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

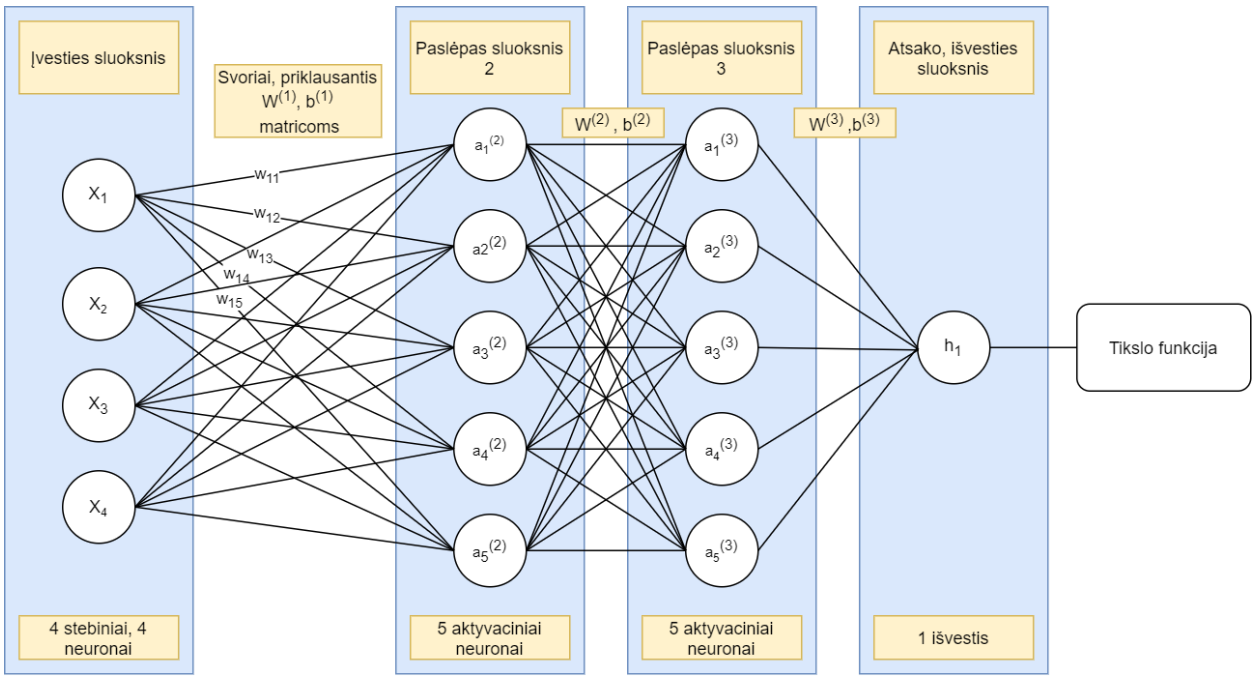
1.1.1. Neuroninio tinklo struktūra

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.

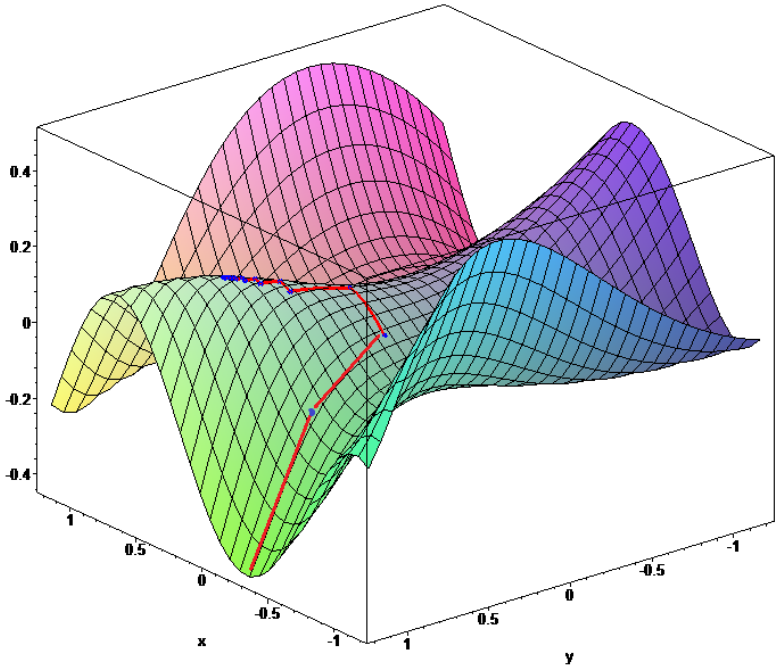
Vėlgi gradientinio nuolydžio vektorizacija yra tokia pati tiek įvesties, tiek paslėptuose sluoksniuose, bet priklausomai nuo analizuojamos problemos - skirtinga išvesties sluoksnyje. Gradiento vektorizacija:

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

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:

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

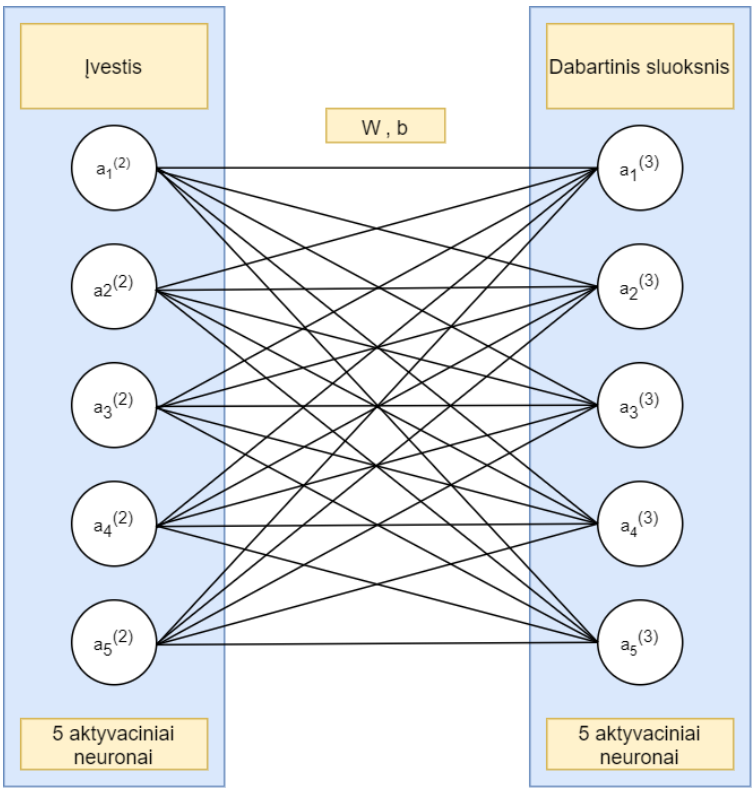
Žymėjimas: stulpelių skaičius arba mokymosi duomenų kiekis yra žymimas m. Toliau bus aptariamos priekinio, atgalinė propogacija bei tikslo funkcijos, kurios sudaromos atitinkamai pagal turima klasifikacijos problemą.



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.

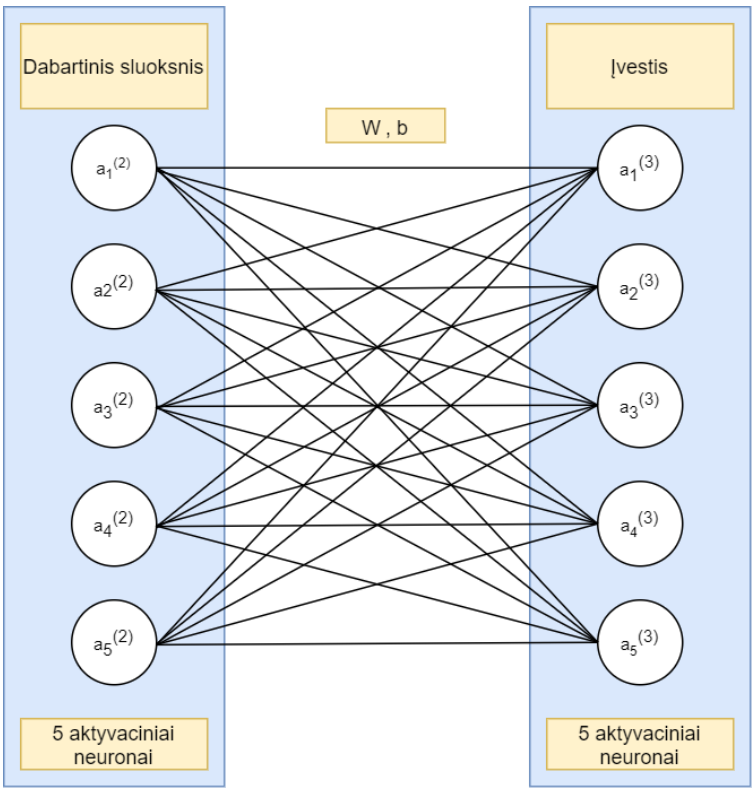


3 pav. Priekinės propogacijos vieno neuroninio sluoksnio apskaičiavimo grafas

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

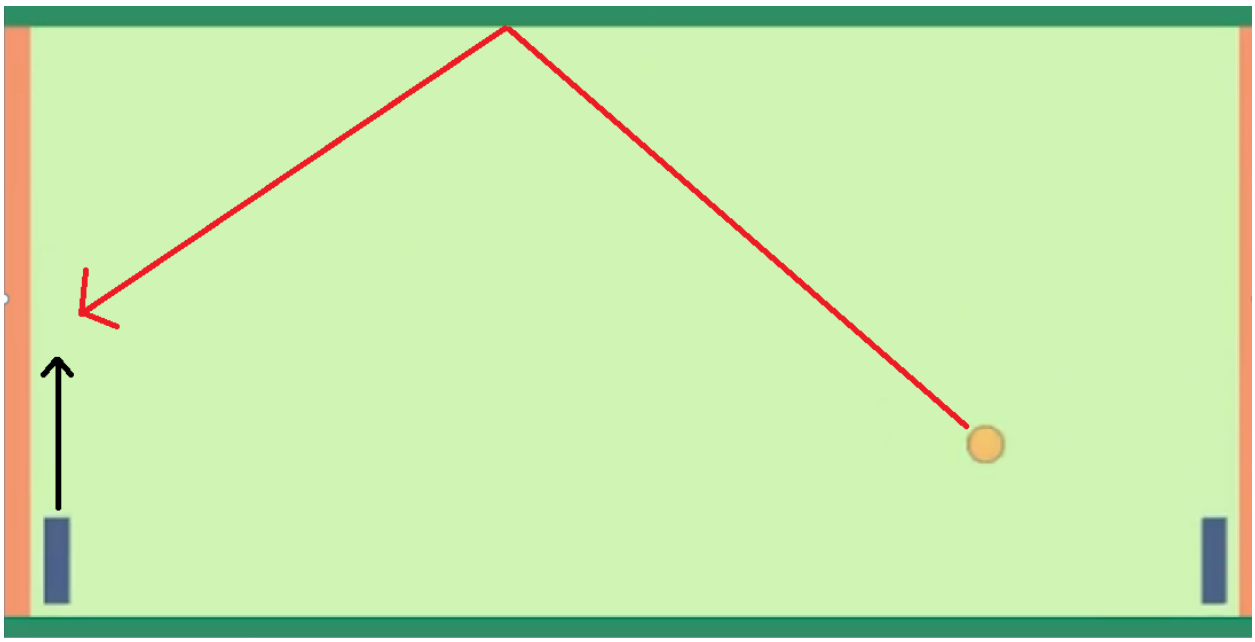
Jei atsakas yra regresinis, tuomet tikslo funkcija tampa didžiausių kvadratų optimizacijos tikslo funkcija ir paskutinis neuronas neturi aktyvacijos funkcijos. Tikslo funkcijos dydžiausių kvadratų sumos vidurkio formulė:

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

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.

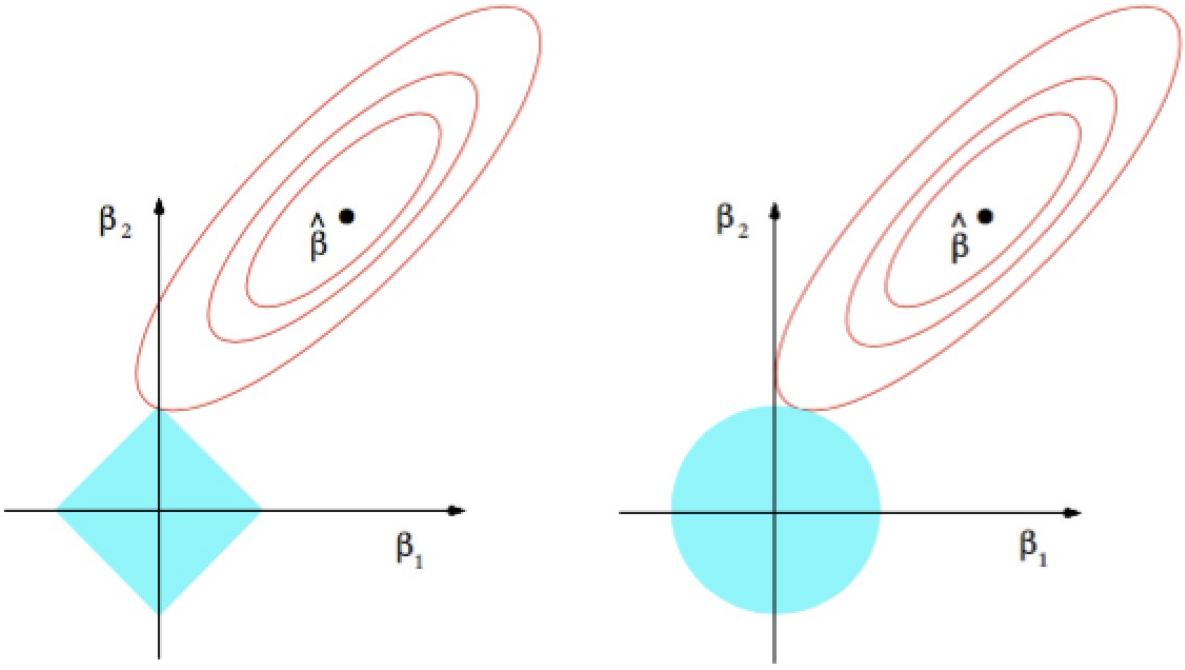
1.2.1. L1 bei L2 reguliarizacija

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.

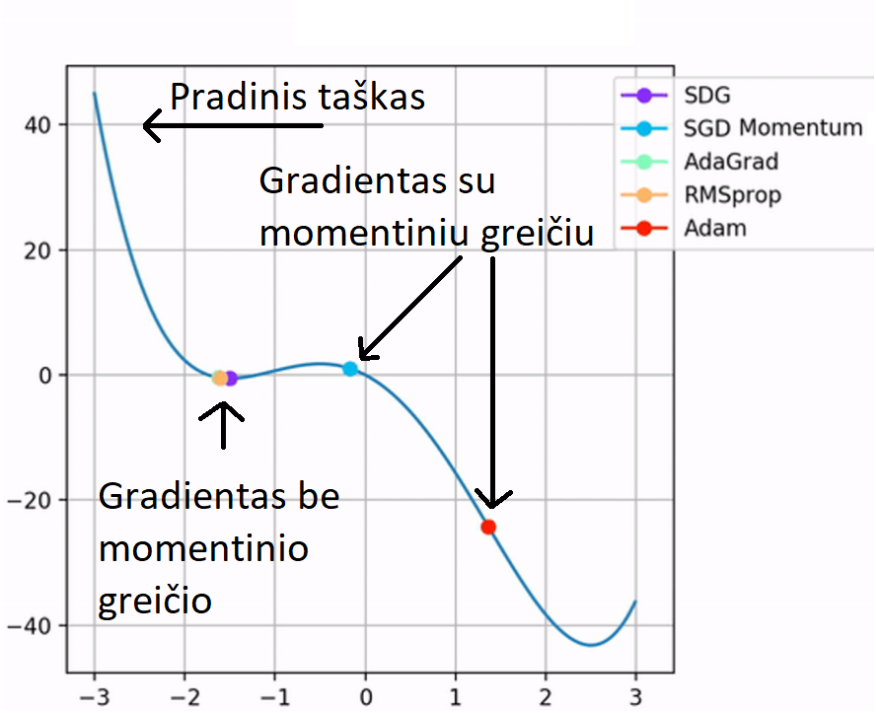
1.5.1. Momentinis gradientinis nuolydis bei eksponentinškai pasvertų svorių vidurkis

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:

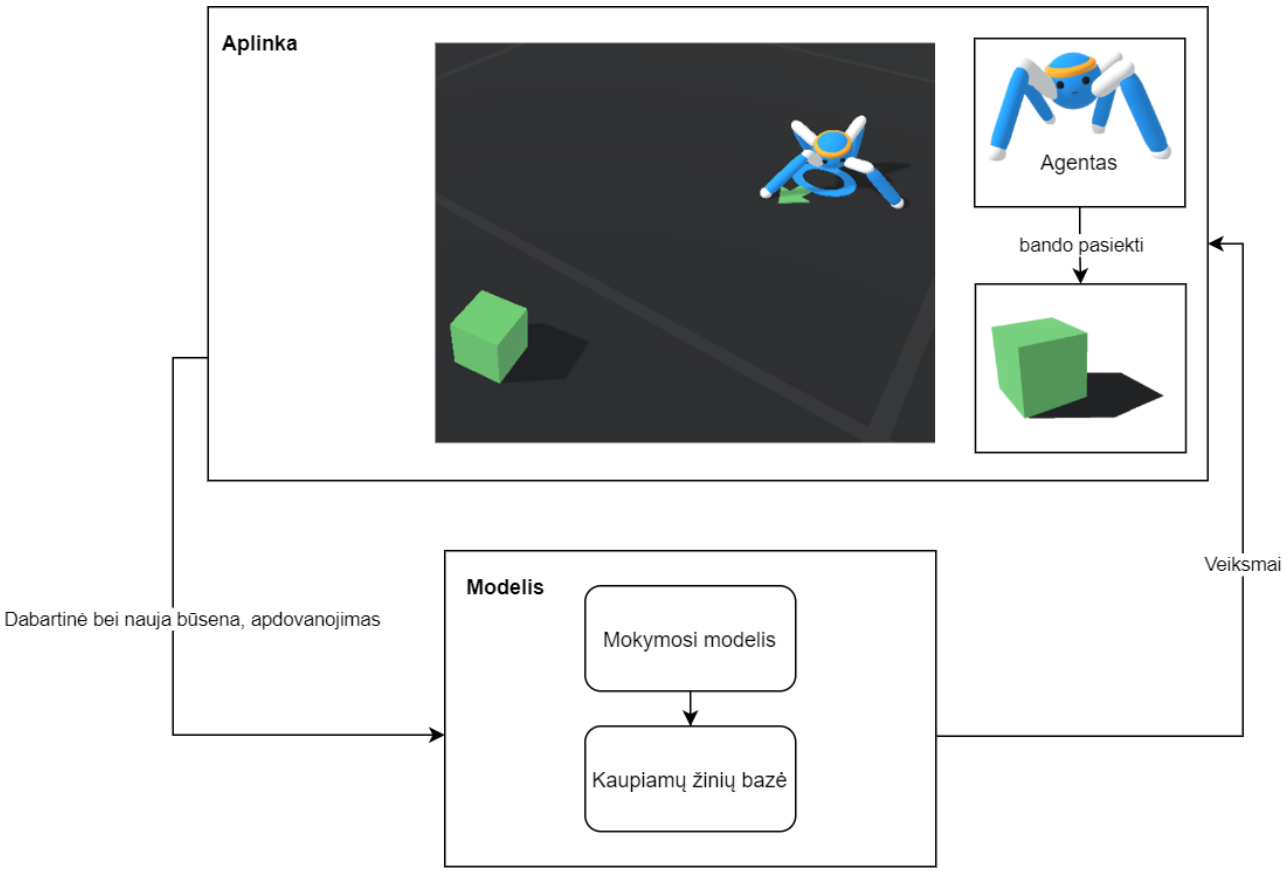
$$W = W - \alpha \frac{Vdw}{\sqrt{Sdw}} \tag{12}$$

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.

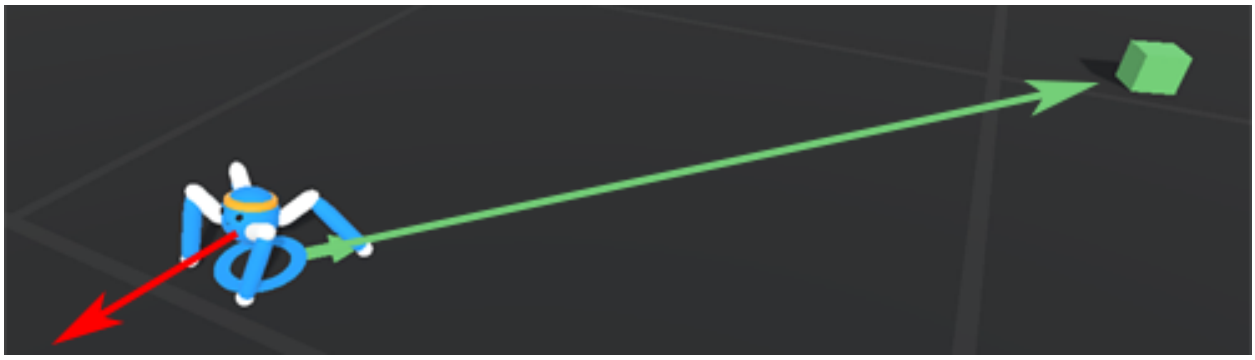


8 pav. Bendras skatinamojo mokslo modelis

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.

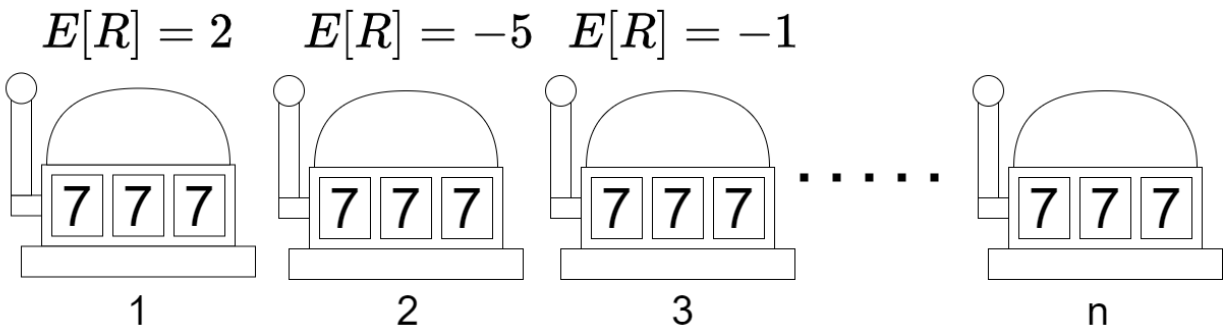
Taigi visas skatinamasis mokslas paremtas tik vienu kintamuoju - apdovanojimo gausumu. Šis apdovanojimas apdovanoja pageidautina elgesį bei baudžia neigiamą elgesį. Tokiu budu agentas skatinamas siekti dydžiausio ilgalaikio nuopelno, kas veda ties optimaliu sprendimu. Tolimesniuose skyriuose bus aptarema 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:

Algorithm 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
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$;

end

Result: 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 refeqn:Vidurkio formule:

1. Toliau plėtojant duotą pavyzdį, galima įsivaizduoti situaciją, kurioje, kas tam tikrą laiko intervalą, kiekvieno aparato apdovanojimo funkcija pakinta ir įgyja nauja $\mathcal{N}(\mu = \mathcal{N}(0{,}2), \sigma = 1)$ 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 tuos judančio vidurkio principus.

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

1.6.3. 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 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] Tobias Oetiker. The Not So Short Introduction to LaTeX2e. Or LaTeX2e in 157 minutes. Version 5.06, June 20, 2016.

A. Priedai