VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Kursinis darbas

Objektų atpažinimas ir sekimas kompiuterinės tomografijos vaizduose

(Object detection and tracking in computed tomography)

Atliko: 3 kurso Programų sistemų studentas
Paulius Milmantas (parašas)

Darbo vadovas:
Linas Petkevičius (parašas)

Vilnius
2020

Įvadas

Pastaraisiais metais į medicinos sritį labai sparčiai skerbiasi informacinės technologijos. Atliekant įvairias diagnostikas ar tiriant ligas, vaistus yra pasitelkiama programinių sistemų pagalba. Tačiau didžiąją radiologo darbo dalį sprendimus turi priimti jis pats be jokios programinės įrangos pagalbos, nors dažniausiai yra apibrėžti tam tikri ligų aptikimų algorimtai, kaip vėžinių lastelių aptikimas, lūžiai ir taip toliau. Mano darbo tikslas yra prisidėti prie programinės įrangos kūrimo radiologams ir sukurti programą, kuri aptinka plaučius, kad vėliau tai galėtų būti panaudota tolimesnei IT plėtrai medicinos srityje.

Šiandien viena iš labiausiai perspektyvių sričių informacinių technologijų srityje yra giliojo mokymosi metodai. Jie padeda duomenyse aptikti sunkiai pastebimus dėsningumus ir nuspėti išeities rezultatus su tam tikra klaidos tikimybe. Šiame darbe kalbėsiu vieną iš šių metodų: dirbtinius neuroninius tinklus. Juos pasirinkau nes kompiuterinėsje tomografijos vaizdai turi labai daug informacijos ir juos apdoruoti naudojant logines taisykles yra per daug sunku. Tačiau net ir jiems ši užduotis yra per sudėtinga, jeigu tiriamos yra įvairios mutacijos arba nedažnos ligos, nes su kiekvienu nauju nematytu ligos atveju, yra reikalinga apmokyti modelį, o tai reikalauja daug duomenų.

Pasirinktam mano dirbtinių neuroninių tinklų metodui viena iš galimų alternatyvų yra ”sprendimo medžiai”. Tai yra prižiūrimo mokymosi metodas skirtas duomenų klasifikacijai ir duomenų regresijai. Šis metodas naudoja taisyklių rinkinį. Norint gauti mažą šio metodo paklaidą, turime aprašyti vis daugiau šių taisyklių. Taisyklės paprastai būna aprašomos naudojant ”if, else” aprašus. Mano nagrinėjamame atvejyje duomenys yra labai dideli ir įvairūs, todėl aprašyti visas šias taisykles užtruktų neproporcingai daug laiko ir tam tikrų radiologijos žinių.

Turinys

1. Dirbtinio neuroninio tinklo sudėtis

1.1. Bazinė struktūra

Daugelį dirbtinių neuronų tinklų sudaro panašios struktūrinės dalys:

1. Įvesties sluoksnis: tai dalis kuri priima įvestį ir perduoda kitiems sluoksniams.
2. Išvesties sluoksnis: tai dalis, kuri naudoja aktyvacijos funkciją kuri grąžina galutinį tinklo rezultatą: tikimybių rinkinį, kuris parodo kokia tikimybė, kad objektas atitinka tam tikrą klasę.
3. Paslėptas sluoksnis: perduoda svorius iš praeito sluoksnio į sekantį.
4. Susijungimai ir svoriai: tarp kiekvieno neurono, kuris yra susijungęs, turi savo svorį, pagal kurį yra pakeičiama perduodama reikšmė.
5. Aktyvacijos funkcija: tai funkcija, kokia turi būti neurono išvestis.
6. Mokymosi taisyklė: apibrėžia, kaip tinkle keičiasi svoriai, kad tinklas išvestų norimus rezultatus.

Dirbtinį neuroninį tinklą taip pat sudaro svoriai ir tendenciškumas. Šie du parametrai yra tinklo automatiškai sugeneruojami ir apmokant tinklą pagal duomenis, svoriai ir tendenciškumas yra automatiškai tinklo keičiami. Tendenciškumas apibrėžia kaip tinklo išvestis yra nutolusi nuo tikrosios reikšmės. Tendenciškumas priartina tinklo išvestį prie tikrųjų reikšmių. Svoriai nurodo koks stiprus yra ryšys tarp neuronų, taip nurodant, kaip tinklo reikšmės kinta pačiame tinkle. Jeigu tinklui yra paduodama reikšmė, o pirmasis tinklo neuronas, kuris priima reikšmę turi mažą svorį, tai reiškia, jog galutiniams rezultatui ši reikšmė neturi daug įtakos. Jeigu svoris yra didelis, tai duota reikšmė turi daug įtakos rezutatui.

Svoriai ir tendenciškumas yra tinklo tvarkomi treniravimo proceso metu. prižiūrimasis tinklo treniravimas vyksta dviem žingsniai: priekine ir atgaline sklaida.

1.2. GPU naudojimas

Šiame darbe bus naudojamas GPU, atliekant tinklo treniravimą. Tai yra spartesnis būdas už CPU naudojimas, nes CPU turi kelis galingus branduolius, o GPU žymiai daugiau. Giliajame mokyme, naudojami aritmetiniai veiksmai yra ganėtinai paprasti, todėl juos apdoroti gali ir CPU ir GPU. Tačiau GPU turi daugiau branduolių, kas leidžia atlikti daugiau matematinių veiksmų paralizuotai, negul CPU. Taip yra pasiekiamas didesnis treniravimo našumas.

Didesniam treniravimo našumui pastebėti naudojant GPU, negu CPU, buvo atliktas paprastas bandymas: naudojant Google tensorflow struktūrą, apmokame sistemą atpažinti kur yra žmogaus

plaučiai kompiuterinės tomografijos vaizduose su CPU ir GPU. Vidutiniškai apdoruojant 60 megabaitų duomenų, vienam ciklui tiklas su CPU resursais užtrunka 6 sekunes, o naudojant GPU vidutiniškai vienas žingsnis užtrunka 0.3 sekundės.

Šiame darbe naudojamas Tensorflow karkasas naudoja NVIDIA CUDA sistemą. CUDA yra paralelinio skaičiavimo platforma, naudojama su grafikos plokštėmis. Ji kiekvieną skaičiavimą paleidžia ant atskiros gijos. Visi duomenys yra suskirstyti į vienos, dviejų ir trijų dimensijų blokus. Kiekvienams blogas gali turėti virš 512 gijų. Visos gijos kurios operuoja viename bloke dalinasi bendra GPU atmintimi. Jeigu norima optimizuoti programą ir paleisti jos skaičiavimus ant GPU, tada reikia susiskaičiuoti kiek vyks paralelinių operacijų ir kiek iš jų dalinsis bendra atmintimi. Pagal apskaičiuotą kiekį nurodome kiek mes norime turėti blokų ir kiek kiekvienas blokas turės atskirų gijų. Prieš kiekvieną operaciją, siekiant pasiekti optimalų našumą, jeigu reikia įkeliame kintamuosiu į bendrą bloko atmintį. Po atliktų skaičiavimų išvestus rezultatus turime iškelti iš bendros atminties.

TODO

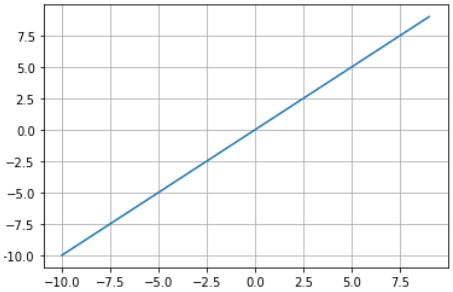
Atgalinė sklaida vyksta, kai tinklas išveda rezultatą, patikrinama ar rezultatas yra teisingas ir atitinkamai tinklas reaguoja į tai. Inicijuojant tinklą svoriai yra nustatomi atsitiktinai. Kiekvienam duomenų rinkiniui yra stebima tinklo išvestis ir tikrinama kokia ji turėtų būti. Padaryta klaida yra perduodama tinklui atgal, per praeitus sluoksnius, einant nuo išvesties sluoksnio. Šiame žingsnyje yra naudojama optimizavimo funkcija. Ji parodo kokie turėtų būti svorių ir tendenciškumo pokyčiai tinklai ir taip yra keičiami tinklo svoriai ir tendenciškumas atsižvelgiant į padarytą klaidą. Kai klaidos yra padaromos pakankamai mažos ir jos yra mažesnės už duotą ribinę reikšmę, yra laikoma, kad tinklas baigė mokymosi procesą.

2. Aktyavcijos funckijos

2.1. Tiesinė aktyvacijos funkcija

$$R(x, a) = x * a$$

Tiesinės aktyvacijos funkcijos mių diapazonas yra labai didelis ir nėra dvejetainis, kaip dalis kitų funkcijų. Tačiau ši funkcija nėra plačiai naudojama, nes jos išvestinė yra konstanta, kas reiškia jog gautas gradientas nepriklauso nuo 'x'. Taigi, jeigu spėjime yra klaida, vykdant atgalinę sklaidą mes reikšmę taisysime konstanta, neatsižvelgiant į pačią klaidą.

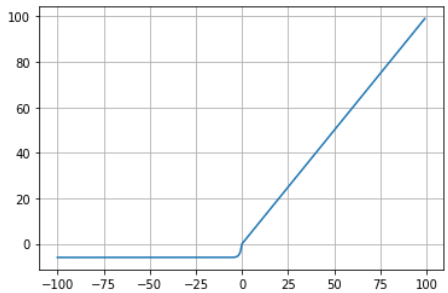


1 pav. A boat.

2.2. ELU

$$R(z) = \begin{cases} z; z > 0 \\ \alpha * (e^z - 1); z \leq 0 \end{cases}$$

ELU funkcija vadinasi: 'Exponential Linear Unit'. Ši funkcija paprastai konverguoja į nulį greičiau nei kitos ir taip duoda tikslesnius rezultatus. Skirtingai nuo kitų funkcijų, ši priima papildomai konstantą , kuri turi būti teigiamas skaičius. Ši funkcija yra labai panaši į RELU funkciją, tačiau ELU tampa tolygi lėtai, kol išvestis pasiekia , o RELU staigiai sutolygėja.

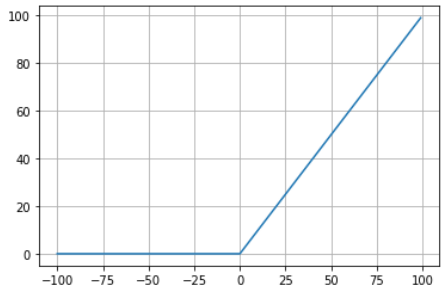


2 pav. A boat.

2.3. ReLU

$$R(z) = max(0, z)$$

ReLU funkcija vadinasi: ‘Rectified Linear Unit’. Ši funkcija pasiekia panašų efektyvumą kaip ir Sigmoid funkcija, tačiau ji reikalauja mačiau skaičiavimo resursų nei sigmoid ir tanh, nes minėta formulė yra paprastesnė, nei kitos.



3 pav. A boat.

2.4. Sigmoid

$$R(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid funkcijos parametrai priima realiuosius skaičius ir gražina reikšmę [0; 1]. Šios funkcijos geros savybės yra: tolydus gradientas, prognozuojamas išvesties diapazonas, netiesiškumas, visur diferencijuojama ir monotoniška.

2.5. Softmax

$$softmax(x)_i = \frac{exp(x_i)}{sum_{j=1}^{n} exp(x_j)}$$

Softmax funkcija priima parametrų skirstinį ir gražina tikimybių skirstinį, kurio bendra suma yra lygi vienam

2.6. Argumentacija už pasirinktą aktyvacijos funkciją

Šiame darbe bus naudojama Softmax aktyvacijos funkcija. Ši funkcija buvo pasirinkta dėl dėl dviejų priežasčių.

Pirma, galima teigti, kad Softmax funkcija bando optimizuoti entropijos skirtumą tarp tikros reikšmės ir gautos. Entropija tarp tikros reikšmės ‘p’ ir spėjamos ‘q’ yra apskaičiuojama pagal:

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

Taip pat entropiją galima aprašyti ir pagal Kullback-Leibler divergenciją:

$$H(p, q) = H(p) + D_{KL}(p||q)$$

Delta p entropijos reikšmė yra lygi nuliui, taip optimizuojama divergencija tarp Kullback-Leibler abiejų distribucijų. Taigi, galime teigti, jog Softmax funkcija bando optimizuoti entropiją tarp tikros reikšmės ir spėjamos.

Antra priežastis yra dėl tikimybinės interpretacijos… TODO

3. Problemos su aktyvacijos skaičiaja matematika

3.0.1. ”Underflow” ir ”overflow”

Ian Goodfellow Yoshua Bengio ir Aaron Courville apibrėžė ‘overflow’ ir ‘underflow’ problemas giliojo mokymosi algoritmuose. Minėta ‘underflow’ problema pasireiškia, kai funkcijos reikšmė yra labai arti nulio ir ji yra apvalinama į nulį. Kadangi kai kurie algoritmai gali ‘sulūžti’ dėl nulinio parametro (pvz. dalybos iš nulio). Kita problema yra ‘overflow’. Ji pasireiškia kai gauta reikšmė yra labai didelė ir ji yra apvalinama į begalybę.

3.0.2. Gradientų optimizavimas

TODO

4. Optimizavimo algoritmai

Tinklo siekiamas tikslas yra atliekant kuo mažiau mokymosi iteracijų, pasiekti norimą išvesties tikslumą. Šiam mokymosi greičiui ir tikslumui pasiekti ir naudojami optimizavimo algoritmai.

4.1. Gradientinis nuolydis

Gradientinio nuolydžio algoritmas Tai yra pati paprasčiausia optimizavimo funkcija.

$$\delta = \delta - \alpha * \Delta L(\delta)$$

Pagal algoritmą yra atliekama daug iteracijų. Kiekvienos iteracijos metu yra apskaičiuojama delta. Ji yra gaunama iš esamos deltos atėmus mokymosi žingsnio konstanos ir nuostolių funkcijos gradientą parametro delta atžvilgiu. Kadangi žingsnis kurį atliekame kiekvienos iteracijos metu yra konstanta, jeigu pradinis parinktas taškas yra toli nuo lokalaus minimumo, tai prireiks daug iteracijų, kol jis bus pasiektas. Šiuo metodu taip pat kiekviena iteracija yra peržiūrimi visi duomenys, kad nėra gerai jeigu duomenų yra itin daug. Tokiu atveju reiktų daug atminties. Šis algoritmas nebus naudojamas, nes nėra atsižvelgta į praeitas iteracijas ir minimumas yra per lėtai pasiekiamas, algoritmas.

4.2. Poaibio gradieninis nuolydis

Šis algoritmas yra panašus į paprastą gradientinį metodą, tačiau yra kelis pakeitimai. Kai yra pereinama per visus duomenis, po kiekvieno duomenų elemento peržiūrėjimo, yra atnaujinami parametrai. Todėl po kiekvienos iteracijos, turime visų duomenų elementų pakeitimų sudėtį. Todėl minimumas yra pasiekiamas statesne trajektorija ir nereikalinga turėti daug atminties. Pagrindinis šio metodo trūkumas yra galimas jo lėtumas. Pereiti per visus duomenis užtrunka daug laiko ir nevisos duomenų elementų iteracijos prie parametrų pakeitimo daug neprisideda.

4.3. Mažo poaibio gradientinis nuolydis

Abiejų minėtų algoritmų idėjų kombinacija išvedė mažo poaibio gradientinį metodą. Šis algoritmas suskirsto duomenis į poaibius, kurių elementų skaičius yra daugiau nei 1. Dažniausiai poaibių dydis yra pasirenkamas dvejetas pakeltas kokiu nors laipsniu, nes vaizdo plokštėm tokius poaibius apdoroja geriau. Šis algoritmas yra greitesnis nei poaibio gradientinis metodas, nes yra išvengiama mažų iteracijų, kurių duomenų elementas mažai prisideda prie parametrų pakeitimo. Taip pat šio metodo pasiekiamas tikslumas yra didesnis nei kitų minėtų algoritmų, nes atsitiktinai parenkami poaibiai skiriasi kiekvieno poaibių paskirstymo metu, taip sudarant daugiau varijacijų ir daugiau triukšmo. Kadangi bendrai yra apdorojama daugiau duomenų variacijų, šis metodas gali pasiekti didesnį tikslumą. Nors šis metodas gali pasiekti didesnį tikslumą ir yra greitesnis.

4.4. Bendros gradientinių metodų problemos

Gradientinio nuolydžio metodai yra pirmo laipsnio optimizavimo funckijos, kas reiškia jog jos naudoja tik pirmo laipsnio išvestines. Kadangi pirma išvestinė rodo tik funkcijos statumą, bet ne išlenktumą, tai:

1. Jeigu antra išvestinė yra lygi nuliui, tai funkcija yra tiesinė. Todėl, mokymosi žingsnis yra lygus alpha.

2. Jeigu antra išvestinė yra didesnė už nulį tai funkcijos išlenktumas eina į viršų. Todėl funkcijos žingsnis yra mažesnis už mokymosi žingsnį ir optimizavimo funckija gali diverguoti.

3. Jeigu antra išvestinė yra mažesnė už nulį, tai funkcija yra išlenkta į apačią. Todėl funkcijos žingsnis yra didesnis už mokymosi žingsnį.

To pasekoje, optimizavimo funkcijos greitis gali būti lėtas arba ji gali diverguoti.

5. Nuostolių funkcijos

Nuostolių funkcijos grąžina skirtumą tarp tinklo išvesties ir tikrų reikšmių. Pagal nuostolių reikšmes yra atitinkamai apskaičiuojami gradientai ir atitinkamai keičiami tinklo svoriai ir tendenciškumas.

5.1. Vidutinė kvadratinė paklaida (MSE)

$$Nuostolis = \frac{1}{N} * [\sum (Y_{Gautas} - Y_{Tikras})^2]$$

MSE funckija sumuoja skirtumų kvadratus ir randa bendrą nuostolių vidurkį. Ši funkcija yra dažnai numatyta įaviriuose MATLAB ir Python algoritmuose. Vienintelė neigiama šios funkcijos pusė yra tai, kad jeigu duoti duomenys yra pirmojo laipsio, tada negalima teigti, kad rezultatai tiesiogiai koreliuoja tarpusavyje, nes MSE yra antrojo laipsnio funckija.

5.2. Šaknis iš vidutinės kvadratinės paklaidos (RMSE)

$$Nuostolis = \sqrt{\frac{1}{N} * [\sum (Y_{Gautas} - Y_{Tikras})^2]}$$

RMSE funkcija yra panaši į MSE, tik pridėta šaknies traukimo operacija. Šaknies traukimas paverčia RMSE pirmojo laipsnio funckija, todėl ją galima naudoti su pirmojo laipsnio duomenimis.

5.3. Paklaidų pakelti kvadratu suma

$$Nuostolis = [\sum (Y_{Gautas} - Y_{Tikras})]^2$$

]subsectionArgumentacija už pasirinktą aktyvacijos funkciją TODO

6. Neuroninių tinklų modeliai (SSE)

Šiame darbe yra naudojamas Tensorflow karkaso Mask RCNN inception modelis.

7. Mask RCNN inception

Mask RCNN modelį sudaro dvi dalys: vaizdo regiono pasiūlymo tinklas (RPN) ir klasifikavimo modelis. Bendras algoritmas:

1. Nuotrauka yra praleidžiama per konvoliucinį neuroninį tinklą (RCNN), kuris sugeneruoja požymių aibę, pagal kurią galima nuspėti, kad tai yra ieškomas objektas.

8. Tutorial

8.1. Poskyris

Citavimo pavyzdžiai: cituojamas vienas šaltinis [PPP01]; cituojami keli šaltiniai [Pav05; PPP+02; PPP03; PPP04; STU+02; STU01; STU03; STU04; Sur05].

8.1.1. Skirsnis

8.1.1.1. Straipsnis

8.1.2. Skirsnis

9. Skyrius

9.1. Poskyris

9.2. Poskyris

Išvados

Išvadose ir pasiūlymuose, nekartojant atskirų dalių apibendrinimų, suformuluojamos svarbiausios darbo išvados, rekomendacijos bei pasiūlymai.

Literatūra

[Pav05] A. Pavardonis. *Magistrinio darbo pavadinimas*. Magistrinis darbas, Universiteto pavadinimas, 2005.

[PPP+02] A. Pavardenis, B. Pavardonis, C. Pavardauskas ir D. Pavardinskas. Straipsnio pavadinimas. *Rinkinio pavadinimas*, p.p. 3–15, Miestas, šalis. Leidykla, 2002.

[PPP01] A. Pavardenis, B. Pavardonis ir C. Pavardauskas. Straipsnio pavadinimas. *Žurnalo pavadinimas*, IV:8–17, 2001.

[PPP03] A. Pavardenis, B. Pavardonis ir C. Pavardauskas. *Knygos pavadinimas*. Leidykla, Miestas, šalis, 2003. 172 psl.

[PPP04] A. Pavardenis, B. Pavardonis ir C. Pavardauskas. Elektroninės publikacijos pavadinimas. `http://example.com/kelias/iki/straipsnio.pdf`, 2004. 45 KB, tikrinta 2015-02-01.

[STU+02] A. Surname, B. Tsurname, C. Usurname, and D. Vsurname. Article title. In *Conference book title*, pp. 3–15, City, country. Publisher, 2002.

[STU01] A. Surname, B. Tsurname, and C. Usurname. Article title. *Journal Title*, IV:3–15, 2001.

[STU03] A. Surname, B. Tsurname, and C. Usurname. *Book title*. Publisher, City, country, 2003. 172 p.

[STU04] A. Surname, B. Tsurname, and C. Usurname. Online publication title. `http://example.com/path/to/the/article.pdf`, 2004. 45 KB, accessed 2015-02-01.

[Sur05] A. Surname. *Ttitle fo PhD thesis*. PhD thesis, Title of university, 2005.

Priedas Nr. 1
Niauroninio tinklo struktūra

4 pav. Paveikslėlio pavyzdys

Priedas Nr. 2
Eksperimentinio palyginimo rezultatai

1 lentelė. Lentelės pavyzdys

Algoritmas	$\bar{x}$	σ^2
Algoritmas A	1.6335	0.5584
Algoritmas B	1.7395	0.5647