SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 000

Polunadzirana klasifikacija rukom pisanih znakova generativnim suparničkim modelima

Marko Jelavić

Zagreb, lipanj 2017.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Zahvaljujem se mentoru, izv. prof. dr. sc. Siniši Šegviću na podršci i vodstvu kroz diplomski studij te na savjetima i pomoći prilikom izrade diplomskog rada.

SADRŽAJ

1. Uvod

Klasifikacija rukom pisanih znakova se često koristi kao uvodni problem u područje računalnog vida. Najpoznatiji takav problem je klasifikacija rukom pisanih znamenki ([20]) gdje se svakoj slici mora pridijeliti pripadajuća znamenka kao klasa. Pošto se radi o maloj bazi podataka slika niskih rezolucija, rezultati se mogu brzo dobiti te se novi modeli najčešće provjeravaju nad njom. Modeli koji uspješno obavljaju klasifikaciju rukom pisanih znakova najčešće primjenu nalaze kao dio sustava za pretraživanje informacija iz dokumenata.

Duboki diskriminativni modeli se smatraju najsuvremenijom tehnologijom iz razloga što pružaju najbolje rezultate u širokom spektru primjena te su najzaslužniji za ekspanziju primjene umjetne inteligencije u industriji. Mana takvih modela je ta što zahtijevaju ogroman broj ručno označenih podataka da bi postigli zadovoljavajuće rezultate. Označavanje podataka je izrazito skup proces te ga se nastoji zaobići.

S druge strane, duboki generativni modeli ne zahtijevaju veliku količinu označenih podataka. No, problemi s dosadašnjim modelima se javljaju pri aproksimacijama vjerojatnosnih izračuna pri modeliranju vjerojatnosti ulaznih podataka što usporava proces treniranja modela ([15]). U navedenom radu predlaže se model koji se zasniva na suparničkim neuronskim mrežama.

Model generativnih suparničkih mreža se sastoji od dviju neuronskih mreža, generatorske i diskriminatorske. Spada pod familiju modela bez učitelja, odnosno nisu mu potrebni označeni primjeri za učenje, već samo uči distribuciju ulaznih podataka. Mreža generator ima zadatak naučiti distribuciju ulaznih podataka, a to čini provođenjem latentnih varijabli na ulazu kroz svoje slojeve te kao izlaz daje sliku poput primjera za učenje. Mreža diskriminator ima zadatak odrediti da li je primjer na njenom ulazu generiran ili stvaran. Kroz postupke optimizacije težina, istovremeno se poboljšavaju generator i diskriminator.

Nije dugo trebalo da se ovaj model, kao i većina generativnih, pokuša preformulirati u polunadzirani model, te da u konačnici služi kao klasifikator. Pokazalo se da za manji broj ulaznih podataka model funkcionira bolje od izoliranog klasifikatora ([24]).

Upravo to je i cilj ovog rada koji ima zadatak dati uvid u najpoznatije inačice generativnih suparničkih modela, te isprobati model u polunadziranom okruženju na novoj bazi primjera za učenje.

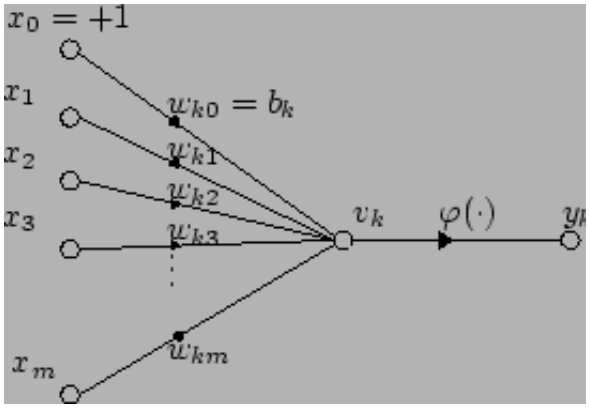
TODO: ŠTO U KOJEM POGLAVLJU

2. Neuronske mreže

Umjetne neuronske mreže nastale su kao simulacija rada ljudskog mozga. Skupina umjetnih neurona koji su međusobno povezani zajedno čine neuronsku mrežu. Svaki od neurona procesira informaciju na ulazu te daje izlaz. Grupiranje više neurona omogućava modeliranje složenijih relacija koji postoje u podacima na ulazu. Arhitektura neuronske mreže određuje način na koji su umjetni neuroni u istoj spojeni. Također, svaki od neurona može na više načina procesirati ulaze. Neuronske mreže stoga predstavljaju izrazito konfigurabilan model strojnog učenja, koji nalazi široku primjenu u polju umjetne inteligencije.

2.1. Umjetni neuron

Osnovni model umjetnog neurona postavili su autori u radu [22]. Sastoji se od ulaza u neuron koji su težinski povezani s istim, zatim se otežani ulazi zbrajaju, te se primjenjuje aktivacijska funkcija neurona, kao što je prikazano na slici 2.1.



Slika 2.1: Umjetni neuron [7]

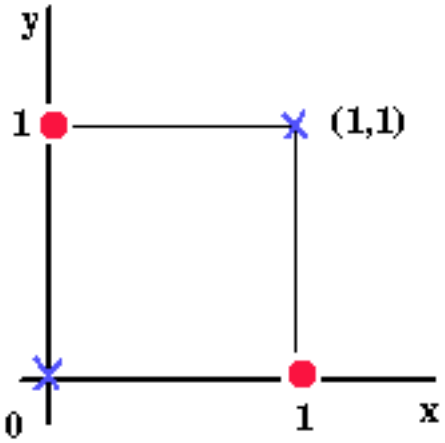
Pošto je svim neuronima zajedničko otežano zbrajanje ulaza, svojstva pojedinog neurona zapravo najviše ovise o njegovoj aktivacijskoj funkciji. O njima više u idućoj sekciji.

2.1.1. Aktivacijske funkcije

Aktivacijske funkcije su najznačajnija karakteristika svakog neurona, stoga ne čudi da napredak u razumijevanju pojedinih aktivacijskih funkcija, te predlaganje novih često dovodi do napretka u neuronskim mrežama.

Prag

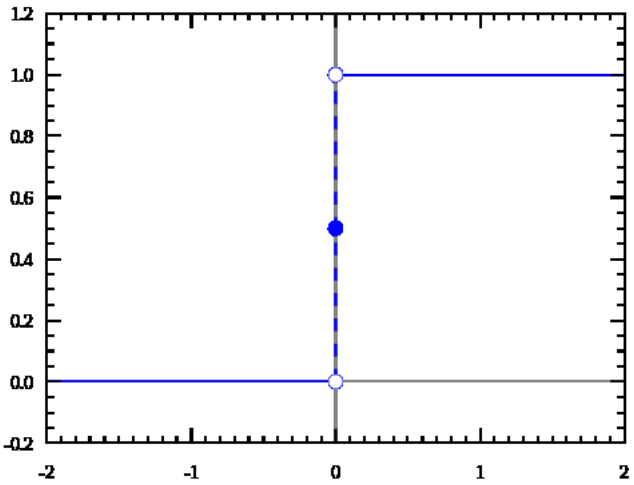
Funkcija praga predložena je u radu [25] gdje je opisan algoritam Perceptrona. Koristi se kao binarni klasifikator. Model Perceptrona je linearan, što znači da algoritam nije u mogućnosti ispravno klasificirati ni najjednostavnije linearno neseparabilne probleme poput prikazanog na slici 2.2. Dokaz za tu tvrdnju daju autori u knjizi [23].



Slika 2.2: XOR problem [13]

Nakon sumiranja otežanih ulaza, funkcija praga se primjenjuje po izrazu 2.1, a graf funkcije je vidljiv na slici 2.3.

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{2.1}$$



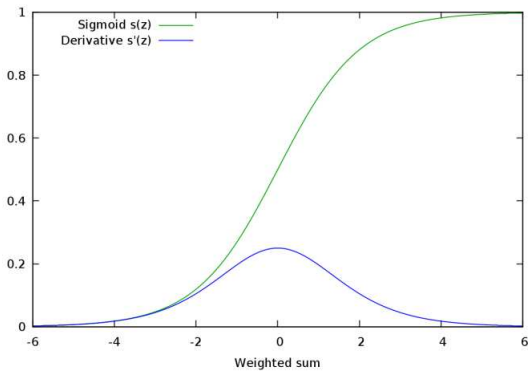
Slika 2.3: Funkcija praga [11]

Sigmoidalna funkcija

Sigmoidalna funkcija se prvotno koristila u logističkoj regresiji, te sama logistička regresija može bit interpretirana kao neuron kojem je aktivacijska funkcija sigmoidalna funkcija. Logistička regresija je također linearan klasifikator, no svoju značajnost za razliku od funkcije praga poprima u neuronskim mrežama.

Sigmoidalna funkcija je nelinearna aktivacijska funkcija. U neuronskim mrežama izlaz jednog neurona je ulaz drugome. Uvođenjem nelinearnosti preko aktivacijske funkcije omogućava se modeliranje kompleksnijih relacija od linearnih, te širenje primjene modela i na linearno neseparabilne probleme. Zadana je jednadžbom 2.2, a slika 2.4 prikazuje graf funkcije te graf njene derivacije.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$



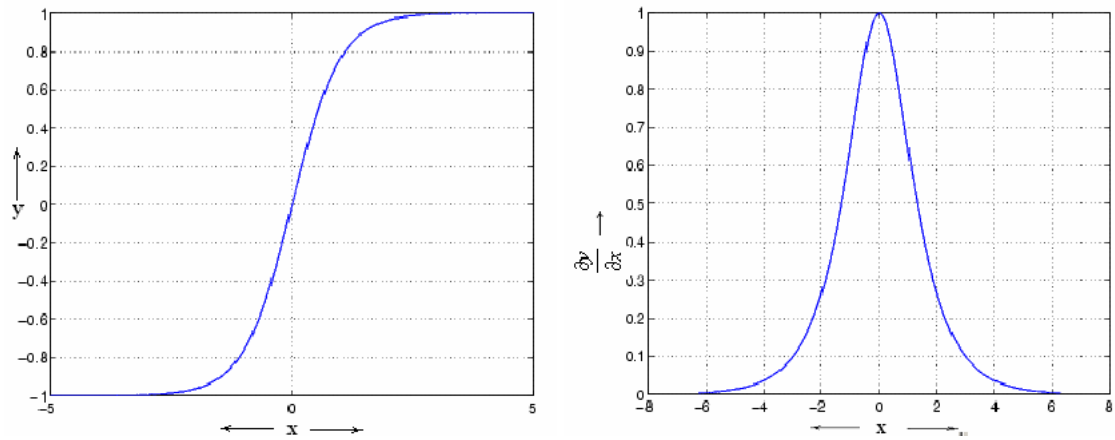
Slika 2.4: Sigmoidalna funkcija [10]

Iz grafa funkcije je vidljivo da se sigmoidalnom funkcijom dobije pouzdanost klasifikacije neurona za određeni primjer, što je korisna informacija. No, prva derivacija otkriva manu, a to je problem nestajućeg gradijenta. Ukoliko je izlaz neurona primjerice 0.99, odnosno klasa 1, a zapravo bi trebao biti klasa 0, gradijent je izrazito mali, što znači da će za korekciju klasifikacije trebati velik broj iteracija algoritma gradijentnog spusta.

Tangens hiperbolni

Tangens hiperbolni kao aktivacijska funkcija se počeo češće koristiti kao zamjena za sigmoidalnu funkciju iz razloga što ona daje isključivo pozitivne izlaze u intervalu $[0, 1]$ te je centrirana oko 0.5. To ne predstavlja problem samo jednom neuronu, dok u višeslojnoj neuronskoj mreži koči brzinu konvergencije u postupku učenja. Stoga se predlažu funkcije centrirane oko 0, a upravo jedna od takvih je tangens hiperbolni [21]. Funkcija je zadana jednadžbom 2.3, a graf funkcije i njene derivacije je vidljiv na slici 2.5.

$$f(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.3}$$



Slika 2.5: Tangens hiperbolni [12]

Autori u literaturi [21] također predlažu modificiranje aktivacijske funkcije u izraz:

$$f(x) = 1.7159tanh\left(\frac{2}{3}x\right). \tag{2.4}$$

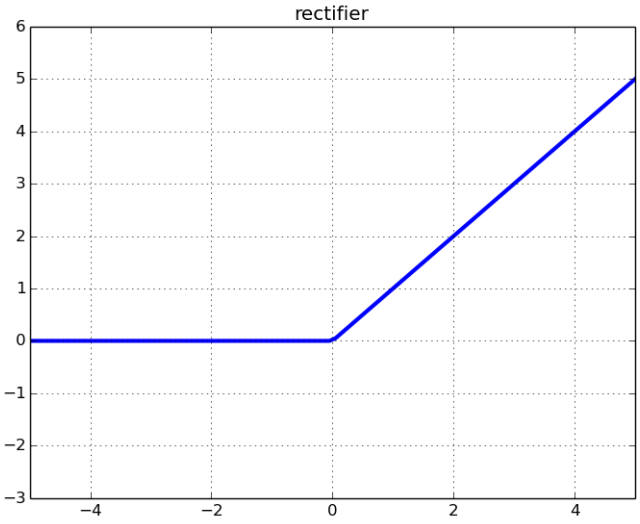
Modificirana funkcija za ulaze koji su centrirani na 0, sa varijancom 1, daje izlaze koji su također centrirani na 0 i varijanca bi im trebala biti blizu 1.

Rectifier

U kontekstu neuronskih mreža, rectifier je aktivacijska funkcija čija formula glasi:

$$f(x) = max(0, x) \tag{2.5}$$

Trenutno je najpopularnija aktivacijska funkcija u arhitekturama dubokih neuronskih mreža ([8]). Izračunski je vrlo jednostavna, te njen gradijent također (0 za $x < 0$, 1 za $x > 0$, nediferencijabilan u 0), dovodi i do rjeđih aktivacija neurona. Potencijalni problemi su: nije centrirana oko 0, nediferencijabilna u 0, neograničena aktivacija te se zna dogodit fenomen umirućih neurona, odnosno neuroni se dovedu u mrtvo stanje iz kojeg ne mogu više izaći, efektivno smanjujući kapacitet neuronske mreže [8]. Slika 2.6 prikazuje rectifier aktivacijsku funkciju.



Slika 2.6: Rectifier [9]

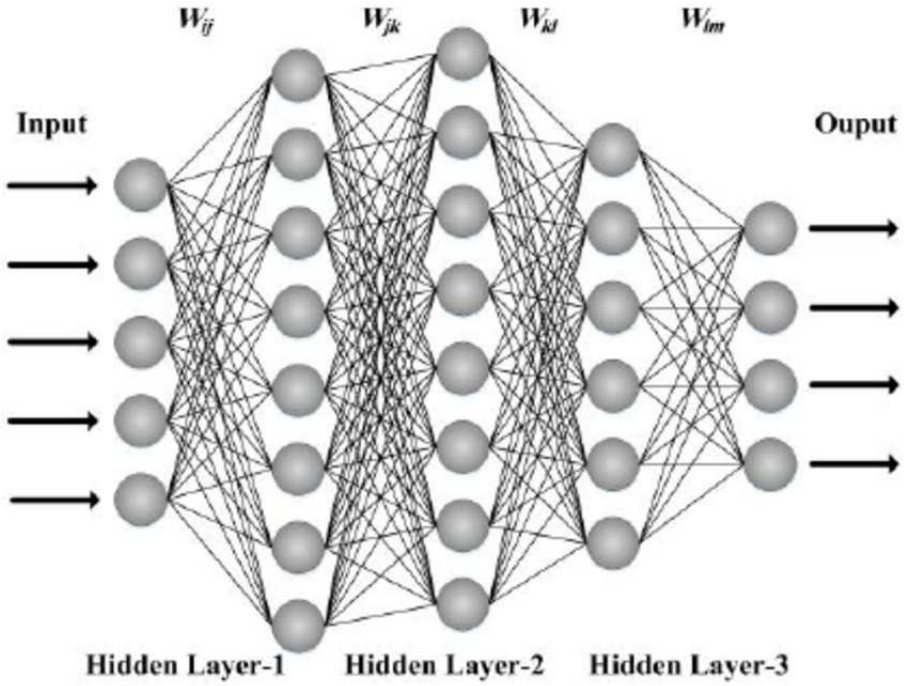
ReLU je lesto korištena skraćenica za rectifier. Poznate varijante su Parametric ReLU (2.6) i ELU (2.7) [8].

$$f(x) = \begin{cases} x, & x > 0 \\ ax, & otherwise \end{cases} \tag{2.6}$$

$$f(x) = \begin{cases} x, & x \geq 0 \\ a(e^x - 1), & otherwise \end{cases} \tag{2.7}$$

2.2. Višeslojni perceptron

Općenita unaprijedna neuronska mreža najčešće se naziva i višeslojnim perceptronom. Sadrži 3 vrste slojeva: ulazni, skriveni i izlazni. Ima 1 ulazni i 1 izlazni sloj, a skrivenih može biti po volji mnogo. Ilustracija jednog višeslojnog perceptrona dana je na slici 2.7.



Slika 2.7: Višeslojni perceptron [5]

Ulazni sloj predstavlja značajke podataka koje obrađujemo, na primjer za sliku, svaki pojedini neuron bi bio jedan piksel slike. U njima se ne vrši nikakva obrada već se samo proslijede prvom idućem skrivenom sloju. Najčešće su slojevi potpuno povezani, odnosno svaki neuron iz prethodnog sloja je povezan sa svakim neuronom u idućem sloju.

Skriveni slojevi vrše obradu sumacije otežanih ulaza te primjenu aktivacijske funkcije. Ključ leži u nelinearnosti aktivacijskih funkcija, u protivnom se proizvoljno dubok višeslojni perceptron svodi na najobičniji linearni perceptron. Broj skrivenih slojeva kao i broj neurona u skrivenim slojevima ovisi o problemu koji zadan, no općenito se može reći da neuronska mreža u skrivenim slojevima uči različite reprezentacije ulaznih podataka sa ciljem što uspješnijeg rješavanja problema. Rad [16] dokazuje da je neuronska mreža s barem jednim skrivenim slojem služi kao univerzalni funkcijski aproksimator, neovisno o izboru aktivacijske funkcije.

Izlazni sloj predstavlja izlaz iz mreže, pri klasifikacijskim problemima to bude vektor pouzdanosti pripadnosti ulaznog primjera određenom broju klasa. Za učenje parametara neuronske mreže koristi se algoritam unazadne propagacije.

2.2.1. Algoritam unazadne propagacije

Algoritam unazadne propagacije korigira parametre neuronske mreže s obzirom na gradijent funkcije pogreške. Gradijentni postupci u optimizaciji se koriste već stoljećima, pravilo ulančavanja gradijenata je također otprije poznato stoga je teško navesti sam početak primjene ovog postupka, pa čak i kod neuronskih mreža, no većina se slaže da je metoda popularizirana u radu [26].

Algoritam zahtijeva da su aktivacijske funkcije diferencijabilne, te da za zadane ulaze imamo i odgovarajuće izlaze, odnosno primjenjiv je samo u nadziranom učenju. Učenje se odvija u dvije faze:

1) Unaprijedni prolaz odnosno računanje aktivacija neurona za zadani ulaz sve do izlaza mreže, te posljedično računanje izlaza mreže za zadani ulaz.

2) Unazadni prolaz gdje se računa odstupanje izlaza mreže od traženog izlaza zadanog primjera, propagira se unazad kroz mrežu računajući grešku pri svakom od neurona, te naposlijetku se obavlja korekcija težina mreže jednom od gradijentnih metoda.

Ulančavanje gradijenata pokazat ćemo na jednostavnom primjeru mreže sa ulaznim, jednim skrivenim, te izlaznim slojem. Neka je zadano N primjera za učenje oblika $(x_\mathrm{n}, y_\mathrm{n})$. Neka mreža koristi sigmoidalnu aktivacijsku funkciju u skrivenom i izlaznom sloju, te minimizira funkciju srednje kvadratne pogreške

$$E = \frac{1}{2N}\sum_{n=1}^{N}(y_\mathrm{n} - o_\mathrm{n})^2 \tag{2.8}$$

gdje je o_n izlaz mreže za ulaz x_n. Za početak ćemo pokazati izraz za podešavanje parametara izlaznog sloja. Neka $\mathbf{W^{(i)}}$ označava matricu parametara izlaznog sloja, a $\mathbf{h_n}$ vektor izlaza srednjeg sloja za n-ti primjer, odnosno ulaza u izlazni sloj, kojemu je dodan član pristranosti 1. Tada derivacija funkcije greške po parametrima izlaznog sloja izgleda:

$$\frac{\partial E}{\partial \mathbf{W^{(i)}}} = \frac{\partial E}{\partial o_\mathrm{n}} \frac{\partial o_\mathrm{n}}{\partial net_\mathrm{n}{}^{(\mathrm{i})}} \frac{\partial net_\mathrm{n}{}^{(\mathrm{i})}}{\partial \mathbf{W^{(i)}}} \tag{2.9}$$

$$\frac{\partial E}{\partial o_\mathrm{n}} = -\frac{1}{N}\sum_{n=1}^{N}(y_\mathrm{n} - o_\mathrm{n}) \tag{2.10}$$

Pošto je o_{n} sigmoidalna funkcija, njena derivacija iznosi:

$$\frac{\partial o_{\mathrm{n}}}{\partial net_{\mathrm{n}}^{(\mathrm{i})}} = o_{\mathrm{n}}(1 - o_{\mathrm{n}}) \tag{2.11}$$

net označava ulazni sloj (u ovom slučaju skriveni sloj) pomnožen s matricom težina odnosno parametara, u ovom slučaju izlaznog sloja:

$$net_{\mathrm{n}}^{(\mathrm{i})} = \mathbf{W^{(i)}} h_{\mathrm{n}} \tag{2.12}$$

$$\frac{\partial net_{\mathrm{n}}^{(\mathrm{i})}}{\partial \mathbf{W^{(i)}}} = h_{\mathrm{n}} \tag{2.13}$$

Izračunavši sve potrebne parcijalne derivacije konačan izraz za gradijent izlaznog sloja glasi:

$$\frac{\partial E}{\partial \mathbf{W^{(i)}}} = -\frac{1}{N} \sum_{n=1}^{N} (y_{\mathrm{n}} - o_{\mathrm{n}}) o_{\mathrm{n}} (1 - o_{\mathrm{n}}) h_{\mathrm{n}} \tag{2.14}$$

Često se koristi i skraćena verzija zapisa:

$$\frac{\partial E}{\partial \mathbf{W^{(i)}}} = -\frac{1}{N} \sum_{n=1}^{N} \delta_{\mathrm{n}}^{(\mathrm{i})} h_{\mathrm{n}} \tag{2.15}$$

gdje je

$$\delta_{\mathrm{n}}^{(\mathrm{i})} = (y_{\mathrm{n}} - o_{\mathrm{n}}) o_{\mathrm{n}} (1 - o_{\mathrm{n}}) \tag{2.16}$$

Neka $\mathbf{W^{(h)}}$ predstavlja matricu parametara skrivenog sloja, a neka x_{n} predstavlja ulazni vektor u skriveni sloj, također proširen s članom pristranosti 1. Da bi dobili gradijent skrivenog sloja moramo se vratit korak unatrag:

$$\frac{\partial net_{\mathrm{n}}^{(\mathrm{i})}}{\partial \mathbf{W^{(h)}}} = \mathbf{W^{(i)}} \frac{\partial h_{\mathrm{n}}}{\partial \mathbf{W^{(h)}}} \tag{2.17}$$

Odnosno nastavljamo s parcijalnim deriviranjem unatrag kroz mrežu. Trenutno gradijent izgleda:

$$\frac{\partial E}{\partial \mathbf{W^{(i)}}} = -\frac{1}{N} \sum_{n=1}^{N} \delta_{\mathrm{n}}^{(\mathrm{i})} \mathbf{W^{(i)}} \frac{\partial h_{\mathrm{n}}}{\partial \mathbf{W^{(h)}}} \tag{2.18}$$

$$\frac{\partial h_{\mathrm{n}}}{\partial \mathbf{W^{(h)}}} = \frac{\partial h_{\mathrm{n}}}{\partial net_{\mathrm{n}}^{(\mathrm{h})}} \frac{\partial net_{\mathrm{n}}^{(\mathrm{h})}}{\partial \mathbf{W^{(h)}}} \tag{2.19}$$

Pošto je h_{n} izlaz sigmoidalne aktivacijske funkcija, derivacija glasi:

$$\frac{\partial h_{\mathrm{n}}}{\partial net_{\mathrm{n}}^{(\mathrm{h})}} = h_{\mathrm{n}}(1 - h_{\mathrm{n}}) \tag{2.20}$$

$$net_{\mathrm{n}}^{(\mathrm{h})} = \mathbf{W^{(h)}} x_{\mathrm{n}} \tag{2.21}$$

$$\frac{\partial net_{\mathrm{n}}{}^{(\mathrm{h})}}{\mathbf{W^{(h)}}} = x_{\mathrm{n}} \tag{2.22}$$

Izračunom svih potrebnih parcijalnih derivacija dobije se:

$$\frac{\partial h_{\mathrm{n}}}{\partial \mathbf{W^{(h)}}} = h_{\mathrm{n}}(1 - h_{\mathrm{n}})x_{\mathrm{n}} \tag{2.23}$$

Konačan gradijent parametara skrivenog sloja glasi:

$$\frac{\partial E}{\partial \mathbf{W^{(h)}}} = -\frac{1}{N}\sum_{n=1}^{N} \delta_{\mathrm{n}}{}^{(\mathrm{i})}\mathbf{W^{(i)}}h_{\mathrm{n}}(1 - h_{\mathrm{n}})x_{\mathrm{n}} \tag{2.24}$$

ili skraćeno:

$$\frac{\partial E}{\partial \mathbf{W^{(h)}}} = -\frac{1}{N}\sum_{n=1}^{N} \delta_{\mathrm{n}}{}^{(\mathrm{h})}x_{\mathrm{n}} \tag{2.25}$$

gdje je

$$\delta_{\mathrm{n}}{}^{(\mathrm{h})} = \delta_{\mathrm{n}}{}^{(\mathrm{i})}\mathbf{W^{(i)}}h_{\mathrm{n}}(1 - h_{\mathrm{n}}) \tag{2.26}$$

Ulančavanje se nastavlja po istom principu i za više slojeva. Učenje se vrši optimizacijskim postupcima koji se temelje na gradijentima. Nakon što smo pokazali kako se računaju gradijenti s obzirom na funkciju pogreške neuronske mreže, slijedi pregled odabranih optimizacijskih metoda.

Gradijentni spust

Gradijentni spust je osnovna numerička metoda minimizacije funkcije pogreške u strojnom učenju. Promatramo funkciju pogreške kao funkciju parametara algoritma, te optimiziramo parametre algoritma u smjeru minimalne greške te funkcije. Gornji primjer pokazuje kako izračunati gradijente parametara, dok se podešavanje parametara odvija po idućoj formuli:

$$\mathbf{W^{(n+1)}} = \mathbf{W^{(n)}} - \eta\frac{\partial E}{\partial \mathbf{W^{(n)}}} \tag{2.27}$$

gdje je $n + 1$ vremenska oznaka buduće vrijednosti, n trenutne, a η nazivamo stopom učenja, ona određuje koliki će biti pomak u smjeru gradijenta. Kao i svi numerički postupci, stabilnost ovog postupka ovisi o odabiru vrijednosti η.

Postoje tri različita načina osvježavanja parametara. Stohastički gradijentni spust se temelji na tome da se za svaki primjer izračunaju gradijenti i odma osvježe parametri modela. Grupni gradijentni spust zbroji gradijente svih primjera, uzme prosjek tako da zbroj podijeli s brojem primjera te onda osvježi parametre modela. Treći način je kompromisni, ujedno i najkorišteniji, u kojem se odredi nekakav n koji je veličina grupe

nakon koje će model osvježavati parametre. Stohastički gradijentni spust je vremenski skup radi čestog osvježavanja parametara, no otporniji je na lokalne optimume. Grupni gradijentni spust radi izrazito dobro pri konveksnim problemima, no ne i u onim gdje krivulje funkcije pogreške imaju mnogo lokalnih optimuma. Također je memorijski zahtjevno koristiti sve primjere za učenje odjednom. Upravo zato je i najkorišteniji kompromisni način, jer veličinom grupe kontroliramo prednosti i mane oba načina. Postupak optimizacije se vrši do zadovoljavanja zadanih kriterija konvergencije ili do isteka postavljenog maksimalnog broja iteracija.

Momentum

Momentum metoda osim gradijenta uzima u obzir i gradijent u prethodnom koraku. Cilj je dobiti konzistentniji smjer gradijenta, dajući mu "širu sliku" kretnje po hiperravnini funkcije pogreške, a ne samo trenutno stanje. Predložena je kao poboljšanje nad stohastičkim gradijentnim spustom u radu [26].

$$\Delta W^{(\mathrm{n})} = \eta \frac{\partial E}{\partial \mathbf{W^{(n)}}} + \alpha \Delta W^{(\mathrm{n\text{-}1})} \tag{2.28}$$

$$W^{(\mathrm{n+1})} = W^{(\mathrm{n})} - \Delta W^{(\mathrm{n})} \tag{2.29}$$

RMSProp

RMSProp (engl. *Root Mean Square Propagation*) je varijanta gradijentnog spusta s dodanim parametrom. Uz stopu učenja η, postoji i stopa zaboravljanja γ. Osnovna ideja je da model na početku brzo podešava težine, no što je bliži konvergenciji gradijenti su manji stoga je i proces učenja sporiji. RMSProp to mijenja tako što modificira stopu učenja obrnuto proporcionalno magnitudi gradijenta. Optimizacijski postupak je predložen u literaturi [28].

$$v^{(\mathrm{n})} = \gamma v^{(\mathrm{n\text{-}1})} + (1-\gamma)\left(\frac{\partial E}{\partial \mathbf{W^{(n)}}}\right)^2 \tag{2.30}$$

Po izrazu 2.28 modificiramo stopu učenja. $v^{(\mathrm{n})}$ označava magnitudu gradijenta u n-tom koraku, a pri tom koristi i magnitudu gradijenta u $n-1$ koraku, sve zajedno parametrizirano s γ. Sada postupak osvježavanja parametara izgleda:

$$\mathbf{W^{(n+1)}} = \mathbf{W^{(n)}} - \frac{\eta}{\sqrt{v^{(\mathrm{n})}}} \frac{\partial E}{\partial \mathbf{W^{(n)}}} \tag{2.31}$$

Adam

Adam (engl. *Adaptive Moment Estimation*) je kombinacija Momentum i RMSProp metoda. Postupak je prezentiran u radu [18], gdje autori detaljnije izvode izraza u algoritmu, pružaju eksperimentalne rezultate kao dokaz te predlažu odabir parametara za koje smatraju da su robusni i široko primjenjivi.

$$m_{\mathrm{w}}{}^{(\mathrm{n+1})} = \beta_1 m_{\mathrm{w}}{}^{(\mathrm{n})} + (1-\beta_1)\frac{\partial E}{\partial \mathbf{W}^{(\mathbf{n})}} \tag{2.32}$$

$$v_{\mathrm{w}}{}^{(\mathrm{n+1})} = \beta_2 v_{\mathrm{w}}{}^{(\mathrm{n})} + (1-\beta_2)\left(\frac{\partial E}{\partial \mathbf{W}^{(\mathbf{n})}}\right)^2 \tag{2.33}$$

$$\hat{m}_{\mathrm{w}}{}^{(\mathrm{n})} = \frac{m_{\mathrm{w}}{}^{(\mathrm{n})}}{1-\beta_1^n} \tag{2.34}$$

$$\hat{v}_{\mathrm{w}}{}^{(\mathrm{n})} = \frac{v_{\mathrm{w}}{}^{(\mathrm{n})}}{1-\beta_2^n} \tag{2.35}$$

$$\mathbf{W}^{(\mathbf{n+1})} = \mathbf{W}^{(\mathbf{n})} - \eta\frac{\hat{m}_{\mathrm{w}}{}^{(\mathrm{n})}}{\sqrt{\hat{v}^{(\mathrm{n})}}+\epsilon} \tag{2.36}$$

Iz gornjih relacija očigledno je da postupak ima 4 parametra. Parametri β_1 i β_2 su faktori zaboravljanja za prvi, odnosno drugi moment gradijenata, ϵ je mali korekcijski faktor za slučaj izbjegavanja dijeljenja s 0 i η koja je stopa učenja kao i u prethodnim gradijentnim postupcima. Relacije 2.32 i 2.33 daju pristrane procjenitelje prvog i drugog momenta gradijenta stoga se korigiraju za faktor pristranosti, pa se dobiju nepristrani procjenitelji iz relacija 2.34 i 2.35.

2.3. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže, za razliku od višeslojnih perceptrona, koriste konvolucijske slojeve o kojima će biti govora kasnije u ovoj sekciji. Prva takva zabilježena mreža nalazi se u literaturi [14], no za popularizaciju istih uglavnom se navodi rad [19]. Najveći napredak su postigle u polju računalnog vida, gdje su pri nekim problemima postigle i nadljudske rezultate.

2.3.1. Konvolucija

U obradi slike koristimo 2D konvolucijski operator nad diskretnim vrijednostima. Stoga operator 2D diskretne konvolucije se definira preko idućeg izraza:

$$f(x,y) * g(x,y) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f(n_1,n_2)g(x-n_1,y-n_2) \tag{2.37}$$

Na prvi pogled jednostavan izraz, oko kojeg je teško razviti intuiciju [1]. No, gornji izraz je općenit slučaj za konvoluciju dva dvodimenzionalna signala, na primjerima ćemo vidjeti da se u obradi slike radi o mnogo jednostavnijem slučaju. Slika 2.8 prikazuje 2 često korištena Laplace filtra.

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Slika 2.8: Laplace filter [2]

Slika 2.9: Primjer djelovanja Laplace filtra na sliku [3]

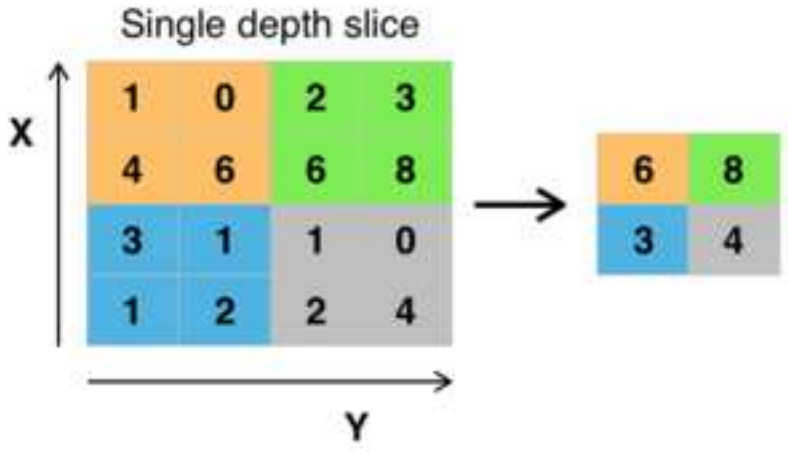
Konvolucijskim prozorom se prolazi kroz cijelu sliku, sumirajući težinski pikselne vrijednosti, te na izlazu dobijemo obrađenu sliku. Primjer djelovanja Laplace filtra dan je na slici 2.9.

Upravo ideja da se koriste konvolucijski operatori s filtrima koji su naučeni, a ne predodređeni, omogućile su neuronskim mrežama izrazit uspjeh u polju računalnog vida. Pri obradi slike to je od iznimne važnosti jer se postiže lokalna osjetljivost i dijeljenje parametara, odnosno model sam nauči mapiranje iz slike u prostor značajki koji je kasnije povezan na potpuno povezani sloj najčešće, ili nekakav drugi klasifikator.

Korištenje konvolucijskih slojeva omogućava slojevitiju mrežu, pošto su parametri dijeljeni, problem naglog rasta broja parametara sa slojevitošću je manji nego kod potpuno povezanih slojeva višeslojnog perceptrona. Stoga se konvolucijski slojevi bliže ulaznoj slici najčešće fokusiraju na detekciju značajki niske razine poput rubova i linija, dok kasniji konvolucijski slojevi modeliraju značajke viših razina poput razumijevanja scene i objekata.

2.3.2. Slojevi sažimanja

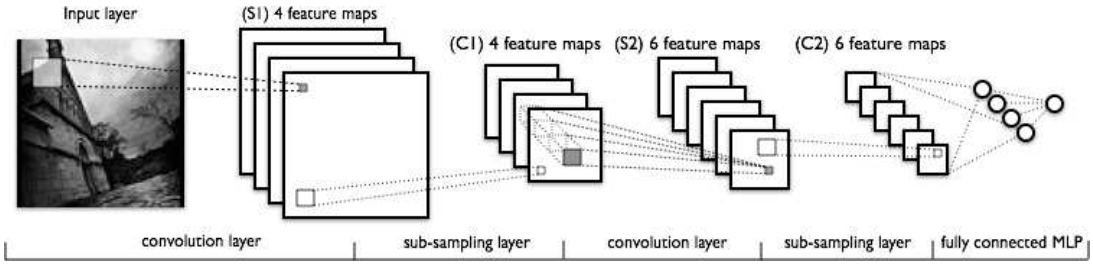
Uz konvolucijske slojeve često se u arhitekturama konvolucijskih neuronskih mreža mogu pronaći i slojevi sažimanja. Najčešće korišten operator je sažimanje po maksimalnom odzivu, no postoji sažimanje po minimalnom i prosječnom odzivu. Slojevi sažimanja općenito primaju sliku te zatim po regjiama slike primjenjuju odabrani operator. Postižu smanjenje dimenzionalnosti, posljedično i smanjenje broja parametara modela što spriječava prenaučenost.



Slika 2.10: Primjer sažimanja po maksimalnom odzivu [4]

2.3.3. Arhitektura konvolucijskih neuronskih mreža

Konvolucijske neuronske mreže primaju 2D ulaze poput slike, provlače ih kroz prvi konvolucijski sloj, nakon što se dobije prva mapa značajki, neuroni konvolucijskog sloja primjenjuju aktivacijske funkcije te se nakon toga vrši sloj sažimanja. Taj proces se ponavlja ovisno o tome koliko arhitektura mreže ima konvolucijskih slojeva. Izlaz zadnjeg konvolucijskog sloja je također mapa značajki koja se zatim "izravna" te se predaje potpuno povezanim slojevima.



Slika 2.11: Primjer arhitekture konvolucijske neuronske mreže [6]

Autori u radu [27] predlažu pojednostavljenja u arhitekturama konvolucijskih mreža. Predlažu izbacivanje slojeva sažimanja te nadomještajući to proširenjem koraka konvolucijskih filtara.

U zadnje vrijeme počeo se koristi pojam prijenosa učenja (engl. *transfer learning*). Odnosi se na to da se mreža naučena nad jednim skupom podataka primjenjuje i nad drugim skupovima podataka. Često samo zahtijeva modifikacije nad posljednjim klasifikacijskim slojem jer se skupovi podataka razlikuju u broju klasa. Proces treniranja je isti, no praksa je pokazala da značajke naučene na skupom A služe bolje od učenja nove mreže s incijaliziranim težinama. Stoga nije ni čudo da dosada već postoji par standardnih, pretreniranih arhitektura koje se dalje prilagođavaju nad ostalim skupovima podataka.

2.4. Grupna normalizacija

Grupna (engl. *batch*) normalizacija je metoda predstavljena u literaturi [17]. Motivacija je parametrizirano normalizirati ulaze u svaki sloj neuronske mreže pošto se distribucije ulaza u slojeve mijenjaju kroz treniranje. Gore navedeni problem povlači i druge probleme poput opreznog postavljanja stope učenja te inicijalizacije parametara mreže. Grupna normalizacija do neke mjere eliminira te probleme. Grupe se normaliziraju po sljedećim izrazima:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.38}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{2.39}$$

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.40}$$

$$y_i = \gamma\hat{x_i} + \beta \tag{2.41}$$

Relacija 2.38 računa prosječnu vrijednost grupe, relacija 2.39 računa varijancu grupe. Koristeći oboje, podaci iz grupe se normaliziraju na prosječnu vrijednost 0 s varijancom 1 s izrazom 2.40. Izraz 2.41 nudi 2 parametra, γ i β koji mogu skalirati i pomaknuti distribuciju. To su optimizirajući parametri, odnosno optimizacijski postupci traže idealne γ i β za svaki sloj. y_i predstavlja izlaz iz grupe normalizacije, odnosno transformirani ulaz.

Metoda se ispostavila od značajne važnosti jer ulazi kasnijih slojeva više ne ovise o magnitudama prijašnjih slojeva pošto se vrši parametrizirana normalizacija. Autori su eksperimentima pokazali da postupak dovodi do brže konvergencije i boljih rezultata, te se grupna normalizacija smatra neizostavnim dijelom većine modernih arhitektura dubokih neuronskih mreža.

3. Generativne suparničke mreže

4. Zaključak

Zaključak.

LITERATURA

[1] Konvolucija. https://graphics.stanford.edu/courses/cs178/applets/convolution.html. Accessed: 2017-05-25.

[2] Laplace filter. http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm, . Accessed: 2017-05-25.

[3] Primjer laplace filtra. https://docs.gimp.org/en/plug-in-laplace.html,. Accessed: 2017-05-25.

[4] Primjer sažimanja po maksimalnom odzivu. https://en.wikipedia.org/wiki/Convolutional_neural_network. Accessed: 2017-05-25.

[5] Višeslojni perceptron. https://www.researchgate.net/figure/287209604_fig4_Figure-4-Multilayer-perceptron-neural-network-mode Accessed: 2017-05-20.

[6] Primjer arhitekture konvolucijske neuronske mreže. http://deeplearning.net/tutorial/lenet.html. Accessed: 2017-05-25.

[7] Artificial neuron. https://en.wikipedia.org/wiki/Artificial_neuron. Accessed: 2017-05-15.

[8] Rectifier. https://en.wikipedia.org/wiki/Rectifier_(neural_networks),. Accessed: 2017-05-16.

[9] Rectifier graph. https://datascience.stackexchange.com/questions/5706/what-is-the-dying-relu-problem-in-neural-networks, . Accessed: 2017-05-16.

[10] Sigmoidalna funkcija. http://whiteboard.ping.se/MachineLearning/BackProp. Accessed: 2017-05-16.

[11] Funkcija praga. https://en.wikipedia.org/wiki/Heaviside_step_function. Accessed: 2017-05-16.

[12] Tangens hiperbolni. https://www.researchgate.net/figure/260677487_fig12_Figure-13-a-ytanhx-b-Derivative-of-y. Accessed: 2017-05-16.

[13] Xor problem. https://faculty.iiit.ac.in/~vikram/nn_intro.html. Accessed: 2017-05-16.

[14] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130, 1988.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, i Yoshua Bengio. Generative adversarial nets. U *Advances in neural information processing systems*, stranice 2672–2680, 2014.

[16] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[17] Sergey Ioffe i Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[18] Diederik Kingma i Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Yann LeCun i Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[20] Yann LeCun, Corinna Cortes, i Christopher JC Burges. The mnist database of handwritten digits, 1998.

[21] Yann A LeCun, Léon Bottou, Genevieve B Orr, i Klaus-Robert Müller. Efficient backprop. U *Neural networks: Tricks of the trade*, stranice 9–48. Springer, 2012.

[22] Warren S McCulloch i Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[23] Marvin Minsky i Seymour Papert. Perceptrons. 1969.

[24] Augustus Odena. Semi-supervised learning with generative adversarial networks. *arXiv preprint arXiv:1606.01583*, 2016.

[25] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[26] David E Rumelhart, Geoffrey E Hinton, i Ronald J Williams. Learning internal representations by error-propagation. U *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1*, svezak 1, stranice 318–362. MIT Press, Cambridge, MA, 1986.

[27] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, i Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[28] Tijmen Tieleman i Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

Polunadzirana klasifikacija rukom pisanih znakova generativnim suparničkim modelima

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.