



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	3
Lentelių sąrašas	4
Įvadas	5
1. Teorinė dalis	7
1.1. Neuroninių tinklų klasifikatorių tipai	7
1.1.1. Neuroninio tinklo struktūra	7
1.1.2. Neuroninio tinklo priekinė propogacija	9
1.1.3. Neuroninio tinklo atgalinė propogacija	10
1.1.4. Binarinis klasifikatorius	11
1.1.5. Daugiau nei viena klasė	11
1.1.6. Tiesinės regresijos atsakas	12
1.1.7. Tiesinė regresija su daugybe atsako kintamųjų	13
1.2. Neuroninių tinklų reguliarizacija	13
1.2.1. L1 bei L2 reguliarizacija	14
1.2.2. Atvirkštinis išmetimas	14
1.2.3. Duomenų augmentacija	15
1.2.4. Įvesties normalizacija	15
1.3. Neuroninių tinklų parametrų inicializacija	15
1.3.1. Gradiento sprogimas bei nykimas, aktyvacinių funkcijų parametrų priskirimas	15
1.4. Mokymosi greičio didinimas	16
1.4.1. Mažos mokymosi imties metodas	16
1.4.2. Stochastinis nuolydis	16
1.5. Gradientinio nuolydžio optimizavimas	16
1.5.1. Momentinis gradientinis nuolydis bei eksponentinškai pasvertų svorių vidurkis	16
1.5.2. Normalizuotas gradientinis nuolydis	17
1.5.3. RMSProp	18
1.5.4. Adam	18
1.6. Skatinamasis mokymasis	18
1.6.1. Skatinamojo mokslo pagrindiniai kintamieji, stacionariojo pasiskirstymo problema	20
1.6.2. Ne stacionarus pasiskirstymas	23
1.6.3. Markovo grandinės	23

1.6.4.	Apdovanojimai bei tikslai	25
1.6.5.	Vertės bei veiksmų taisyklių funkcijos	26
1.6.6.	Verčių bei veiksmų taisyklių iteravimas	28
1.6.7.	Dinaminis programavimas	29
1.6.8.	Monte Karlo metodai	31
1.6.9.	TD bei Q-mokymasis	32
1.6.10.	Gilusis skatinamasis mokslas, gilusis-Q	33
1.6.11.	Veiksmo taisyklių gradientas	35
1.6.12.	TODO	36
1.6.13.	Pirmo skyriaus skyrelio poskyris	36
1.7.	Skatinamasis mokslas	36
Išvados		37
Literatūra		38
A. Priedai		39

Iliustracijų sąrašas

1.	Gilaus neuroninio tinklo bendra struktūra	8
2.	Gradientinio nuolydžio pavyzdys	9
3.	Priekinės propogacijos vieno neuroninio sluoksnio apskaičiavimo grafas	10
4.	Atgalinės propogacijos vieno neuroninio sluoksnio apskaičiavimo grafas	11
5.	Sukonstruota stalo teniso simuliacija	13
6.	Regularizacija - po kaire L1, po dešine L2	14
7.	Gradientinė optimizacija su ir be momentinio greičio. SGD momentinis bei Adam konverguoja į globalų minimumą, like konvergavo į lokalų minimumą .	17
8.	Bendras skatinamojo mokslo modelis	19
9.	Apdovanojimo pavyzdys, pagal voro greičio (raudona rodyklė) bei norimo judėjimo (žalia rodyklė) vektorių atitikimu.	20
10.	N lošimo aparatų, su skirtingais vidutiniais laimėjimais	21
11.	Dulkio siurblio markovo grandinė	24
12.	Vienas agento epizodas	25
13.	Veiksmų taisyklių pasiskirstymas, kur $\epsilon = 5$	26
14.	Viena optimizacijos iteracija	28
15.	Dinaminio programavimo ve	30
16.	Giliojo Q neuroninio tinklo struktūra	34

Lentelių sąrašas

1. Table to test captions and labels. 37

Ivadas

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 neuroniniai 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 teisingą 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 išspręsta 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 daugybę sričių: automobilių autonominis vairavimas, logistikos grandžių optimizavimas, robotikos judėjimo problemos ir dar daugybę 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 parenti 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

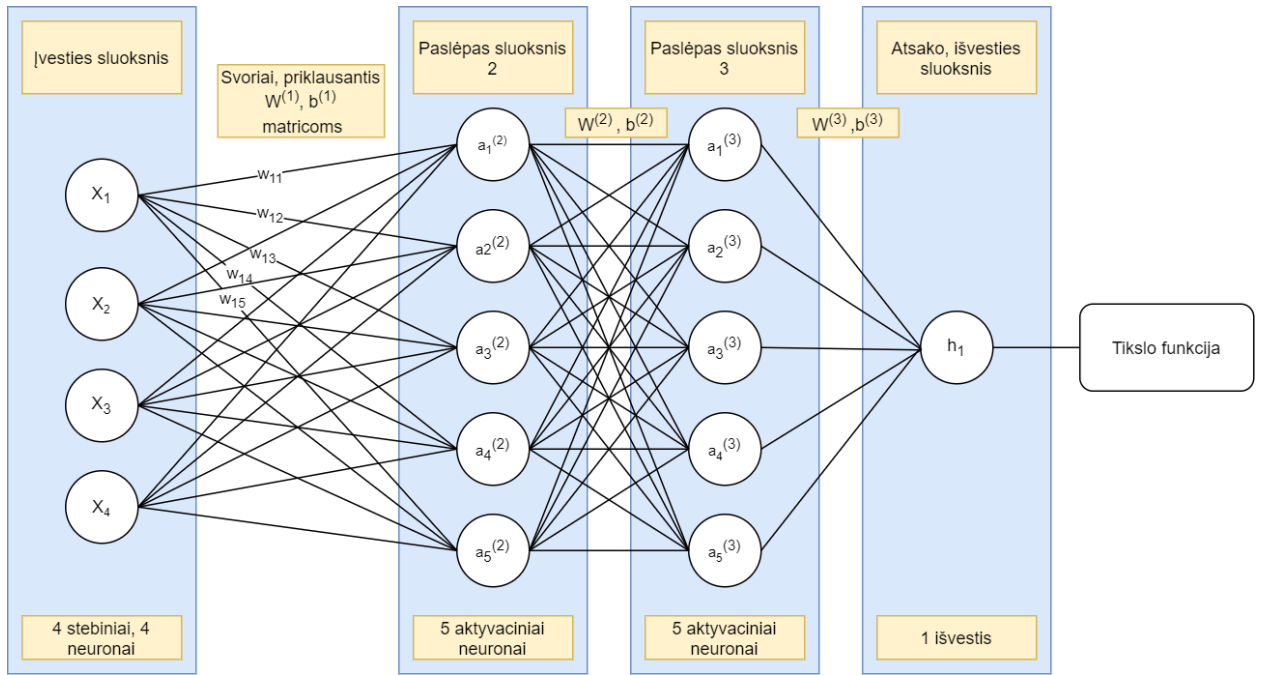
1.1.1. Neuroninio tinklo struktūra

Neuroninių tinklų sandarą galima 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 regresijai, 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 sluoksniu, 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 sluoksnius. 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](#) paveikslėlyje.

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 skirtingos ašys yra svorių reikšmės bei viena iš jų yra tikslo funkcijos rezultatas. Pateiktame 2 paveikslėlyje apatinės 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 mažina tikslo funkcijos paklaidą. Šią trajektoriją atspindi 2 paveikslėlyje esanti raudona tiesė, kuri kiekvienos iteracijos metu (mėlynai taškai) po truputi slenka žemyn, optimizuoja svorius.

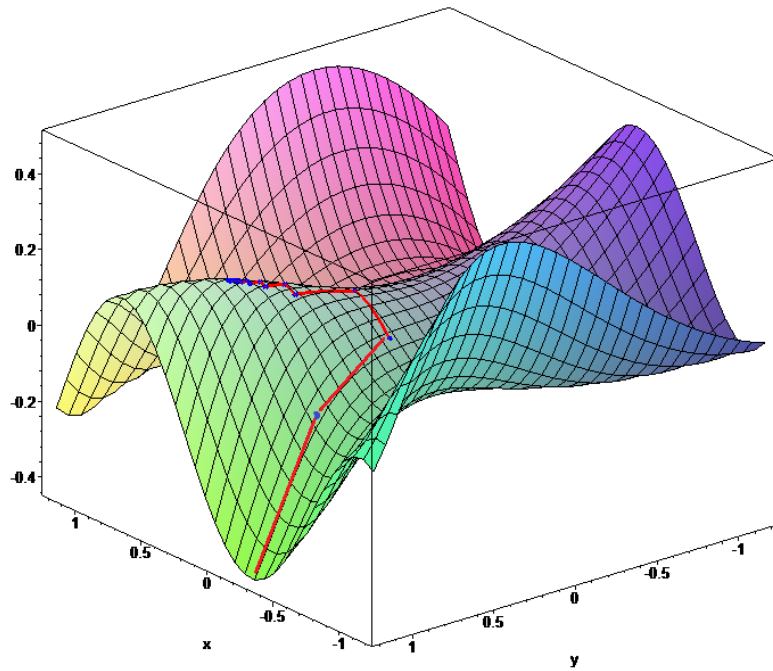
Vėlgi gradientinio nuolydžio vektorizacija yra tokia pati tiek įvesties, tiek paslėptuose sluoksniuose, bet priklausomai nuo analizuojamos problemos - skirtinga išvesties sluoksnyje. Gradiento vektorizacija:

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

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

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

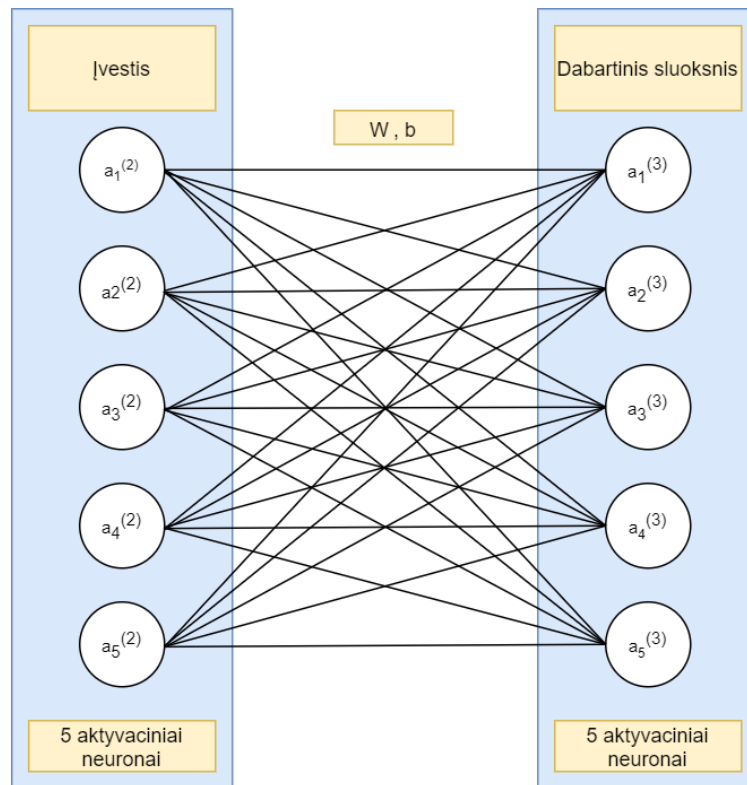
Žymėjimas: stulpelių skaičius arba mokymosi duomenų kiekis yra žymimas m . Toliau bus aptariamasi priekinio, atgalinės propagacija bei tikslo funkcijos, kurios sudaromos atitinkamai pagal turima klasifikacijos problemą.



2 pav. Gradientinio nuolydžio pavyzdys

1.1.2. Neuroninio tinklo priekinė propogacija

Igyvendinti 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 dešinę) į neuroninį tinklą galima žiūrėti kaip į atskirus blokus susidedančius iš įvesties bei išvesties, kaip tai matosi **3** paveikslėlyje.

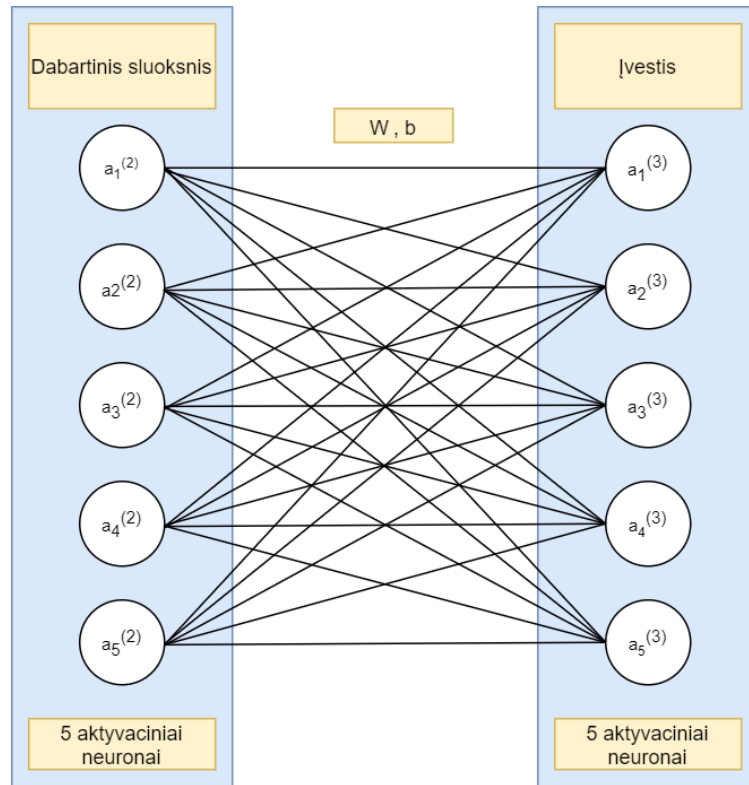


3 pav. Priekinės propogacijos vieno neuroninio sluoksnio apskaičiavimo grafas

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 rezultatai. Jos bus reikalingos apskaičiuoti gradientinį nuolydį, kaip tai matosi 1 bei 2 formulėse. Paskutinės iteracijos metu, atlikus aktyvacinės funkcijos apskaičiavimą, lieka tik apskaičiuoti tikslo funkcijos rezultatai. Ši struktūra pasižymi formulių universalumu, jos tinka neuroniniams tinklams, kurie yra negilieji bei gilieji, kitaip tariant - nepriklauso nuo sluoksnių kiekio. Atlikus priekinę propogaciją ir suskaičiavus tikslo funkcijos klaidą, galime pradėti ją mažinti, judėdami priešinga gradiento linkme. Tai atlieka sekanti potėmė - 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 iteracijas, tik šį kart pradėdami nuo tikslo 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:

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

Šios funkcijos neurono gradientas yra susiprastiną į šią formą:

$$dZ^l = A^l - Y, \quad (4)$$

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:

$$W^l = W^l - \alpha dW^l \quad (5)$$

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:

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

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:

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

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 perduodamas į tikslo funkciją. Taigi šis metodas yra dažnai naudojamas klasifikuoti reikšmes, kurios turi daugiau nei vieną klasę.

1.1.6. Tiesinės regresijos atsakas

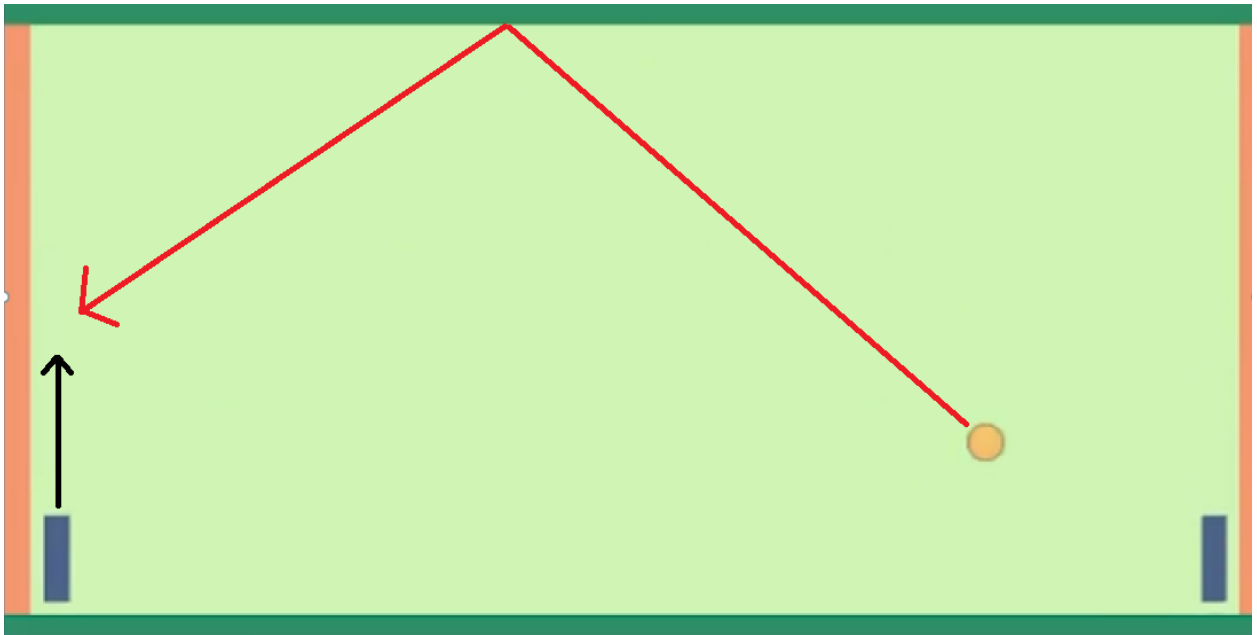
Jei atsakas yra regresinis, tuomet tikslo funkcija tampa didžiausių kvadratų optimizacijos tikslo funkcija ir paskutinis neuronas neturi aktyvacijos funkcijos. Tikslo funkcijos dydžiausių kvadratų sumos vidurkio formulė:

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

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 dviem 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 kamuoliuko 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:

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

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 kontrolieris, 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 regularizacija

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

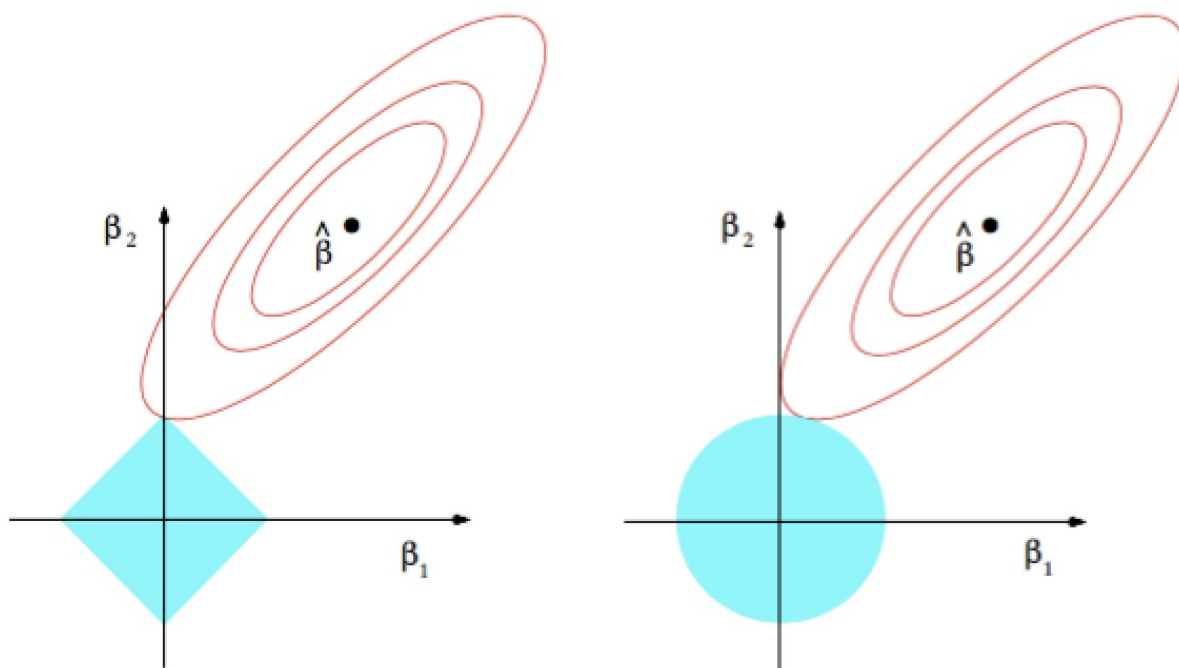
1.2.1. L1 bei L2 reguliarizacija

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:

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

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 imtį.

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 būdu niekada nekonverguoti. Taip pat svoriai nėra proporcingai panašūs, vieni turi žymiai didesnę įtaką nuolydžiui, kiti mažesnę. 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 sproginimas bei nykimas, aktyvacinių funkcijų parametrų priskirimas

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

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

Matome, kad jei W^l parametrai priskiriami su didelėmis reikšmėmis, turint daugybę sluoksnių mūsų tiek neurono tiek svorių reikšmės eksponentiškai išauks. Tuo tarpu jei parametro reikšmės W^l priskiriamos su mažesnėmis nei vieneto reikšmėmis, 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 pasiskirstymo 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 pasiskirstymo 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.

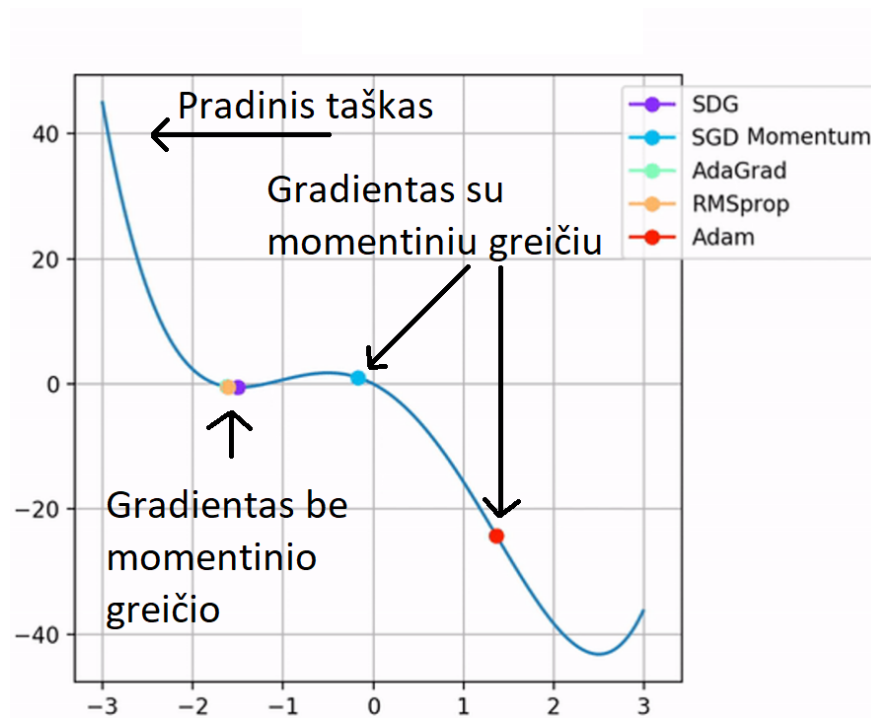
1.5.1. Momentinis gradientinis nuolydis bei eksponentinškai pasvertų svorių vidurkis

Pagrindinis šio optimizavimo principas remiasi momentiniu nuolydžiu, kuomet gradientas įgyja pagreiti ties dažniausiai pasikartojančiu vektoriumi. Primena įsibėgėjusi automobilį. Šis įsibėgėjimas padeda išvengti lokalių minimumų, 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 mokymosi

paremti gradientinio nuolydžio metodai praskrieja lokalių 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 gradiento įsibėgėjimo greičiu ir gradientas tik truputi suletėja. Šio gradiento formulė primena eksponentiškai pasvertų svorių vidurkio formulę ir išsireiškia tokiu pavidalu:

$$Vdw = \beta_1 * Vdw + (1 - \beta_1) * dw \quad (9)$$

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 lokalių 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:

$$Sdw = \beta_2 * Sdw + (1 - \beta_2) * (dw)^2 \quad (10)$$

Ir svorių atnaujinimo formulė:

$$W = W - \alpha \frac{dw}{\sqrt{Sdw}} \quad (11)$$

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 judėjimą plokštuma. Tai pasiekiamo sukombinuojant momentinį greitį bei RMSProp:

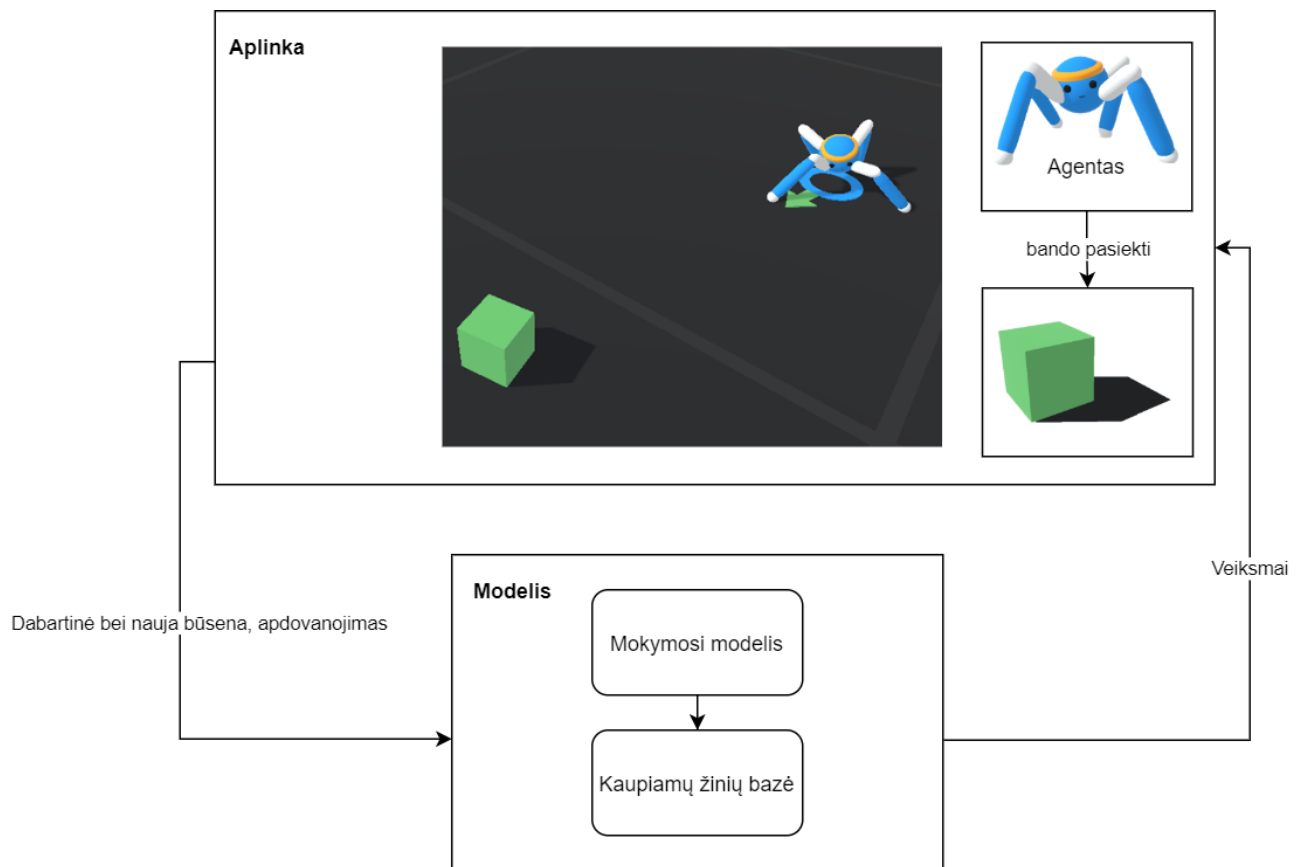
$$W = W - \alpha \frac{Vdw}{\sqrt{Sdw}} \quad (12)$$

Vdw suteikia momentinį greitį kiekvienam svoriui. Bet jei momentinis greitis yra didelis Sdw jį sumažina. Tačiau jei gradientas užstringa lygumoje, Sdw padeda atstrigti. Šį metodą pritaikiau minėtame automobilio kontrolerio pavyzdyje. Jis padėjo pasiekti trygubai mažesnę 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 nepasiekiamo. 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 judinimą, kuriomis bandoma kuo arčiau priartinti vorą ties žaliu kubeliu.

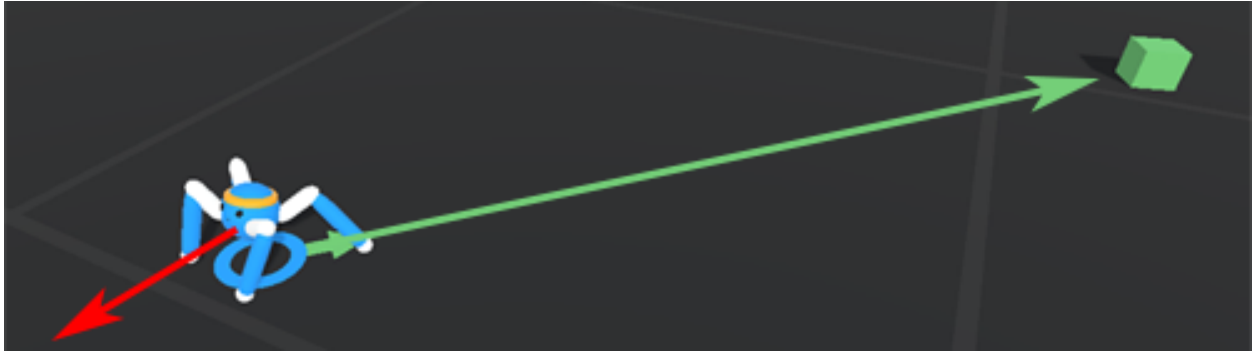


8 pav. Bendras skatinamojo mokslo modelis

Šio modelio veikimo principai išsiskaido į šias dalis:

1. **Aplinka.** Aplinkoje egzistuoja mūsų agentas, kuris priiminėja modelio sugeneruotus veiksmus. Pateiktame pavyzdyje tai būtų voro galūnių judėjimo trajektorijos. Atlikęs duotus veiksmus, voras įgauna naujas būsenas, pasistumi 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 vektoriaus sutapimą su norimos krypties vektoriumi, kaip tai matosi 9 paveikslėlyje. Šiuo atveju apdovanojimas būtų nulinis, nes vektorius trajektorijos visai nesutampa. Atlikus veiksmą bei apskaičiavus jo naudingumą, tokiu būdu gaunamas ryšys tarp veiksmo, kuris sugeneruoja naują būseną, bei įgyto apdovanojimo, naujoje būsenoje. Šis ryšys tarp veiksmo bei gauto rezultato yra neseniai atgimusio skatinamojo mokslo pamatas, kuriuo moderniausi algoritmai ne per seniausius 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 vektorius su šiomis reikšmėmis - praeita būsena, atliktas veiksmas, nauja būsena ir apdovanojimas. Mokymosi modelis naudojasi sukaupytą žinių baze gerinti savo modelio rezultatus. Modelio tikslas yra pateikti tokias veiksmų sekas, kurios suteiktų didžiausią įmanomą apdovanojimą. Vorų pavyz-

džiu tai būtų sėkmingas žalio kubelio pasiekimas. Modelio apmokymui yra sukurta daugybė 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.

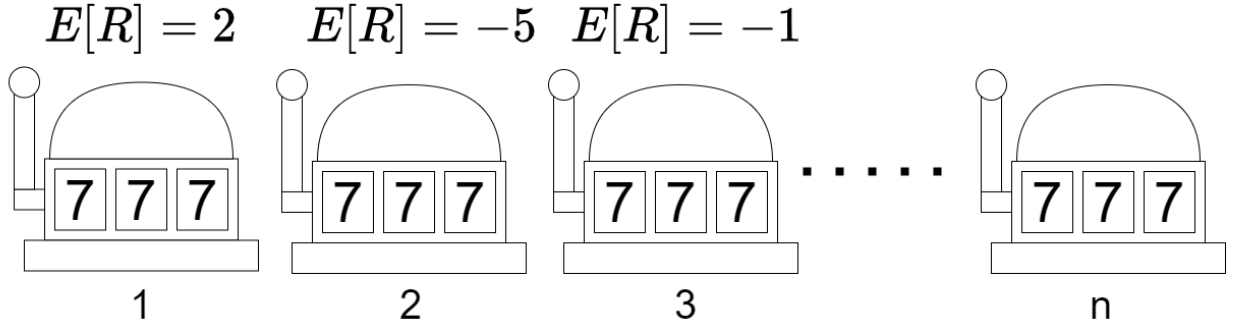
Taigi visas skatinamasis mokslas paremtas tik vienu kintamuoju - apdovanojimo gausumu. Šis apdovanojimas apdovanoja pageidautina elgesį bei baudžia neigiamą elgesį. Tokiu būdu agentas skatinamas siekti didžiausio ilgalaikio nuopelno, kas veda ties optimaliu sprendimu. Tolimesniuose skyriuose bus aptarema 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 pavyzdys, 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 didžiausią įmanomą apdovanojimą. Kiekvienas aparatas duoda atsitiktinį laimėjimą iš $\mathcal{N}(\mu = \mathcal{N}(0,2), \sigma = 1)$ pasiskirstymo. Žaidimo sesija, dažniausiai vadiname epizodu, nėra ribojama, agentas gali atlikti begalybę 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 sukaupą laimėjimą. Jis pritaiko viena iš populiariausių 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.

$$A_t = \operatorname{argmax}_a Q_t(a) \quad (13)$$

Tačiau su maža ϵ tikimybe, agentas pasirenka atsitiktinį lošimo aparatą. Taigi kiekviename t žingsnyje agentas renkasi lošimo aparatą, jį aktyvuoja ir gauna 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 optimalėsnio lošimo aparato, 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.

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

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 dėsnį, $Q(a)$ 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 užduotį - kokį pasirinkti veiksmą a , kuris suteiktų dydžiausią apdovanojimą.

Pagrindiniai skirtumai atsiranda dėl kompiuterio resursų optimizavimo. Viena iš problemų, su kuria iškart 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 daugybė esminių metodų šią problemą mažinti bei išvengti. Viena iš metodų, kurio formą galima

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

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

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

Algoritmas 1: Lošimo aparato pseudo algoritmas

```

Q(a) ← 0;
N(a) ← 0;
for ∞ do
    A ← { arg maxa Q(a) su tikimybe 1 - ε (lygiąsias reikšmes pasirenkame atsitiktinai)
          atsitiktinis veiksmas su tikimybe ε }
    R ← LošimoAparatas (A) ; // Pasirenkame veiksmą A ir gauname apdovanojimą R
    N(A) ← N(A) + 1 ; // Kiek kartų kiekvienas veiksmas buvo pasirinktas
    Q(A) ← 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 įpattingai svarbi, nes ji yra ištaka tiek TD, tiek Q-mokymosi, tiek bellmano formulėms, kurios yra skatinamojo mokslo pamatas. Apibendrinant formulę ją galima šitaip išreikšti:

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

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

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

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 pasiskirstymas 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 daugybę 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:

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

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.

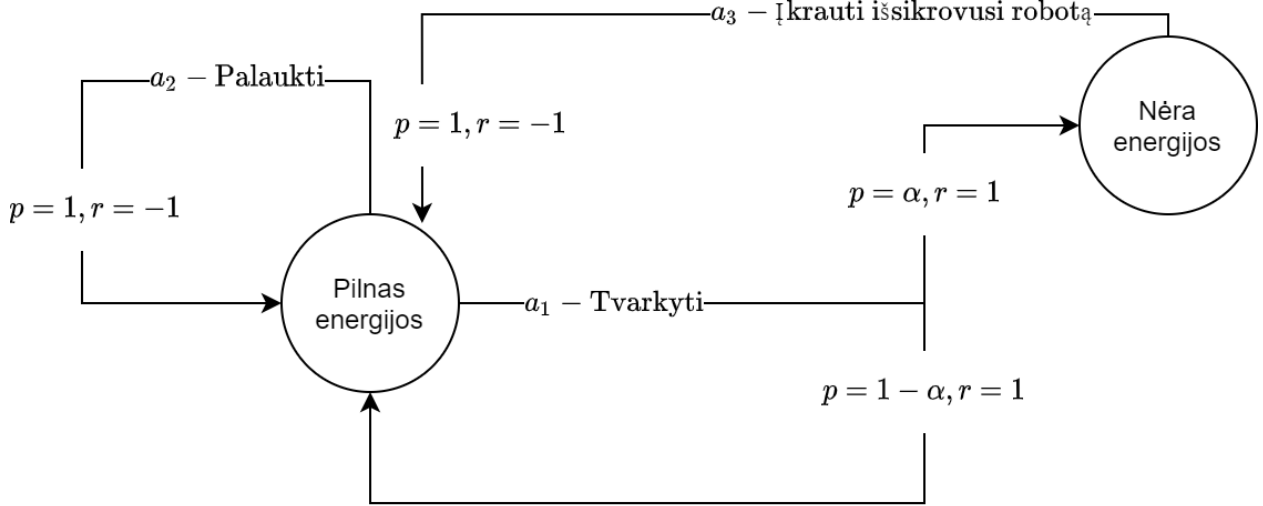
1.6.3. Markovo grandinės

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

1. s - esama būseną.
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. Dulquio siurblio markovo grandinė

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

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

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

Veiksmų pasirinkimas vadovaujantis didžiausiu 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:

$$P(X_{n+1} = x \mid X_0, X_1, X_2, \dots, X_n) = P(X_{n+1} = x \mid X_n), \quad (20)$$

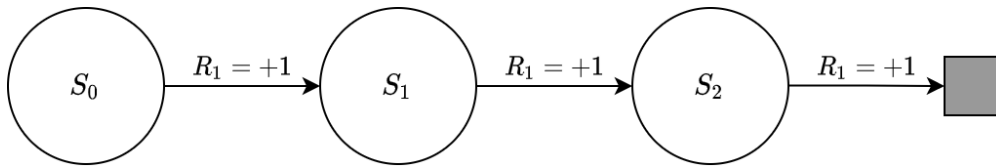
kur X yra prieš tai buvę įvykiai. Ši savybė leidžia apskaičiuoti tikėtiną apdovanojimą pasitelkiant tik turimą būseną, prieš tai buvusios būsenos neturi įtakos tikėtinam apdovanojimui.

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

Apdovanojimų 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:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T, \quad (21)$$

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 proporcingus mažinimus:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (22)$$

kur γ yra "nuolaidos" kintamasis, kurio dažniausia vertė yra 0.9. Ši vertė indikuoja, kad 11 žingsnio apdovanojimas bus proporcingas 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:

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

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

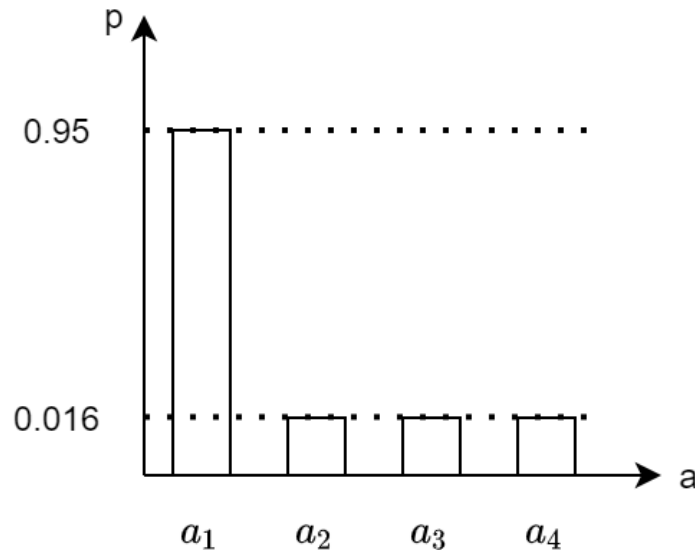
Vienas iš svarbiausių praktinių patarimų, padedančių pasiekti dydžiausią ilgalaikį apdovanojimą yra priskirti apdovanojimus už pasiektą tikslą, o ne skirti apdovanojimus už tai kaip mes norėtume, kad agentas pasiektų tikslą. Agentas mokymo procese turi pats atrasti optimaliausius veiksmus vedančius ties tiek tikslo pasiekimu, tiek dydžiausiu epizodo apdovanojimu. Kitame skyriuje bus apibrežiama, kaip būtent agentas optimizuoja savo elgesį, jo viena iš tikslo funkcijos konceptu, 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 būseną yra nusakoma pagal savo vertę. Ši vertė yra žymima $v_\pi(s)$ ir nusako pasirinktos s būsenos tikėtiną apdovanojimą:

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

Žinant kiekvienos būsenos tikėtiną 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 pasiskirstymas 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 $\epsilon = 5$

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

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

Turint kiekvienos būsenos taisyklių pasiskirstymą, galima suskaičiuoti kiekvienos būsenos tikėtiną reikšmę, kaip tai matosi [1.6.7](#) formulėje.

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

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 :

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

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

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

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

$q_\pi(s, a)$ 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 iteracinį procesą, kurio tikslas yra optimizuoti šias reikšmes.

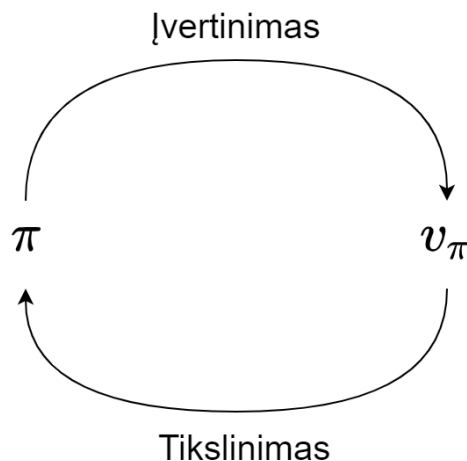
1.6.6. Verčių bei veiksmų taisyklių iteravimas

Optimizavimo metu, mes ieškome tokių būsenų verčių bei optimalių taisyklių, kurios gražintų dydžiausią epizodo apdovanojimą. Yra daugybė 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. Ši procesą iteruojame, iki konvergacijos pasiekimo. Nenaudojamas metodas, nes labai retai tenkinama šio metodo privaloma sąlyga - 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 tikslas 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

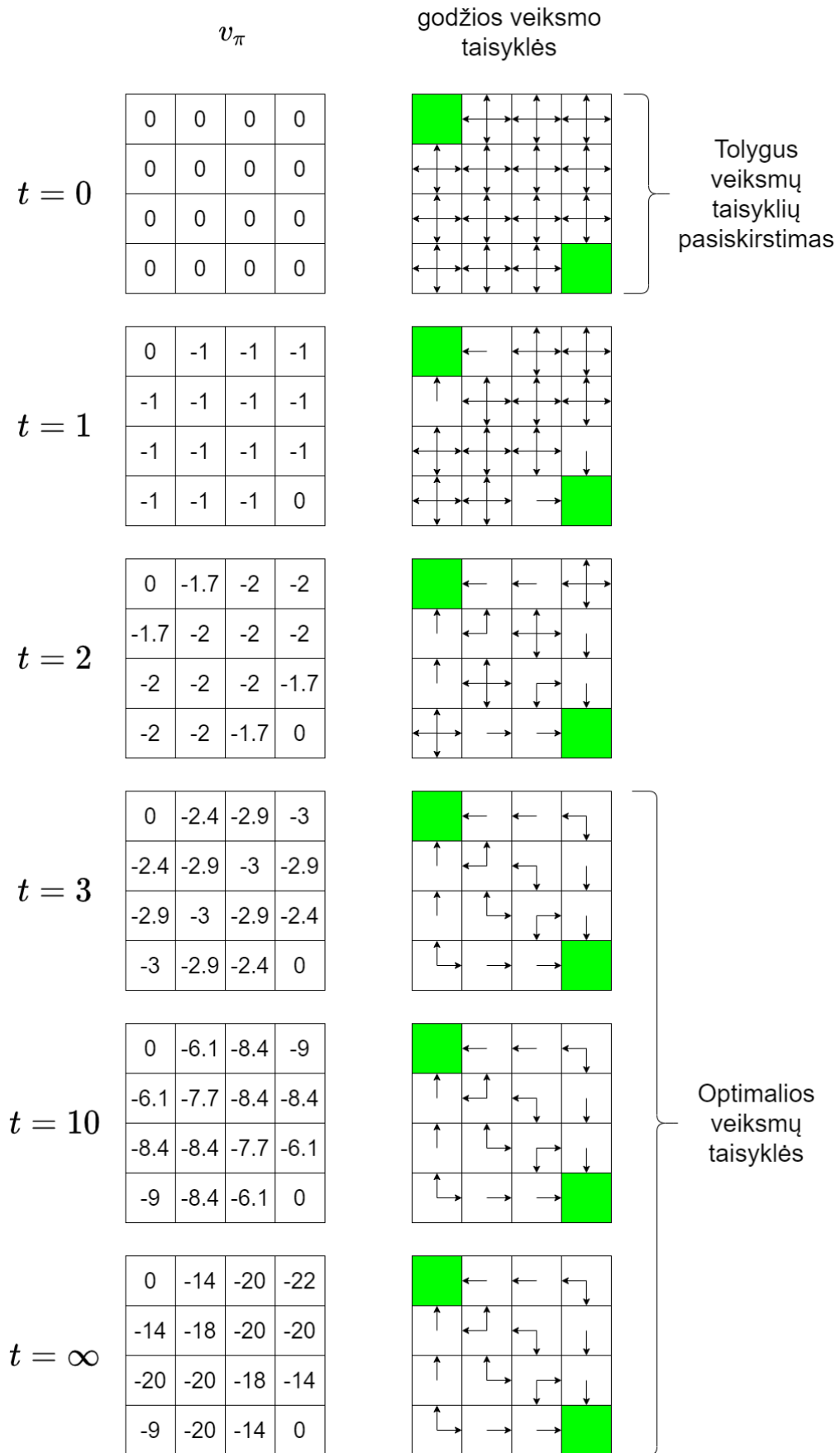
Šiuos žingsnius kartojame tol, kol pasiekiamo $v_\pi(s)$ konvergaciją - naujos iteracijos nepakeičia $v_\pi(s)$ reikšmių. Tolimesniuose skyriuose trumpai apžvelgsiu šio proceso įgyvendinimą dinamininiuose bei monte karlo metoduose. Neuroninių tinklų aproksimacijos bus plėtojamos giliausiai, kadangi šiuo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

1.6.7. Dinaminis programavimas

Įvertinimo bei tikslinimo procesą, pateikta paveikslėlyje, paprasčiausiai iliustruoja dinaminio programavimo metodas pateiktas 15 pavyzdyje. Šiame pavyzdyje mūsų agentas atsiranda viename iš keturiolikos langelių ir gali eiti į keturias puses, pateiktomis rodyklėmis. Agento tikslas patekti į žalius langelius. Kairėje paveikslėlio pusėje yra pateikti $v_\pi(s)$ įverčiai naudojantis formule, o dešinėje perskaičiuojame optimalias veiksmų 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 įvertinimą, 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 $v_\pi(s)$ reikšmėms 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 išsiskaidymas 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 ve

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 specifiniam 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 daugybės 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 epizodą sekant $\pi : S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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

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

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

Išsaugoti G į Vertės(S_t, A_t) sąrašą

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

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

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

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

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

2 Algoritmo problema, kaip ir prieš tai buvusių metodų, 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.

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

Šios formulės ištakas galima pastebėti tiek 15 bei 18 formulėse. Šias dvi formules sukombinavus, galime perrašyti 1.6.7 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 naudoja šia TD atnaujinimo formulę:

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]. \quad (30)$$

o Q-mokymasis šia:

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

Nors šios formulės yra vos ne identiškos, tačiau jos skiriasi skirtingomis optimaliomis veiksmų trajektorijomis. SARSA modelio atveju agentas stengiasi 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 įrangą. 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, kuomet Q-mokymasis ir pasižymi. Q-mokymasis [2] yra vienas iš populiariausių bei seniausių egzistuojančių metodų. 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 pasi-
telkia neuroninių tinklų aproksimacijomis. Aproksimacijos padėjo išspręsti viena dydžiausių
skatinamojo mokslo problemų - aukštų būsenų dimensijų neįmanoma konvergavimą. Dėl
šios priežasties toliau bus aptariamas gilusis Q - vienas iš pirmųjų ir nors jau nebeefektyviu
modeliu, sukėlusiu 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 $[R + \gamma \max_a Q(S', a)]$ pa-
vidalu. 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:

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

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.

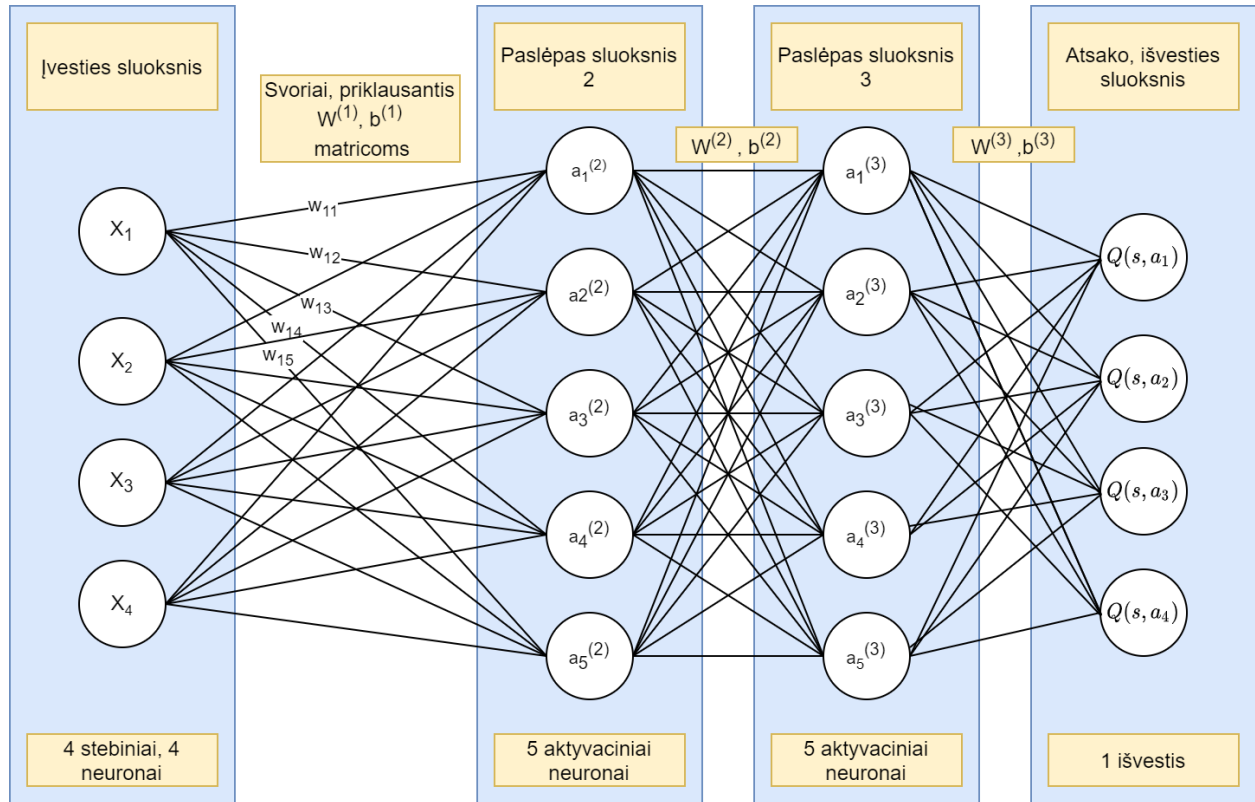
Taigi neuroninis tinklas stengesi aproksimuoti tikslą, kuris geriau žinomas kaip G_t arba
naujausias aproksimuotas $q_\pi(s, a)$ narys. Tuo tarpu neuroninio tinklo įvestis yra būsenos
vektorius s . Kadangi naudojamas neuroninis tinklas, vektoriaus s dydis nėra svarbus, kon-
vergacija išlieka sparti.

Optimizavimo procesas susideda iš šių tryju dalių:

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

Patirčių talpyklos bei dviejų neuroninių tinklų sukūrimas Agentas tyrinėdamas
aplinką, kaupia duomenis patirčių talpykloje, dar kitaip vadinama buferiu. Ši patirčių tal-
pykla 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 didžiausia.



16 pav. Giliojo Q neuroninio tinklo struktūra

Atlikus veiksmą mes apskaičiuojame gautą apdovanojimą r bei naują buseną s' , kurioje atsiradome atlikus veiksmą a . Tokiu būdu 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 ištrauktą iš talpyklos. Svarbu neuroninį tinklą apmokyti tik su duomenimis iš patirčių talpyklos, norint išvengti koreliacijos. Šią koreliacijos tendenciją lengva pastebėti intuityviai išanalizavus patirčių talpyklos veikimo principus:

1. Atsitiktinės duomenų eilutės (s, a, r, s') paimamos iš patirčių talpyklos. Dėl atsitiktinio ėmimo, didžioji dalis patirčių - duomenų eilučių, yra iš skirtingų trajektorijų, epizodų.
2. Su paimtais duomenimis apmokome modelį. Šio proceso metu, dėl atsitiktinio ėmimo, skirtingos būsenos įgyja 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 apmokėti neuroninį tinklą su tuo pačiu dėmeniu ne vieną kartą. Šis bruožas naudingas apmokėti 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:

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

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 kiekis 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 vieną 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

Svorių atnaujinimas Q-mokymosi yra nebenaudojamas nes įvyko vienas dydžiausių proveržių nes buvo įrodytas veiksmų taisyklių gradientas

1.6.11. Veiksmo taisyklių gradientas

$$P(\tau | \theta) = \rho_0(s_0) \prod_{t=0}^T P(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t) \quad (34)$$

$$\nabla_\theta P(\tau | \theta) = P(\tau | \theta) \nabla_\theta \log P(\tau | \theta) \quad (35)$$

$$\log P(\tau | \theta) = \log \rho_0(s_0) + \sum_{t=0}^T (\log P(s_{t+1} | s_t, a_t) + \log \pi_\theta(a_t | s_t)) \quad (36)$$

$$\begin{aligned}
\nabla_{\theta} \log P(\tau \mid \theta) &= \nabla_{\theta} \log_0(s_0) + \sum_{t=0}^T \left(\nabla_{\theta} \log P(s_{t+1} \mid s_t, a_t) + \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t) \right) \\
&= \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t)
\end{aligned} \tag{37}$$

1.6.12. TODO

-qmokymasis, bufferis -policy gradient -aktorius kritikas -DDPG -TD3

1.6.13. 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.

1.7. Skatinamasis mokslas

hehe

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 L^AT_EX 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](#)

A. Priedai