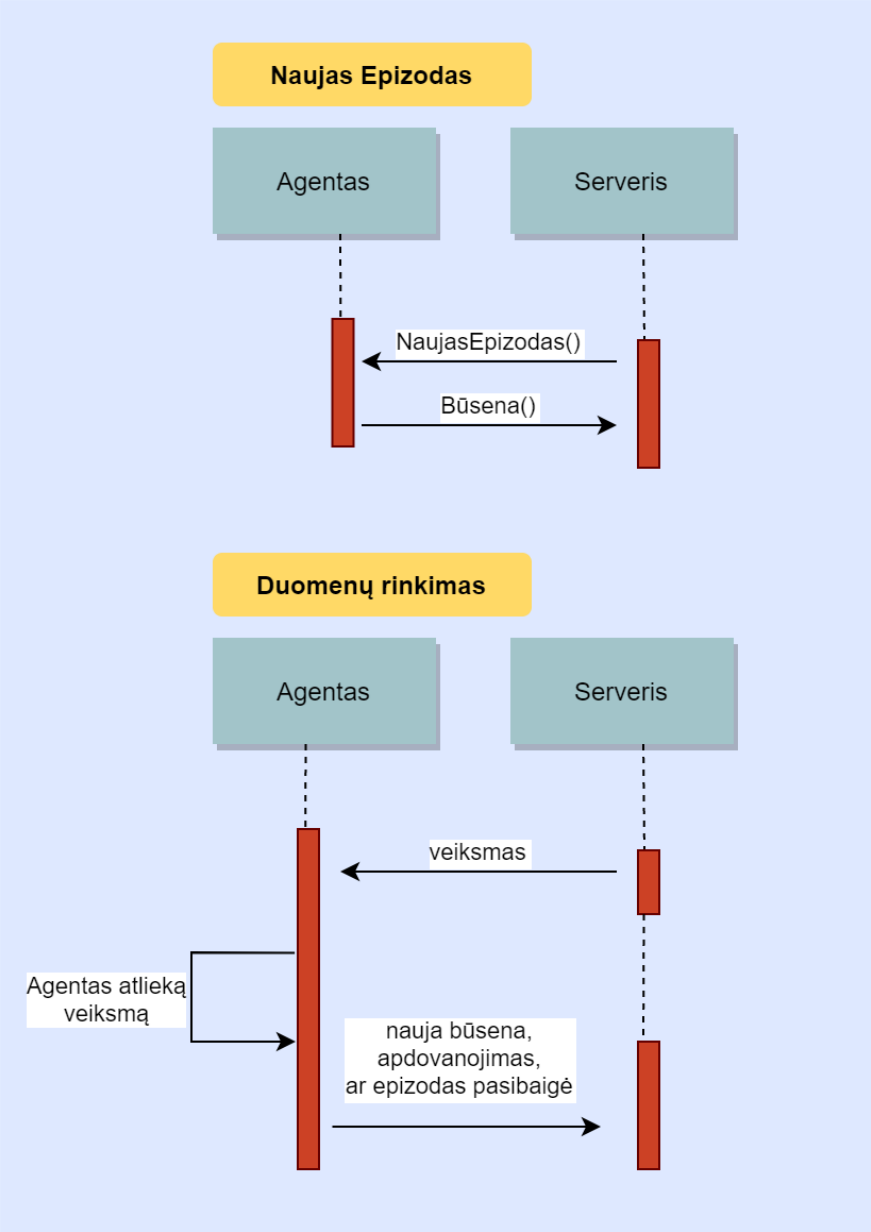
2.6.1. Pirmo skyriaus skyrelio poskyris

Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Pirmo skyriaus skyrelio poskyrio paragrafas. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

2.7. Skatinamasis mokslas

hehe



21 pav. Serverio bei agento komunikavimo sluoksnis

Išvados

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim.

Lentelė 1 is an example of a referenced LaTeX element.

Col1	Col2	Col2	Col3
1	6	87837	787
2	7	78	5415
3	545	778	7507
4	545	18744	7560
5	88	788	6344

1 lentelė. Table to test captions and labels.

Literatūra

[1] Sutton, R. & Barto, A. 1987, 'A Temporal-Difference Model of Classical Conditioning', in Proceedings of the Ninth Annual Conference of the Cognitive Science Society, Seattle, WA, pp. 355–78.PDF.

[2] Watkins, C.J.C.H. (1989). Learning from Delayed Rewards (PDF) (Ph.D. thesis). University of Cambridge. EThOS uk.bl.ethos.330022/ PDF

[3] **?**Methods and Apparatus for Reinforcement Learning, US Patent #20150100530A1**?** (PDF). US Patent Office. 9 April 2015. Retrieved 28 July 2018. PDF

[4] Human-level control through deep reinforcement learning PDF

[5] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, et al.. Deterministic Policy Gradient Algorithms. ICML, Jun 2014, Beijing, China. ffhal-00938992f PDF

[6] CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess,Tom Erez, Yuval Tassa, David Silver & Daan Wierstra Google Deepmind London, UK countzero, jjhunt, apritzel, heess, etom, tassa, davidsilver, wierstra @ google.com PDF

[7] Addressing Function Approximation Error in Actor-Critic Methods Scott Fujimoto 1 Herke van Hoof 2 David Meger 1 PDF

A. Priedai