SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1147

Semantička segmentacija prirodnih scena dubokim neuronskim mrežama

Ivan Borko

Zagreb, lipanj 2015.

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

SADRŽAJ

1. Uvod

Razumijevanje scene osnovni je problem mnogih segmenata računalnog vida. Jedan od načina na koji se može ostvariti razumijevanje scene je semantička segmentacija, odnosno proces pridjeljivanja semantičkih oznaka dijelovima slike. Primjeri oznaka su: nebo, more, livada, stablo, automobil, osoba, zgrada. Scena je obično predstavljena digitalnom fotografijom pa onda govorimo o pridjeljivanju oznaka svakom pikselu.

Razumijevanje scene bitan je dio mnogih autonomnih robotiziranih sustava koji čine interakcije s okolinom. Primjer su autonomni automobili koji su u zadnje vrijeme postigli veliki razvojni napredak pa često svjedočimo reportažama o vožnjama ili pak testovima novih prototipova. Takvi automobili koriste sustave za razumijevanje scene kako bi pratili cestu kojom voze, pratili druge sudionike u prometu i mogli izbjeći potencijalne nesreće.

Ovaj rad opisuje sustav za semantičku segmentaciju koji koristi već segmentirane scene za "učenje" svojih parametara. Arhitektura sustava se temelji na dubokim neuronskim mrežama, konkretno konvolucijskim neuronskim mrežama. Sustavi temeljeni na konvolucijskim mrežama postižu najbolje rezultate (engl. *state-of-the-art*) u mnogim zadacima računalnog vida poput klasifikacije ([9], [21], [19], [17]), detekcije objekata ([6]), fine kategorizacije (engl. *fine grained categorization*)([24]) i semantičke segmentacije ([4], [12], [3]).

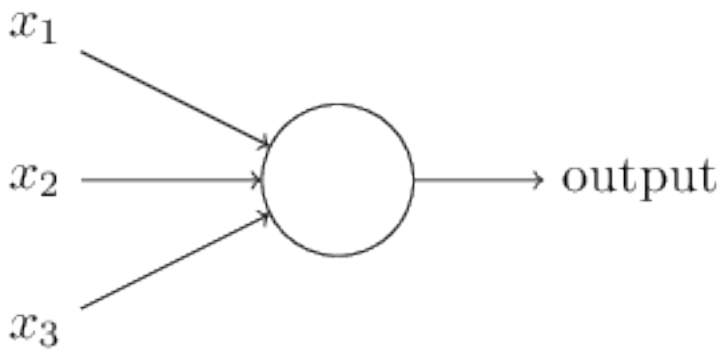
Do prije četiri godine godine konvolucijske mreže nisu bile glavni alat za semantičku segmentaciju. Tada su se uglavnom koristile razne metode temeljene na grafovima i grupiranju piksela u skupine. U takvim postupcima često se za svaku regiju od 8x8 piksela računaju HOG (engl. *histogram of gradients*) ili SIFT (engl. *scale invariant feature transform*) opisnici [13]. Ti opisnici predstavljaju značajku te regije i ulaze u klasifikator koji ih zatim grupira i klasificira. Kada se koriste klasifikatori temeljeni na stablima odluka tada klasifikator ima svoju energetsku funkciju koju optimira tako da je slika optimalno pokrivena semantičkim oznakama.

TODO: Opisi poglavlja...

2. Neuronske mreže

Umjetne neuronske mreže (engl. *artificial neural networks*) ili skraćeno neuronske mreže su paradigma koja opisuje procesiranje informacija inspirirano načinom na koji ljudski mozak (promatran samo kao skup neurona) procesira informacije, odnosno električne impulse. Glavna odlika neuronske mreže je procesiranje informacija jednostavnim elementima (neuronima) na visokoj razini paralelizma. Ti jednostavni elementi su međusobno povezani velikim brojem veza (u biološkom sustavu te veze su sinapse). Učenje neuronskog biološkog sustava podrazumijeva podešavanje sinapsi, a slično tome se podešavaju parametri u umjetnoj neuronskoj mreži.

Osnovni element umjetne neuronske mreže je umjetni neuron. Prvi umjetni neuron, nazvan perceptron, razvio je Frank Rosenblatt 1950-ih inspiriran ranijim radom [15] Warrena McCullocha i Waltera Pittsa iz 1943. o njihovom viđenju umjetnog neurona. Taj početni entuzijazam nije predugo trajao jer Minsky i Papert 1969. godine objavljuju knjigu [16] u kojoj navode mane neuronskih mreža i daju pesimističan pogled na neuronske mreže. Tada većina znanstenika prestaje s istraživanjima na tome polju i razvoj ostaje na par znanstvenika koji su trebali dokazati mogućnosti neuronskih mreža.



Slika 2.1: Primjer perceptrona s 3 ulaza

Slika 2.1 prikazuje primjer perceptrona koji ima 3 ulaza, x_1, x_2, x_3. Općenito, broj ulaza je proizvoljan. Rosenblatt je predstavio pravilo kako se računa izlaz iz neurona:

postoje težine $w_1, w_2, ...$, realni brojevi koji predstavljaju značaj pripadajućih ulaza. Izlaz neurona se određuje prema tome je li suma $\sum_j w_j * x_j$ veća ili manje od neke fiksne vrijednost. Matematički se to može zapisati kao:

$$izlaz = \begin{cases} 0 & \text{ako } \sum_j w_j * x_j \leq prag \\ 1 & \text{ako } \sum_j w_j * x_j > prag \end{cases} \tag{2.1}$$

Učenje takvog modela se zapravo svodi na podešavanje težina w_j i praga, čime se postižu različite vrijednosti izlaza.

Današnji umjetni neuron koji se koristi u neuronskim mrežama se malo razlikuje od originalne ideje perceptrona po tome što ne koristi step funkciju na izlazu, već neku funckciju iz porodice sigmoidalnih funkcija (neki autori takav neuron nazivaju logistički neuron ili logistički perceptron).

No, nije samo rad Minskog i Paperta kočio razvoj neuronskih mreža: računalna snaga je tada bila veliko ograničenje. Zato su se proučavali tek jednostavni modeli, koji nisu imali dovoljnu moć. Prava moć dolazi tek kada se više perceptrona spoji zajedno kroz više slojeva (vidi Višeslojni perceptron). Tokom 80-ih godina dolazi do razvoja računalnog paralelizma i povećanja računalne snage čime rastu mogućnosti za treniranje i evaluacije kompleksnijih neuronskih mreža.

Značajan napredak postigao je i Werbos 1975. godine kada je u svojem radu [22] primjenio algoritam unazadne propagacije (engl. *backpropagation*) za učenje neuronskih mreža (vidi poglavlje Backgropagation). Često se taj trenutak naziva i renesansom neuronskim mreža jer opisani postupak omogućava lako treniranje neuronskih mreža neovisno o njihovoj složenosti i arhitekturi.

Za to vrijeme snažno su se razvijali drugi algoritmi strojnog učenja. Stroj s potpornim vektorima (engl. *support vector machine*), ili kraće SVM, tako postaje moćniji i zastupljeniji klasifikator od neuronskih mreža. Tek zadnjih 10-ak godina duboke neuronske mreža polako preuzimaju primat SVM-a zbog dolaska GPGPU (engl. *general purpose graphics processsing unit*) jedinica koje su sposobne izvršavati velike količine paraleliziranih matričnih operacija potrebnih za treniranje dubokih mreža.

2.1. Klasifikacija i regresija

Dva temeljna problema prediktivnih sustava su klasifikacija i regresija. Klasifikacija je razvrstavanje ulaznih podatka u jedan od K razreda. Ako je $K = 2$ onda govorimo o binarnoj klasifikaciji. Kod neuronskih mreža se to može ostvariti postavljanjem step

funkcije na na kraj zadnjeg neurona čime izlaz postaje 0 ili 1. Za razliku od klasifikacije, regresija daje realni broj kao izlaz. Cilj je da realni izlaz bude čim sličniji traženom izrazu.

2.2. Nadzirano i nenadzirano učenje

Dva su glavna pristupa učenju prediktivnih modela koja vrijede i za neuronske mreže: nadzirano i nenadzirano učenje. Tokom nadziranog učenja mreži prezentiramo par podataka $(x, y), x \in X, y \in Y$, gdje je X skup svih ulaza u mrežu, a Y skup pripadnih izlaznih (traženih) vrijednosti. Cilj je pronaći funkciju $f : X \to Y$, koja aproksimira preslikavanje implicirano u podacima za učenje.

Parametri modela se optimiraju tako da se smanjuje zadana funkcija troška (engl. *cost function*). Funkcija troška može biti srednja kvadratna pogreška između traženih i dobivenih vrijednosti na izlazu (većinom korištena za regresijske probleme),

$$\mathcal{L} = (y_t - y_p)^2 \tag{2.2}$$

ili pak pogreška unakrsne entropije (većinom korištena za klasifikaciju)

$$\mathcal{L} = -(y_t \log(y_p) + (1 - y_t) \log(1 - y_p)) \tag{2.3}$$

gdje je y_t je tražena vrijednost na izlazu, a y_p je dobivena (predviđena) vrijednost.

Tokom nenadziranog učenja nema izlaznog skupa podataka, samo ulazni skup X. Dana je i funkcija troška koja se optimira. Cilj može biti grupiranje podataka u K grupa pa je tada funkcija troška određena mjera disperzije uzoraka unutar grupe. Ili se pak traži kompresija ulaznog višedimenzionalnog vektora u manje dimenzionalan pa je funkcija troška zapravo pogreška rekonstrukcije.

2.3. Logistička regresija

Originalni perceptron se razvijao kroz povijest i njegova verzija koja se trenutno koristi u neuronskim mrežama izgleda malo drugačije. Ulazi više ne moraju biti binarne vrijednosti, već to mogu biti realne vrijednosti, a funkcija na izlazu je sigmoid (logistička funkcija), definirana formulom:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

Takav se model naziva logistička regresija. Logistička regresija je jedan od osnovnih klasifikacijskih modela, iako joj ime govori da je to regresijski model, što ona nije.

Logistička regresija je probabilistički klasifikator jer njezin izlaz predstavlja vjerojatnost da je primjer u određenom razredu. Ona je i linearni klasifikator jer je decizijska granica linearna[25]. Kod jednostavne (dvoklasne) klasifikacije parametri logističke regresije su vektor težina $\boldsymbol{w}$ i pomak $\boldsymbol{b}$. Ako govorimo o višeklasnoj klasifikaciji, parametri logističke regresije su matrica težina $\boldsymbol{W}$ i vektor pomaka $\boldsymbol{b}$.

Da bi se shvatilo opravdanje za vjerojatnosni izlaz logističke regresije, potrebno je pokazati izvod funkcije gubitka. Želimo da logistička regresija modelira vjerojatnosnu distribuciju primjera iz skupa $\mathcal{D}$. Kako bi se to postiglo, potrebno je podesiti težine i pomak. Uvodimo oznaku θ, koja označava skup svih parametara modela (težine i pomak). Koristeći tu notaciju, možemo reći da modeliramo vjerojatnost primjera uvjetovanu skupom parametara $P\left(x|\theta\right)$

Pošto nam je već poznata distribucija primjera $\mathcal{D}$, a zanimaju nas parametri θ, uvodimo termin izglednost (engl. *likelihood*), koju ćemo označiti simbolom $\mathcal{L}$. Izglednost parametara uvjetovana primjerom jednaka je vjerojatnosti tog primjera uvjetovanoj parametrima:

$$\mathcal{L}(\theta|x) = P(x|\theta)$$

Razmotrimo značenje predočenog izraza. Poanta nije u procjeni točne vjerojatnosti, već u ideji da ako mjenjamo parametre θ tako da oni povećavaju vjerojatnost pojavljivanja primjera x (desna strana izraza), tada time istovremeno povećavamo izglednost da su parametri θ baš oni koje tražimo (lijeva strana).

Ista ideja primjenjiva je na cijeli skup primjera za učenje $\mathcal{D}$. Pretpostavimo da su primjeri za učenje uzorkovani iz samo jedne distribucije. Pretpostavimo nadalje da su uzorci međusobno nezavisni . Tada govorimo o podatcima koji su nezavisni i identično distribuirani, za što se u literaturi često koristi akronim *iid* (engl. *independent, identically distributed*). Ako je zadovoljen *iid* uvjet, možemo definirati vjerojatnost cijelog skupa za učenje $\mathcal{D}$:

$$P(\mathcal{D}) \overset{iid}{=} P(x) \tag{2.5}$$

Iz dobivene vjerojatnosti $P(\mathcal{D})$, možemo definirati izglednost parametara s obzirom na cijeli skup za učenje $\mathcal{D}$:

$$\mathcal{L}(\theta|D) = P(D|\theta) \overset{iid}{=} \prod_{x \in D} P(x|\theta) \tag{2.6}$$

Pošto u pravilu znamo za vjerojatnost $P(x|\theta)$, promjenom parametara θ možemo mijenjati vjerojatnost $P(\mathcal{D}|\theta)$. Tražimo skup parametara koji dotičnu vjerojatnost maksimizira. Za logističku regresiju to bi značilo da smo pronašli težine ($\boldsymbol{w}$) i pomak

(b) koji dobro modeliraju skup primjera $\mathcal{D}$. Za pronalazak maksimuma vjerojatnosti $P(\mathcal{D}|\theta)$ koristimo gradijentni uspon.

Preostaje i za objasniti često korišteni trik kod maksimizacije vjerojatnosti. Razmotrimo logaritamsku funkciju $ln(x)$. Ona je rastuća na cijeloj domeni: povećanjem argumenta x, povećava se vrijednost funkcije $ln(x)$. Pošto se računica izglednosti odvija na intervalu $P(x) \in [0, 1]$, unutar domene logaritamske funkcije (izuzevši rubnu vrijednost 0, što nije problem), znamo da povećanjem $P(x)$ raste i $ln(P(x))$. Stoga umjesto maksimizacije izglednosti možemo koristiti maksimizaciju log-izglednosti:

$$ln\mathcal{L}(\theta|D) = lnP(D|\theta) \stackrel{iid}{=} ln \prod_{x \in D} P(x|\theta) = \sum_{x \in D} lnP(x|\theta) \tag{2.7}$$

Maksimizacijom log-izglednosti pronalazimo isti skup parametara θ kao i maksimizacijom obične izglednosti, ali matematički izračun često bude jednostavniji.

Vratimo se sada na logističku regresiju. Za ulazni skup podataka, možemo definirati log izglednost $\mathcal{L}$ i gubitak ℓ:

$$\mathcal{L}(\theta|D) = \sum_{i=0}^{|D|} ln(P(Y = y^{(i)}|x^{(i)}, \boldsymbol{w}, b)) \tag{2.8}$$

$$\ell(\theta|\mathcal{D}) = -\mathcal{L}(\theta|\mathcal{D}) = -\sum_{x \in D} lnP(x|\theta) \tag{2.9}$$

gdje je $\theta = \{\boldsymbol{w}, b\}$.

Klasifikacija se vrši tako da se određuje projekcija točaka u n-dimenzionalnom prostoru na normalu hiperravnine koja predstavljaja granicu između razreda uzoraka.

Matematički se to može zapisati kao:

$$P(Y = 1|\boldsymbol{x}, \boldsymbol{w}, b) = f(\boldsymbol{w} * \boldsymbol{x} + b) = \frac{1}{1 + e^{-(\boldsymbol{w} * \boldsymbol{x} + b)}} \tag{2.10}$$

gdje je f sigmoidalna funkcija, $\boldsymbol{x}$ ulazni vektor, a Y je skup izlaznih vrijednosti (1 znači da pripada nekom razredu).

Odnosno kao

$$P(Y = i|\boldsymbol{x}, \boldsymbol{w}, b) = \text{softmax}(\boldsymbol{w} * \boldsymbol{x} + b) = \frac{e^{\boldsymbol{x}^\top * \boldsymbol{w_i}}}{\sum_{k=1}^{K} e^{\boldsymbol{x}^\top * \boldsymbol{w_k}}} \tag{2.11}$$

za višeklasnu klasifikaciju (i je oznaka klase kojoj pripada primjer $\boldsymbol{x}$).

Izlaz modela je tada jednak:

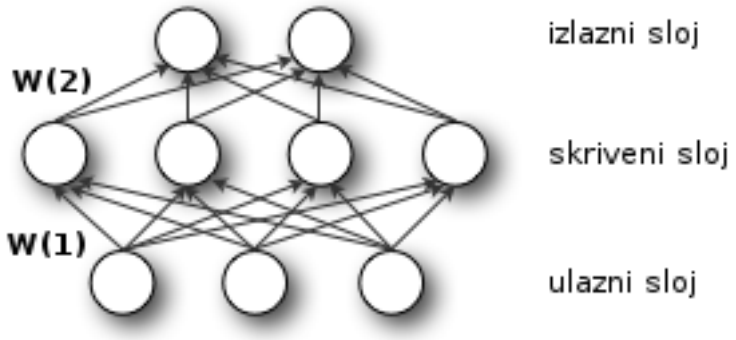
$$h = \arg\max_i P(Y = i|\boldsymbol{x}, \boldsymbol{w}, b) \tag{2.12}$$

2.4. Višeslojni perceptron

Do sada spomenuti modeli i arhitekture modeliraju samo jedan biološki neuron, dok je neuronska mreža skup neurona. Spajanjem više modela neurona u više slojeva (u dubinu) i više neurona u širinu, nastaje neuronska mreža. Najjednostavniji model neuronske mreže naziva se višeslojni perceptron.

Višeslojni perceptron sastoji se od 3 sloja: ulaznog sloja, skrivenog sloja (engl. *hidden layer*) i izlaznog sloja. Ulazni sloj se sastoji od brojeva koji opisuju neki primjer (vektor značajki nekog primjera), skriveni sloj se sastoji od logističkih neurona i zadnji sloj koji se također sastoji od logističkih neurona (broj izlaznih neurona ovisi o broju razreda u koji se klasificira primjer).

Višeslojni perceptron (slika 2.2) može se promatrati i kao vrsta logističke regresije kod koje su ulazi transformirani pomoću nelinearne transformacije. Svrha tih transformacija je prebaciti ulazne podatke u prostor više dimenzije gdje oni postaju linearno separabilni (budući da je logistička regresija linearni klasifikator). Transformacije izvodi skriveni sloj i on je dovoljan da MLP postane univerzalni aproksimator.



Slika 2.2: Višeslojni perceptron

Izlaz višeslojnog perceptrona je tada:

$$f(\boldsymbol{x}) = G\{\boldsymbol{W}(2) \cdot (s[\boldsymbol{W}(1) \cdot \boldsymbol{x} + \boldsymbol{b}(1)]) + \boldsymbol{b}(2)\} \tag{2.13}$$

G i s su aktivacijske funkcije. Za aktivacijsku funkciju se obično odabire logistička sigmoid funkcija ili $\tanh$. $\tanh$ se odabire jer se višeslojna mreža s njime kao prijenosnom funkcijom brže trenira. Jednadžba se može protumačiti na način da se izlaz skrivenog sloja $h(\boldsymbol{x}) = s(\boldsymbol{W}(1) * \boldsymbol{x} + \boldsymbol{b})$ prosljeđuje na ulaz logističke regresije, čime dobijemo višeslojni perceptron.

2.5. Duboke neuronske mreže

Duboke neuronske mreže (ili kraće, DNN) su naziv za neuronske mreže koje sadrže dva ili više skrivena sloja. Takve duboke mreže omogućuju modeliranje komplesnih nelinearnih zavisnosti, poput viešeslojnog perceptrona. Zbog velike količine parametara i same dubine, dugo vremena je treniranje takve mreže bilo problematično, no u zadnjih nekoliko godina, razvile su se metode za njihovo treniranje, ali i same arhitekture koje imaju puno manji broj parametera, a jednaku dubinu i širinu.

Recimo da se želi konstruirati sustav za prepoznavanje lica na slici. Mogli bi koristiti jedan model višeslojnog perceptrona koji bi odgovorio na pitanje postoji li oko slici, drugi model koji bi pak odgovorio na pitanje postoje li usta na slici, treći model koji bi odgovorio postoji li kosa na slici. Izlaze tih modela bi mogli spojiti na logističku regresiju koja koristi sve izlaze postojećih modela i vraća odgovor postoji li lice na slici. Duboke neuronske mreže mogu zamijeniti takav složeni sustav, jer omogućuju spajanje više slojeva koji se specijaliziraju za različite dubine apstrakcije, a treniranje i korištenje dubokih mreža je jednostavnije od navednog skupa modela.

DNN su obično dizajnirane kao mreže s unaprijednom propagacijom, no posljednja istraživanja pokazuju da je moguće koristiti i povratne neuronske mreže (engl. *recurrent networks*) u dubokim arhitekturama (primjer je modeliranje jezika). Jedna specifična verzija dubokih neuronskih mreža su konvolucijske neuronske mreže koje su vrlo često korištene u računalnom vidu (detaljno opisane u poglavlju 2.6).

Pokazalo se da je proces treniranja dubokih neuronskih mreža iznimno težak. Algoritmi bazirani na gradijentnim metodama skloni su pronalasku lokalnih optimuma. Drugi je problem nestajući gradijent (engl. *vanishing gradient problem*) koji se dešava tokom unazadne propagacije gradijenata jer se iznos gradijenata smanjuje za red veličine po sloju što znači da se gradijent teško prenosi do prednjih slojeva.

Nadalje, iole kompleksnije mreže su vrlo teške za interpretaciju, čak i ako rade dobro, količina parametara je toliko ogromna da je mreža zapravo ”crna kutija” i nemoguće je analizom ustvrditi zašto dobro rade.

U zadnjih desetak godina slojevitim umjetnim neuronskim mrežama pristupa se na novi način. Radi se s nekoliko tehnika koje imaju nekoliko zajedničkih aspekata. Kod nekih tehnika (autoenkoderi, ograničeni Boltzmanovi strojevi) slojevi se treniraju jedan po jedan, dok kod drugih (konvolucijske mreže) postoji posebna vrsta slojeva koji imaju smanjen broj parametara i rijetko su povezani čime je smanjen broj operacija po sloju.

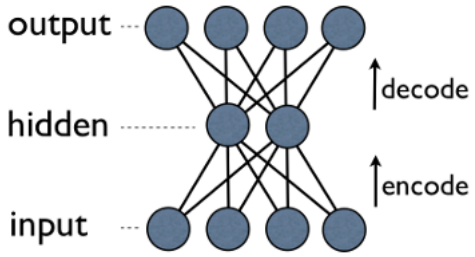
Kao zajedničko ime za više tehnika koje na ovaj način pristupaju treniranju umjetne

neuronske mreže koristi se termin "duboka neuronska mreža". Duboke neuronske mreže na tipičnim klasifikacijskim problemima postižu rezultate koji su među ponajboljima ikad postignutim.

2.5.1. Autoenkoderi

Autoenkoderi su posebna vrsta umjetnih neuronskih mreža koje se koriste za učenje prikaza informacija u manje memorije (sažimanje), na što se može gledati i kao na metodu smanjivanja značajki ulaznog prostora. Osnovna ideja autoenkodera je poprilično jednostavna: imamo neuronsku mrežu kojoj ulazni i izlazni sloj imaju jednak broj neurona, u sredinu dodamo još jedan sloj s manjim brojem neurona. Taj srednji sloj, koji je zapravo ”usko grlo” će naučiti efikasno prikazati ulaze podatke (kodirati ih).

Najjednostavniji model autoenkoderske mreže ima, kao što je već spomenuto, tri sloja (poput MLP-a, poglavlje 2.4): ulazni, skriveni i izlazni. Skriveni sloj ima manje neurona od ulaznog, jer se prebacivanjem iz ulaznog u skriveni vrši sažimanje. Postupkom treniranja se nastoji postići da informacija koja dođe na ulazni sloja ($\boldsymbol{x}$) čim sličnija izađe iz izlaznog sloja ($\hat{\boldsymbol{x}}$). Iz toga slijedi da izlazni i ulazni sloj moraju imati jednak broj neurona. Pogreška rekonstrukcije ($\Delta\boldsymbol{x} = \boldsymbol{x} - \hat{\boldsymbol{x}}$) se koristi za podešavanje težina u mreži.



Slika 2.3: Jednostavni autoenkoder s jednim skrivenim slojem

Hinton je u radu [8] prvi upotrijebio autoenkodere kao metodu treniranje dubokih neuronskih mreža. Takvim se treniranjem svaki sloj redom uzima kao skriveni sloj autoenkodera i trenira. Na kraju se cijela mreža nastala od tako istreniranih slojeva spaja i rezultirajuća se mreža fino ugađa primjenom postojećih gradijentnih tehnika. Treniranjem slojeva pojedinačno poboljšava se pretraga parametarskog prostora mreže: bolje se izbjegavaju lokalni optimumi. Moguće je efikasno trenirati više slojeva koji sadrže veći broj neurona, što omogućava stvaranje puno kompleksnijih mreža koje se dobro nose s težim zadatcima. Niži slojevi mogu se trenirati sa neoznačenim podatcima

(polu-nadzirano učenje).

2.6. Konvolucijske neuronske mreže

Duboke neuronske mreže koje sadrže konvolucijske slojeve (pojašnjeni u sljedećem poglavlju) nazivaju se konvolucijskim neuronskim mrežama. Konvolucijske mreže se danas primjenjuju u većini problema računalog vida gdje vrlo često postižu vrhunske rezultate.

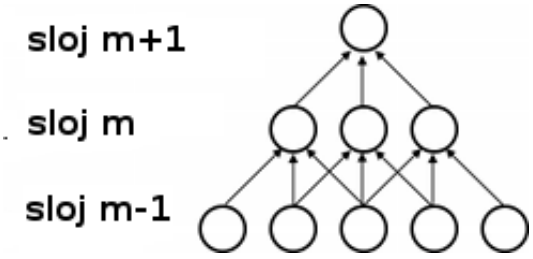
2.6.1. Konvolucijski sloj

Znanstveni radovi temeljeni na vidnom korteksu (engl. *visual cortex*) mačaka pokazali su da postoji složeni raspored vidnih stanica u oku. Neke od tih stanica osjetljive su na samo malu podregiju vidnog polja nazvanu osjetilno polje (engl. *receptive field*). Takve su stanice raspoređene da pokrivaju cijelo vidno polje. Dodatno, identificirana su dva tipa stanica: jednostavne stanice i kompleksne stanice. Jednostavne maksimalno reagiraju na podražajne oblike s izraženim bridovima dok složene imaju veća osjetilna područja i invarijantne su na točnu poziciju podražaja.

Budući da je vizualni korteks trenutno najmoćniji sustav vida, čini se logičnim pokušati emulirati njegovo ponašanje. Mnogi takvi sustavi se mogu pronaći u literaturi, poput sustava NeoCognitron [5] i mnogih drugih [18] [10].

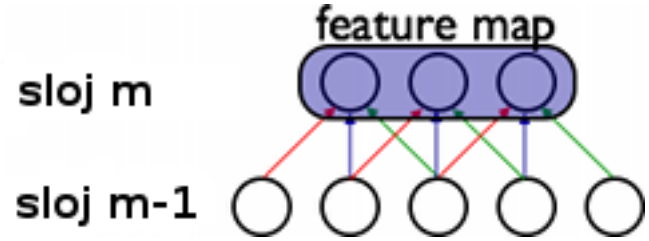
Konvolucija je matematički pojam, definiran kao učestalo primjenjivanje funkcije na podacima. U ovom konktekstu to znači primjenjivati ”filter” na sliku po svim mogućima pomacima, oponašajući osjetilne stanice u ljudskom oku. Filter predstavlja skup težina povezanih sa prethodnim slojem, gdje je prethodni sloj komadić 2-D ulazne slike, a izlaz je jedan neuron. Takav se filter primjenjuje pomičući kroz cijelu sliku, pri čemu se ”osjetilna polja”’ preklapaju, kao što je prikazano na slici 2.4 te nastaje nova 2-D matrica koja se naziva mapa značajki. Jedan konvolucijski sloj sadrži više takvih filtera što ima za posljedicu da nastaje i više mapa značajki.

U zadacima računalnog vida posebno se osjeti rast parametara s dubinom i širinom mreže jer se na ulazu nalaze slike (u digitalnom obliku), dimenzija često preko 100x100 piksela (relativno mala slika), što daje 10 000 značajki na ulazu i čini ulazni prostor visoko dimenzionalnim. Uzmimo za primjer mrežu sa 10 000 ulaznih neurona i 1000 skrivenih neurona. Takav prvi sloj ima već 10 milijuna težina, a slaganjem mreže u dubinu se taj broj još povećava, mreža se teško i sporo trenira te je sklona prenaučenosti.



Slika 2.4: Rijetka povezanost

Konvolucijski slojevi iskorištavaju lokalnu korelaciju (jedan neuron utječe samo na bliske susjede), što znači da je neuron $m-1$ sloja povezan samo sa prostorno bliskim neuronima m-tog sloja. Na slici 2.4 je primjer lokalnog povezivanja širine 3 neurona za jednodimenzijski ulaz. Neka je $m-1$ sloj ulazni sloj. Tada je vidno polje neurona u sloju m široko 3 s obzirom na ulaz. Neuroni u sloju $m+1$ su isto povezani s 3 neurona iz prošlog sloja, ali ako se gledaju s obzirom na ulazni sloj, onda je njihovo vidno polje široko 5 neurona. Arhitektura mreže tako ograničava takve "filtere" na lokalni uzorak (budući da ne reagiraju na druge dijelove ulaza). Slaganje više takvih slojeva omogućava da "filteri" polako postaju globalni (odnosno prostiru se preko većeg broja ulaznih piksela). Na primjer, neuron u sloju $m+1$ može predstavljati (engl. *encode*) nelinearnu značajku duljine 5.



Slika 2.5: Zajedničke težine

Budući da se govori o primjeni u računalnom vidu, ulazi su uglavnom 2D slike. Ulaz u konvolucijski sloj ima dimezije $x \cdot y \cdot r$, gdje je x širina slike, y visina slike, a r je broj kanala (konkretno za RGB sliku je $r = 3$). Dimenzionalnost konvolucijskog sloja ne ovisi direktno o ulaznom sloju, već samo o broju kanala i iznosi $d \cdot d \cdot r \cdot k$, gdje je d širina odnosno visina filtera za konvoluciju, r je broj kanala prethodnog sloja, k je broj mapa značajki trenutnog sloja (ekvivalent broju kanala na ulazu). Primjer takvog sloja ima filter dimenzija $7 \cdot 7$ i 20 mapa značajki što daje $7 \cdot 7 \cdot 3 \cdot 20 = 2940$ težina (značajno manje od prije spomenutih 10 milijuna). Naravno, broj mapa značajki se zna penjati i do nekoliko stotina, no broj težina je i u tom slučaju smanjen naspram potpuno povezanog skrivenog sloja.

Gledano s drugog stajališta, neuroni koji čine konvolucijski sloj dijele zajednički vektor težina $\boldsymbol{w}$ i pomak b. Time se omogućava da se neka značajka otkrije bez obzira na njezinu lokaciju u ulaznom prostoru. Ova je tehnika vrlo efikasna jer uvelike smanjuje broj parametara sloja, pa time i broj parametara koji se treniraju. I *gradijentni spust* se još uvijek može koristiti za učenje parametara. Slika 2.5 prikazuje takav sloj. Težine iste boje su zajedničke neuronima.

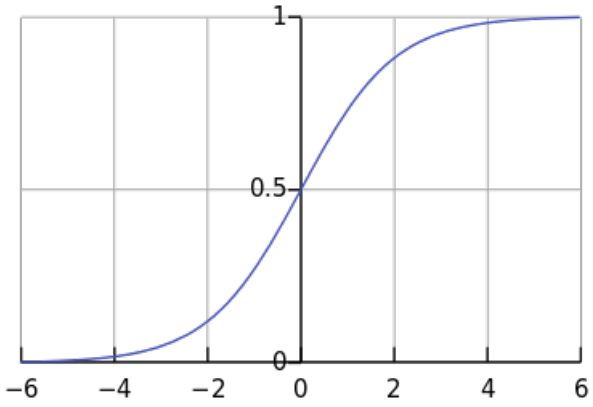
Mapa značajki, odnosno izlaz konvolucijskog sloja, se dobije ponavljanim apliciranjem funkcije po podregiji slike, ili drugim rječima, konvolucijom ulazne slike s linearnim filterom, dodajući pristranost sloja b i primjenjujući aktivacijsku funkciju. Ako označimo k-tu mapu značajki nekog sloja kao h^k, čiji su filteri određeni težinama W^k i pristranošću b_k, onda izlaz takve mape značajki možemo zapisati kao:

$$h_{ij}^k = f\left(\left(W^k * x\right)_{ij} + b_k\right) \tag{2.14}$$

gdje su ij indeksi ulazne (x) odnosno izlazne slike (mape značajki). U praksi se još svaki konvolucijski sloj sastoji od više mapi značajki, $\{h^{(k)}, k = 0..K\}$

2.6.2. Aktivacijske funkcije

Aktivacijska funkcija je funkcija koju koristimo nad izlazima sloja neuronske mreže. Kod logističke regresije aktivacijska funkcija je logistička sigmoid funkcija (formula 2.15, slika 2.6) zbog koje je izlaz logističke regresije probabilistički. Logistička funkcija je nelinearna, monotono rastuća funkcija čiji se izlaz asimptotski približava nekoj konačnoj vrijednosti (na primjer, broju 1) dok ulaz raste prema $\pm\inf$. Zbog tih svojstava MLP (poglavlje 2.4) može aproksimirati proizvoljnu nelinearnu funkciju.



Slika 2.6: Logistička sigmoid funkcija

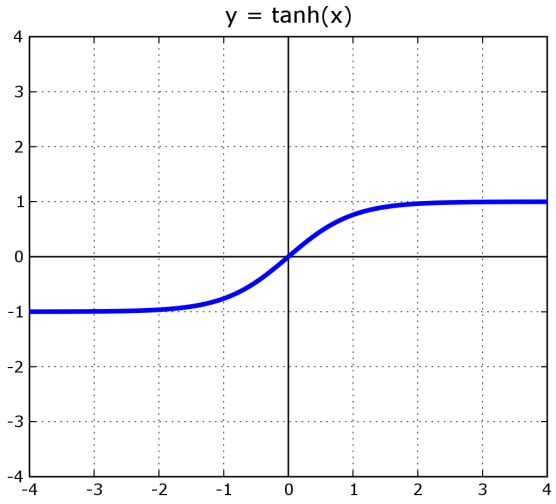
Osim standardne sigmoid funkcije definirane formulom:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.15}$$

često se koristi i hiperbolna tangent funkcija $f(x) = \tanh(x)$ (slika 2.7), isto iz porodice logističkih funkcija.

$$f(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.16}$$

Hiperbolna tangent funkcija se više preferira kao aktivacijska funkcija iz istog razloga zbog kojeg se ulazi normaliziraju: pokazalo se (LeCun u radu [11]) da mreže brže konvergiraju ako su ima ulazi normalizirani (translatirani tako da im je srednja vrijednost 0 i standardna deviacija 1). Izlazi jednog sloja su ulaz u drugi iz čega slijedi da bi i izlazi trebali biti u prosjeku oko nule, što se postiže $\tanh$ funkcijom, dok su izlazi sigmoid funkcije pozitivni i centrirani oko 0.5 (ili neke druge pozitivne vrijednosti, ovisno o konkretnom obliku funkcije).



Slika 2.7: Hiperbolna tangent funkcija

Autor u radu [11] preporuča korištenje logističke funkcije $f(x) = 1.7159tanh(\frac{2}{3}x)$. Konstante u navedenoj funkciji su odabrane tako da, ako je ulaz normaliziran, izlazi će isto imati varijancu blizu 1. Konkretni sigmoid ima i sljedeća svojstva: (a) $f(\pm 1) = \pm 1$, (b) druga derivacija ima maksimum u točki $x = 1$.

Logističke funkcije su često definirane izrazima koji sadrže potenciranje prirodnog logaritma $e^x, x \in R$, što je računalno prilično zahtjevno. Taj se problem rješava aproksimacijom konkretnog izraza polinomijalnog funkcijom.

Drugi je problem nestajući gradijent (engl. *vanishing gradient problem*). Nestajući gradijent je pojava koja se događa kada neuroni nekog sloja duboke neuronske mreže imaju gradijente blizu nule jer su izlazi sloja bili blizu zasićenja, odnosno blizu asimptota (koje su često ± 1). Gradijenti blizu nule uzrokuju sporu konvergenciju jer malen iznos gradijenta znači i malu promjenu težina, a pokazalo se i da takve mreže često zaglave u lokalnim minimumima.

Rectified linear unit (kraće ReLU) je vrsta aktivacijske funkcije definirana formulom:

$$f(x) = max(0, x) = \begin{cases} 0 & \text{ako } x < 0 \\ x & \text{ako } x \geq 0 \end{cases} \tag{2.17}$$

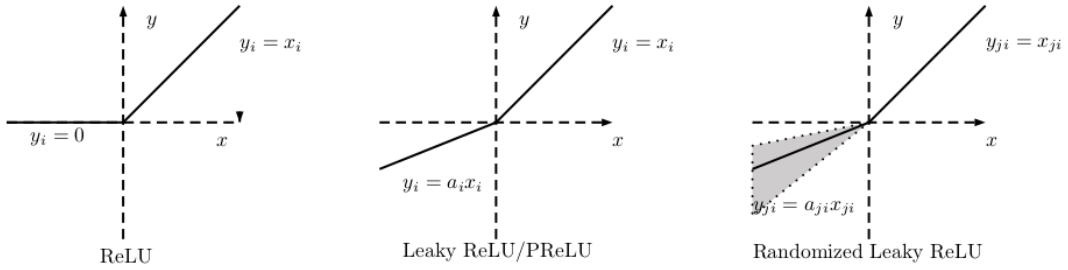
ReLU rješava oba gore navedena problema: brzo se računa, kao i gradijent (koji je 1, ako je $x > 0$ ili 0) i gradijent nikad ne uđe u zasićenje (uvijek je 1 ili 0), neovisno o dubini mreže.

No, ReLU-ovi imaju i potencijalne mane: ako neki neuron nije aktivan, gradijent je 0, što znači da se težine tog neurona više ne mogu pomaknuti gradijentnom optimizacijom. Tome se problemu može doskočiti postavljanjem početnog pomaka (engl. *bias*) na neku malu pozitivnu vrijednost (na primjer 0.001), čime će na početku svi neuroni biti aktivirani. Gradijentna optimizacija ih tada može pomicati prema ishodištu (praktički isključiti) ili u pozitivnom smjeru. [14]

Vjerovalo se i da dobri rezultati, koje ReLU aktivacije postižu u dubokim neuronskim mrežama, potiču i od mogućnosti da ReLU aktivacijska funkcija "isključi" neuron (ako vrijednost na ulaza u aktivaciju postane < 0). Rad [23] je pokazao da uspjeh ReLU-a ne leži u gašenju nekih neurona (implicitnom uvođenju rijetke povezanosti). U radu je dana usporedba standardne ReLU funkcije, *leaky ReLU* (lReLU) funkcije, *parametric ReLU* (PRelU) funkcije i *randomized ReLU* (RReLU) funkcije. Eksperimenti su pokazali da funkcije bez nagiba sa smjerom 0 (praktički sve osim standardnog ReLU-a), konstantno daju bolje rezultate što znači da teorija o gašenju neurone ne vrijedi, jer takve funkcije nikad potpuno ne gase neurone. Pokazano je i da (na manjem skupu podataka) korištenje deterministčkog nagiba, ili učenje nagiba na temelju podatka može uzrokovati prenaučenost, naspram funkcije sa slučajnim odabirom nagiba (nagib iz nekog raspona vrijednosti).

2.6.3. Sažimanje

Sažimanje maksimalnog odziva je vrsta nelinearnog poduzorkovanja gdje se ulazna slika dijeli na više nepreklapajućih pravokutnika i za svaki pravokutnik vrati se najveća



Slika 2.8: Usporedba ReLU aktivacijskih funkcija

vrijednost.

On je važan u računalnom vidu iz dva razloga:

1. smanjuje računsku složenost za gornje slojeve

2. povećava neosjetljivost na translacije u slici

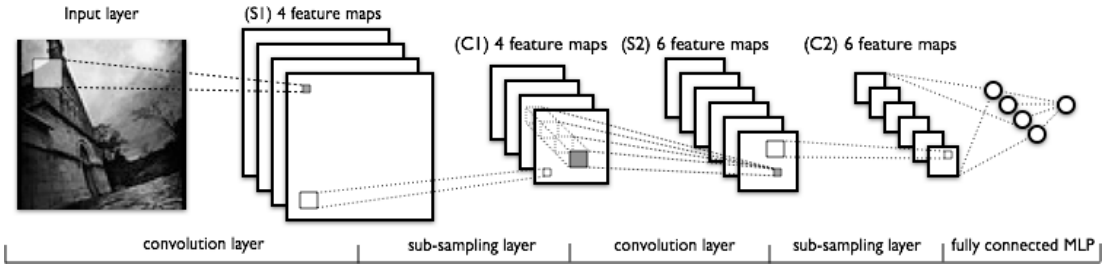
Pojasnimo malo povećanje neosjetljivosti na translacije: recimo da se radi sažimanje maksimuma nad regijom od 2x2 piksela. Neka je gornji lijevi piksel onaj s maksimalnom vrijednošću. Ako se sve na slici pomakne za jedan piksel udesno, piksel s maksimalnom vrijednošću će biti gornji desni. To znači da će u oba slučaja izlaz sloja sažimanja maksimuma biti jednak, tj biti će neosjetljiv na takve translacije.

Uobičajena je praksa da se u gornjim slojevima povećava broj mapa značajki. Time bi računalna složenost previše ekspandirala u višim slojevima. Budući da viši slojevi sadrže informacije visoke razine (engl. *high level information*), nije važno da su takve informacije dostupne za svaki piksel, dovoljno je da one pokrivaju veće regije slike. Iz tog razloga se poduzorkovanjem u višim slojevima ne gube važne informacije, a uvelike se smanjuje broj operacija i vrijeme potrebno za računanje mapa značajki.

2.6.4. Primjer arhitekture

LeCun je još 1989. godine u radu [11], za klasifikaciju rukom pisanih brojeva na poštanskim pošiljkama, predstavio konvolucijsku mrežu čija arhitektura je začetak današnjih arhitektura. Takva mreža imala je 3 konvolucijska sloja, bez slojeva sažimanja maksimumom, s potpuno povezanim slojem na kraju. Značajan je i LeCun-ov rad [10] u kojem se predstavlja arhitektura nazvana LeNet, čije inačice se koriste u mnogim verzijama konvolucijskih mrežama.

U početnim se slojevima mreže izmjenjuju slojevi sažimanja maksimalnog odziva i mape značajki. Konkretno, prvi sloj se sastoji od 4 mape značajki, zatim slijedi sloj



Slika 2.9: LeNet mreža

sažimanja maksimuma, pa sloj od 6 mapi značajki i opet sloj sažimanja maksimuma. Zadnji dio takve mreže je višeslojni perceptron na čije su ulaze spojeni izlazi zadnjeg sloja sažimanja maksimuma. Taj se višeslojni perceptron sastoji od 2 sloja. Prvi je skriveni sloj, a iza njega je sloj logističke regresije. Logistička regresija na kraju čini konačnu klasifikaciju. U konkretnom primjeru postoji 10 izlaza, jedan za svaku znamenku.

3. Učenje mreže

"Učenjem" modela naziva se postupak pronalaska optimalnih parametara modela. Optimalni parametri su oni koji danim modelom modeliraju skup podataka za učenje, $\mathcal{D}$. No, modeli se često znaju previše dobro prilagoditi skupu podataka za učenje pa se zapravo relevantni rezultati dobivaju kad se model testira na neviđenim podacima.

Rješenje za optimalne parametre neuronskih mreža nije dostupno u zatvorenoj formi pa se poseže za gradijentnim metodama koje onda kroz iteracije pronalaze optimalne parametre. To znači da nam je potrebna ciljna funkcija, funkcija čiji rezultat nam govori koliko su parametri modela dobri.

Budući da se i semantička segmentacija na neki način može gledati kao klasifikacija, jer se na kraju svaki piksel ili grupa piksela neke slike klasificiraju (za razliku od klasifikacije gdje svaki ulaz ima jednu oznaku, u semantičkoj segmentaciji svaka slika ima skup oznaka koje je potrebno predvidjeti), prvo što pada na pamet je jedan-nula funkcija gubitka. Ako model preslikava ulazni primjer u jedan od razreda iz skupa $0, ..., L$, definirajmo takvo preslikvanje funkcijom f $f : R^D \rightarrow 0, ..., L$, onda se funkcija gubitka može definirati kao:

$$\ell_{0,1} = \sum_{i=0}^{|\mathcal{D}|} I_{f(x^{(i)}) \neq y^{(i)}} \tag{3.1}$$

gdje je $\mathcal{D}$ skup podataka, a I indikatorska funkcija definirana kao:

$$I(x) = \begin{cases} 1 & \text{ako je x istinit} \\ 0 & \text{inače} \end{cases} \tag{3.2}$$

No, budući da jedan-nula gubitak nije derivabilan pa je nemoguće provoditi metode gradijentnog spusta, on nije praktičan za korištenje. Stoga se uvodi funkcija negativne log izglednosti (engl. *negative log likelihood*), spomenuta u poglavlju 2.3.

$$NLL(\theta, \mathcal{D}) = -\sum_{i=0}^{|\mathcal{D}|} \log P(Y = y^{(i)} | x^{(i)}, \theta) \tag{3.3}$$

3.1. Algoritam unazadne propagacije, *engl. backpropagation*

Algoritmom unazadne propagacije izvode se izrazi za gradijente (iznose derivacija) parametara mreže. Ti se izrazi kasnije koriste u algoritmu gradijentnog spusta za ugađanje parametara.

Algoritam unazadne propagacije prvi je put predstavljen 1970-ih, ali popularnost i prihvaćanje je stekao tek 1986. kad je Hinton izdao poznati rad koji opisuje nekoliko tada poznatih tipova neuronskih mreža i njihovo učenje unazadnom propagacijom koje je bilo brže od prije korištenih metoda učenja i omogućavalo rješavanje problema, koji su do tada bili nerješivi, uz pomoć neuronskih mreža. Do danas je ta metoda učenja postala najproučavanija i nakorištenija.

Slijedi izvod algoritma unazadne propagacije za neuronsku mrežu s d ulaza, m izlaza i l skrivenih slojeva. $l+1$ sloj je izlazni sloj. Ulaz u sloj k je $\boldsymbol{x}^{(k)}$, a izlaz iz sloja je $\boldsymbol{x}^{(k+1)}$.

Mreža se uči nad skupom za učenje od N parova $(\boldsymbol{x}_i, \boldsymbol{t}_i)$. Vektor $\boldsymbol{x}_i$ je d dimenzionalan, vektor $\boldsymbol{t}_i$ je m dimenzionalan, a $i \in \{0..N-1\}$. Mreža koristi sigmoidalnu prijenosnu funkciju:

$$f(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{x}}} \tag{3.4}$$

i funkciju gubitka (srednja kvadratna pogreška):

$$E = \frac{1}{2N}\sum_{n=1}^{N}\sum_{o=1}^{m}(t_{n,o}-y_{n,o})^2 = \frac{1}{2N}\sum_{n=1}^{N}\left(\boldsymbol{t}_n - \boldsymbol{y}_n^{(l+1)}\right)^2 \tag{3.5}$$

gdje je

$$\boldsymbol{y}_n = \boldsymbol{x}_n^{(l+1)} = (W^{l+1}\boldsymbol{x}_n^{(l)} + \boldsymbol{b}^{(l)}) = f(\boldsymbol{net}^{(l+1)}) \tag{3.6}$$

$$\boldsymbol{net}_n^{(k)} = W^k\boldsymbol{x}_n^{(k-1)} + \boldsymbol{b}^{(k)} \tag{3.7}$$

Traže se vrijednosti težina mreže da funkcija E bude minimalna, odnosno želimo izračunati izraz za $\frac{\partial E}{\partial W^{(k)}}$ koji ćemo koristiti kod ugađanja težina:

$$W^{(k)}(t) = W^{(k)}(t-1) - \epsilon\frac{\partial E}{\partial W^{(k)}}(t) \tag{3.8}$$

gdje je $k \in \{1..l+1\}$, $W^{(k)}$ matrica težina sloja k, t je oznaka trenutnog koraka, a $t-1$ oznaka prethodnog koraka.

Parcijalna derivacija funkcije gubitka po težinama za izlazni sloj je:

$$\begin{aligned}\frac{\partial E}{\partial W^{(l+1)}} &= \frac{\partial}{\partial W^{(l+1)}}\left[\frac{1}{2N}\sum_{n=1}^{N}\left(\boldsymbol{t}_n-\boldsymbol{y}_n\right)^2\right]\\ &= \frac{-1}{N}\sum_{n=1}^{N}\left(\boldsymbol{t}_n-\boldsymbol{y}_n\right)\frac{\partial \boldsymbol{y}_n}{\partial W^{(l+1)}}\\ &= \frac{-1}{N}\sum_{n=1}^{N}\left(\boldsymbol{t}_n-\boldsymbol{y}_n\right)\frac{\partial \boldsymbol{y}_n}{\partial \boldsymbol{net}_n^{(l+1)}}\frac{\partial \boldsymbol{net}_n^{(l+1)}}{\partial W^{(l+1)}}\end{aligned} \tag{3.9}$$

$$\frac{\partial \boldsymbol{net}_n^{(l+1)}}{\partial W^{(l+1)}} = \frac{\partial}{\partial W^{(l+1)}}\left[W^{(l+1)}\boldsymbol{x}_n^{(l)}+\boldsymbol{b}^{(l+1)}\right] = \boldsymbol{x}_n^{(l)} \tag{3.10}$$

Budući da je $\boldsymbol{y}_n$ zapravo $\boldsymbol{y}_n = f(\boldsymbol{net}_n^{(l+1)})$, gdje je f logistička funkcija čija je derivacija $f'(x) = f(x)[1-f(x)]$, slijedi da je:

$$\frac{\partial \boldsymbol{y}_n}{\partial \boldsymbol{net}_n^{(l+1)}} = f\left(\boldsymbol{net}_n^{(l+1)}\right)\left[1-f\left(\boldsymbol{net}_n^{(l+1)}\right)\right] \tag{3.11}$$

iz čega onda dobijemo:

$$\begin{aligned}\frac{\partial E}{\partial W^{(l+1)}} &= \frac{-1}{N}\sum_{n=1}^{N}\left(\boldsymbol{t}_n-\boldsymbol{y}_n\right)f\left(\boldsymbol{net}_n^{(l+1)}\right)\left[1-f\left(\boldsymbol{net}_n^{(l+1)}\right)\right]\boldsymbol{x}_n^{(l)}\\ &= \frac{-1}{N}\sum_{n=1}^{N}\delta_n^{(l+1)}\boldsymbol{x}_n^{(l)}\end{aligned} \tag{3.12}$$

gdje se $\delta_n^{(l+1)}$ definira kao:

$$\delta_n^{(l+1)} = \left(\boldsymbol{t}_n-\boldsymbol{y}_n\right)f\left(\boldsymbol{net}_n^{(l+1)}\right)\left[1-\boldsymbol{net}_n^{(l+1)}\right] = \left(\boldsymbol{t}_n-\boldsymbol{y}_n\right)f'\left(\boldsymbol{net}_n^{(l+1)}\right) \tag{3.13}$$

a $\boldsymbol{x}_n^l$ je ulaz u izlazni sloj.

Uzmimo sada za primjer matricu težina nekog skrivenog sloja l. Izvod započinje slično:

$$\begin{aligned}\frac{\partial E}{\partial W^{(l)}} &= \frac{\partial}{\partial W^{(l)}}\left[\frac{1}{2N}\sum_{n=1}^{N}\left(\boldsymbol{t}_n-\boldsymbol{y}_n\right)^2\right]\\ &= \frac{-1}{N}\sum_{n=1}^{N}\left(\boldsymbol{t}_n-\boldsymbol{y}_n\right)\frac{\partial \boldsymbol{y}_n}{\partial W^{(l)}}\\ &= \frac{-1}{N}\sum_{n=1}^{N}\left(\boldsymbol{t}_n-\boldsymbol{y}_n\right)\frac{\partial \boldsymbol{y}_n}{\partial \boldsymbol{net}_n^{(l+1)}}\frac{\partial \boldsymbol{net}_n^{(l+1)}}{\partial W^{(l)}}\end{aligned} \tag{3.14}$$

Izraz $\frac{\partial \boldsymbol{net}_n^{(l+1)}}{\partial W^{(l+1)}}$ smo već derivirali u 3.10, a izraz $\frac{\partial \boldsymbol{net}_n^{(l+1)}}{\partial W^{(l)}}$ deriviramo kao:

$$\begin{aligned}\frac{\partial \boldsymbol{net}_n^{(l+1)}}{\partial W^{(l)}} &= \frac{\partial}{\partial W^{(l)}}\left[W^{(l+1)}\boldsymbol{x}_n^{(l)} + \boldsymbol{b}^{(l+1)}\right] \\ &= W^{(l+1)}\frac{\partial \boldsymbol{x}_n^{(l)}}{\partial W^{(l)}} \\ &= W^{(l+1)}\frac{\partial f\left(\boldsymbol{net}_n^{(l)}\right)}{\partial W^{(l)}} \\ &= W^{(l+1)}\frac{\partial f\left(\boldsymbol{net}_n^{(l)}\right)}{\partial \boldsymbol{net}_n^{(l)}}\frac{\partial \boldsymbol{net}_n^{(l)}}{\partial W^{(l)}} \\ &= W^{(l+1)} f\left(\boldsymbol{net}_n^{(l)}\right)\left[1 - f\left(\boldsymbol{net}_n^{(l)}\right)\right]\boldsymbol{x}_n^{(l-1)}\end{aligned} \tag{3.15}$$

Kad to uvrstimo natrag u izraz 3.14 dobijemo:

$$\frac{\partial E}{\partial W^{(l)}} = \frac{-1}{N}\sum_{n=1}^{N}(\boldsymbol{t}_n - \boldsymbol{y}_n) f\left(\boldsymbol{net}_n^{(l+1)}\right)\left[1 - f\left(\boldsymbol{net}_n^{(l+1)}\right)\right] W^{(l+1)} f\left(\boldsymbol{net}_n^{(l)}\right)\left[1 - f\left(\boldsymbol{net}_n^{(l)}\right)\right]\boldsymbol{x}_n^{(l-1)} \tag{3.16}$$

što se može kraće zapisati kao:

$$\frac{\partial E}{\partial W^{(l)}} = \frac{-1}{N}\sum_{n=1}^{N}\delta_n^{(l)}\boldsymbol{x}_n^{(l-1)} \tag{3.17}$$

gdje je $\delta^{(l)}$ jednak:

$$\begin{aligned}\delta^{(l)} &= \delta^{(l+1)} W^{(l+1)} f\left(\boldsymbol{net}_n^{(l)}\right)\left[1 - f\left(\boldsymbol{net}_n^{(l)}\right)\right] \\ &= \delta^{(l+1)} W^{(l+1)} f'\left(\boldsymbol{net}_n^{(l)}\right)\end{aligned} \tag{3.18}$$

a l je proizvoljni skriveni sloj.

Vidi se da su izrazi za $\frac{\partial E}{\partial W^{(l)}}$ jednaki:

$$\frac{\partial E}{\partial W^{(k)}} = \frac{-1}{N}\sum_{n=1}^{N}\delta_n^{(k)}\boldsymbol{x}_n^{(k-1)} \tag{3.19}$$

gdje je k ili skriveni ili izlazni sloj. Drugačiji je samo način na koji se računa $\delta^{(k)}$. Za zadnji sloj se koristi izraz 3.13, dok se za skriveni sloj koristi izraz 3.18 koji koristi već izračunati δ_n idućeg sloja. Upravo se zato algoritam naziva "algoritam unazadne propagacije" jer se kreće od zadnjeg (izlaznog sloja) i pogreška, odnosno podešavanje težina, se propagira prema prvom (ulaznom) sloju.

3.2. Gradijentni spust

Gradijentni spust je optimizacijski algoritam, odnosno algoritam pronalaska minimuma derivabilne konveksne funkcije. Algoritam gradijentnog spusta traži lokalni minimum

na način da u trenutnoj točki traži smjer gradijenta funkcije i zatim napravi pomak u negativnom smjeru gradijenta. Ako je korak dovoljno malen, nova vrijednost trenutne točke će biti manja od prethodna. U slučaju da je pomak prevelik, moguće je da se pomaknemo na suprotnu padinu funkcije i zapravo povećamo vrijednost funkcije u trenutnoj točki. Korak algoritma označava se konstantom, η.

Neka je $f(x)$ funkcija čiji minimum tražimo, a trenutna točka, t je trenutni korak, a $t-1$ je prethodni korak. Jedan korak algoritma je onda:

$$a(t) = a(t-1) - \eta f'\left(a(t-1)\right) \tag{3.20}$$

ili za funkciju F s više parametara:

$$a(t) = a(t-1) - \eta \frac{\partial F}{\partial x}\left(a(t-1)\right) \tag{3.21}$$

Pod uvjetom da je ϵ dovoljno malen biti će $f(a(t)) < f(a(t-1))$, odnosno pomaknut ćemo se u točku koja je bliže minimumum funkcije.

Kod neuronskih mreža funkcija koju optimiramo je funkcija gubitka E, a točka koju pomičemo je višedimenzionalna točka čija svaka dimenzija predstavlja jednu težinu neuronske mreže.

Da bi se algoritam gradijentnog spusta mogao koristiti za podešavanje težina u neuronskim mrežama, potrebno je derivirati funkciju gubitka. Kao prvo, nužno je da je ona derivabilna što je slučaj kod logističke regresije jer se koristi logistička sigmoid funkcija $f(x) = \frac{1}{1+e^{-x}}$. Njena derivacija je:

$$\begin{aligned} f'(x) &= \left(\frac{1}{1+e^{-x}}\right)' = \frac{0\cdot(1+e^{-x}) - 1\cdot(1+e^{-x})'}{(1+e^{-x})^2} \\ &= \frac{-e^{-x}\cdot(-1)}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\cdot\frac{e^{-x}}{1+e^{-x}} \\ &= \frac{1}{1+e^{-x}}\cdot\frac{1-1+e^{-x}}{1+e^{-x}} = \frac{1}{1+e^{-x}}\cdot\frac{(1+e^{-x})-1}{1+e^{-x}} \\ &= \frac{1}{1+e^{-x}}\cdot\left[\frac{1+e^{-x}}{1+e^{-x}} + \frac{-1}{1+e^{-x}}\right] = f(x)\left[1-f(x)\right] \end{aligned} \tag{3.22}$$

3.2.1. Stohastički gradijentni spust

TODO doraditi, uskladiti oznake

Obično se koristi metoda gradijentnog spusta nazvana stohastički gradijentni spust. Neka je $E_n(w)$ vrijednost funkcije gubitka na n-tom primjeru za neke parametre w.

Kada bi za minimizaciju koristili standardni (ili "grupni") gradijentni spust, jedna bi iteracija izgledala ovako:

$$w := w - \eta \nabla E(w) = w - \eta \sum_{n=1}^{N} \nabla E_i(w), \tag{3.23}$$

gdje je η korak učenja (često zvan i stopa učenja).

U mnogo slučajeva, funkcija nad kojom se vrši zbrajanje ima jednostavan oblik koji omogućava brzo izračunavanje zbrajanje i cijelog izraza.

No, postoje funkcije kod kojih izračuvanje sume u gradijentu zahtijeva dugotrajne izračune. Kad je skup za treniranje velik, to postaje problem. Da bi se ubrzalo izvođenje izračuna, u svakoj iteraciji se izvodi gradijenti spust, odnosno izračun funkcije gradijenta. Takvim se postupkom aproksimira pravi gradijent funkcije gubitka gradijentom jednog primjera (jedne iteracije):

$$w := w - \eta \nabla E_n(w). \tag{3.24}$$

Prolazom algoritma kroz skup za treniranje izvodi se gornje podešavanje težina za svaki primjer. Potrebno je nekoliko prolaza kroz skup za treniranje dok algoritam ne konvergira. Tipične implementacije još i promješaju skup za učenje prije svakog prolaza.

Ako je skup podataka velik i jako redundantan, gradijent na prvoj polovici skupa je skoro identičan gradijentu na drugoj polovici skupa. Umjesto računanja gradijenata za cijeli skup podataka, težine mreže se mogu ugoditi nakon računanja gradijenata za svaku polovicu skupa. Ako dijeljnjem skupa podataka odemo u krajnost, dolazimo do stohastičkog gradijentnog spusta (nazivan još i *online* učenje) gdje se težine ugađaju nakon svakog primjera iz skupa za učenje, čime se konvergencija ubrzava jer nije potrebno računati gradijent svih uzoraka da bi došlo do ugađanja težina.

Za većinu problema najboljim se pokazalo učenje korištenjem male serije skupa podataka (engl. *mini-batch learning*). Budući da je ugađanje težina mreže računalno skupo, takva metoda učenje je brža od običnog stohastičkog gradijentnog spusta jer nema ugađanja težina nakon svakog primjera. Do ubrzavanja dolazi i jer se kod računanja gradijenata za seriju primjera koristi puno matričnog množenja koje je u današnjim arhitekturama računalnih sustava jako efikasno, pogotovo ako se koriste grafičke kartice.

3.2.2. *Momentum* metoda

Za višeslojne neuronske mreže višedimenzionalna ravnina funkcije pogreške je kompliciranog oblika, ali se lokalno može aproksimirati kao kvadratna posuda. Njen vertikalni presjek je onda parabola, a horizontalni elipsa.

(slika)

Uzmimo za primjer kretanje loptice na takvoj funkciji pogreške. Loptica će krenuti u smjeru lokalnog gradijenta, ali jednom kad nakupti brzinu ići će više u smjeru centra kvadratne posude nego u smjeru gradijenta. Njen moment će je natjerati da ide u smjeru ovisnom o prethodnom kretanju.

Slična je stvar i s *momentum* metodom koja uzima u obzir gradijente prethodnih koraka i tako prigušuje oscilacije u promjeni smjera, a povećava brzinu u smjeru konzistentnih gradijenata.

Gradijenti imaju utjecaj na povećanje prethodne brzine (konstanta α polako smanjuje brzinu, ima iznos malo manji od 1, često 0.9):

$$\boldsymbol{v}(t) = \alpha \boldsymbol{v}(t-1) - \epsilon \frac{\partial E}{\partial \boldsymbol{w}}(t) \tag{3.25}$$

Promjena težina je ekvivalentna trenutnoj brzini:

$$\begin{aligned} \Delta \boldsymbol{w}(t) &= \boldsymbol{v}(t) \\ &= \alpha \boldsymbol{v}(t-1) - \epsilon \frac{\partial E}{\partial \boldsymbol{w}}(t) \\ &= \alpha \boldsymbol{w}(t-1) - \epsilon \frac{\partial E}{\partial \boldsymbol{w}}(t) \end{aligned} \tag{3.26}$$

3.2.3. Resilient propagation (RProp)

Rprop metoda, poput stohastičkog gradijentnog spusta, koristi lokalne gradijente da bi ugodila parametre težina mreže. Razlika je da se za Rprop koristi samo predznak derivacija, što znači da je veličina koraka neovisna o amplitudi gradijenta. Ova metoda se koristi samo kod ugađanja težina nakon svih viđenih primjera iz skupa za učenje (engl. *full batch learning*).

Rprop metoda mijenja korak promjene težine za svaku težinu posebno:

- povećava korak množenjem s određenim faktor (na primjer 1.2) ako su posljednja dva predznaka gradijenta jednaka
- smanjuje korak množenjem s određenim faktor (na primjer 0.5) ako se posljedva dva predznaka gradijenta razlikuju

Pojasnimo na primjeru zašto Rprop ne radi dobro kod stohastičkog gradijentog spusta (ugađanja težina nakon svakog primjera). Neka jedna težina mreže ima gradijent +0.1 na devet primjera i gradijent od -0.9 na jednom primjeru. Tada želimo da ta težina ostane otprilike ista, a Rprop će povećati težinu devet puta i smanjiti jednom za otprilike isti iznos.

Postoji više varijanti RProp algoritma: RProp+, RProp-, iRPop+, iRPRop-. U nastavku će biti opisana iRProp- varijanta koja se često koristi. Budući da promjena težina ne ovisi o iznosu gradijenta, definiraju se "vrijednosti promjene" za čiji iznos se podešavaju težine:

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t) & \text{ako} \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ +\Delta_{ij}(t) & \text{ako} \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ 0 & \text{inače} \end{cases} \tag{3.27}$$

gdje $\frac{\partial E}{\partial w_{ij}}(t)$ označava sumu gradijenata za sve primjere.

Drugi korak postupka je određivanje novih "vrijednosti promjene" $\Delta_{ij}(t)$ za svaki parametar mreže (svaku težinu i pristranost).

$$\Delta ij(t) = \begin{cases} \eta^+ * \Delta_{ij}(t-1) & \text{ako} \frac{\partial E}{\partial w_{ij}}(t) * \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ \eta^- * \Delta_{ij}(t-1) & \text{ako} \frac{\partial E}{\partial w_{ij}}(t) * \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ \Delta_{ij}(t-1) & \text{inače} \end{cases} \tag{3.28}$$

gdje je $0 < \eta^- < 1 < \eta^+$. Često se u implementacijama koriste vrijednosti $\eta^- = 0.5$ i $\eta^+ = 1.2$.

Ili rječima: svaki put kada parcijalna derivacija neke težine promjeni predznak, što znači da je prošlo ugađanje težina bilo preveliko i preskočio se lokalni minimum, "vrijednost promjene" se smanji s faktorom η^-. Ako se predznak nije promijenio, "vrijednost promjene" se lagano uveća s faktorom η^+, tako da postupak akcelerira po plitkim područjima.

3.2.4. RMS prop

Rmsprop metoda, kao i stohastički gradijentni spust koristi malene lokalne pomake da bi ugodila težine. Razlika je što *Rmsprop* kod računanja gradijenata u svakom koraku računa i eksponencijalni pomični prosjek (engl. *exponential moving average, EMA*) kvadrata gradijenata. Za svako ugađanje, EMA se koristi kako bi se izračunao korijen iz srednjih kvadrata (engl. *root mean square, RMS*) vrijednosti gradijenata koji su dobiveni u prošlim koracima. Trenutna vrijednost gradijenata se zatim podijeli s korijenom srednjih kvadrata.

Neka je gradijent neke težine W definiran s $grad = \frac{\partial E}{\partial W}$. Eksponencijalni pomični prosjek gradijenata onda možemo definirati s:

$$\boldsymbol{s}(t) = \aleph \cdot \boldsymbol{s}(t-1) + (1-\aleph) \cdot grad^2 \tag{3.29}$$

Opisana varijanta Rmspropa [7] koristi i *momentum* metodu ugrađenu u sam postupak Rmspropa.

$$\boldsymbol{v}(t) = \alpha \boldsymbol{v}(t-1) - \eta \frac{grad}{\sqrt{s+\epsilon}} \tag{3.30}$$

$$W(t) = W(t-1) + \boldsymbol{v}(t) \tag{3.31}$$

gdje je s trenutni iznos eksponencijalnog pomičnog prosjeka, $\aleph$ je konstanta eksponencijalnog pomičnog prosjeka (koristi se vrijednost 0.95), $\boldsymbol{v}$ je trenutna brzina, α je *momentum* konstanta, η je stopa učenja (koristi se vrijednost 0.001), W je matrica težina nekog sloja, a ϵ je konstanta koja sprječava dijeljenje s nulom.

3.3. Regularizacija

Regularizacija je često korištena tehnika u strojnom učenju, a odnosi se na postupak pružanja dodatnih informacija funkciji gubitka kako bi se rješio problem šuma u ulaznom skupu podataka i problem prenaučenosti. Prenaučenost je pojava tokom učenja modela kad se model prilagodi skupu za učenje u tolikoj mjeri da točnost na skupu za provjeru (engl. *validation set*) počne padati, iako se točnost na skupu za učenje povećava.

Regularizacijom se sprječava da se model previše prilagodi skupu za učenje tako da pogreška na neviđenim podacima bude manja (skupu za provjeru) iako se pogreška na skupu za učenju još može smanjiti. U idućim poglavljima navedene su najčešće tehnike za regularizaciju modela dubokih neuronskih mreža.

3.3.1. L_1 i L_2 regularizacija

L_1 norma je suma apsolutnih iznosa kartezijevih komponenti vektora. Drugačije gladajući, L_1 norma se definira kao suma projekcija vektora na koordinatne osi. Za neki vektor $\boldsymbol{x}$ se L_1 norma definira kao:

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i| \tag{3.32}$$

L_2 norma je suma kvadrata komponenti vektora. Za neki vektor $\boldsymbol{x}$ se ona definira:

$$\|\boldsymbol{x}\|_2 = \sum_{i=1}^{n} {x_i}^2 \tag{3.33}$$

Tokom učenja modela, želimo modelom što bolje modelirati podatke nad kojima učimo, no pravi je cilj da model radi dobro na neviđenim primjerima (primjeri koje nije vidio tokom učenja). Pojasnimo na primjeru logističke regresije što se događa ako učenje predugo traje: budući da funkcija negativne log izglednosti asimptotski teži prema nuli, optimizacija nikada neće završiti, a dogodit će se da neke težine u mreži stalno rastu jer tim rastom uzrokuju polagano padanje negativne izglednosti na skupu za učenje. Jednaka stvar se događa i u neuronskim mrežama, gdje je broj težina još puno veći.

Tom rastu se može doskočiti tako da se norma vektora uključi u funkciju gubitka. Funkcija R označava normu vektora težina mreže. Regularizirana funkcija gubitka je onda:

$$E(\theta, \mathcal{D}) = NLL(\theta, \mathcal{D}) + \lambda R(\theta) = \ = NLL(\theta, \mathcal{D}) + \lambda \left\|\theta\right\|_p^p \tag{3.34}$$

gdje je

$$\left\|\theta\right\|_p = (\sum_{j=0}^{|\theta|} |\theta_j|^p)^{\frac{1}{p}} \tag{3.35}$$

p je najčešće 1 ili 2, odnosno koristi se L_1 ili L_2 norma. λ je regularizacijski faktor, jedan od hiperparametara mreže, koji se određuje eksperimentalno.

3.3.2. Slučajne transformacije ili Umjetno povećanje skupa za učenje

Kod modela s velikim brojem parametara, kao što su duboke neuronske mreže, veličina skupa za učenje uvelike utječe na rezultate: što je veći skup za učenje, to su bolji rezultati. No, taj je broj uvijek ograničen. Kod semantičke segmentacije broj podataka je još više ograničen jer je sam postupak označavanja slika semantičkim oznakama dugotrajan pa time i skup. Zato se provodi postupak umjetnog ”povećanja” skupa za učenja slučajnim transformacijama.

Kod računalnog vida, koji se većinom bavi slikama, slučajne transformacije su obično:

- **translacije** slike po x ili y osi za neki malen pomak, na primjer 5 % slike
- **rotacije** slike za kut od $\pm 7\,\mathrm{deg}$
- **skaliranje** (uvećavanje ili smanjivanje) slike za neki faktor koji je obično u raponu od <0.9, 1.1>
- **smik** slike za kut od$\pm 5\,\mathrm{deg}$

Slučajne transformacije se koriste tako da se za svaku sliku prvo odabere podskup filtera koji će se koristiti na toj slici, a zatim se slučajno odabiru parametri svakog od odabranih filtera.

Jedan način korištenja je da se pomoću navednih funkcija unaprijed generira povećana verzija skupa podataka, a drugi da se transformacije sprovode "uživo", odnosno prije svake epohe treniranja. Tokom provođeja transformacija "uživo" mreža se nikad ne trenira s istom slikom dva puta. Svaki put mreža "vidi" drugu verziju (transformiranu) iste slike čime se uvelike sprječava pretreniranost i prevelika prilagodba na skup za učenje.

3.3.3. Dropout

Točnost predikcije raznih modela čije treniranje počinje sa slučajno inicijaliziranim parametrima, se povećava ako se koristi prosjek predikcija više istrenranih modela. Takav način korištenja gotovih modela se naziva uprosječivanje modela (engl. *model averaging*). No, mane takvog pristupa su (i)što treniranje svake mreže traje dugo i (ii) ne želimo tokom korištenja pokretati više modela za isti primjer jer to opet dugo traje što je pogotovo loše za primjenu u realnom vremenu.

Dropout je metoda za učinkovito uprosječivanje modela velikih neuronskih mreža[20]. Uzmimo za primjer jedan skriveni sloj na kojem čemo primjeniti *dropout* metodu. Svaki puta kad mreži na ulaz postavimo jedan primjer iz skupa za učenje, slučajno "ugasimo" svaki neuron s vjerojtnošću p (za skrivene slojeve najčešće je $p = 0.5$, odnosno "ugasimo" polovicu neurona. "Gašenjem" se smatra postavljanje izlaza određenog neurona na 0. To zapravo znači da slučajno odabiremo jednu od 2^H arhitektura mreže, gdje je H broj neurona u skrivenom sloju, a sve arhitekture dijele iste težine. Dijeljenje težina između velikog broja arhitektura, odnosno modela, je zapravo jaka regularizacija mreže. Budući da postoji velik broj arhitektura, svaka će pojedina arhitektura vidjeti samo nekolicinu primjera.

Ostaje još za odgovoriti što se događa tokom testiranja modela (ili korištenja). Prva ideja je odabrati nekoliko različitih arhitektura, napraviti njihove predkcije i uprosječiti ih. No, brži način je koristiti sve neurone skrivenog sloja, a njihove izlaze onda pomnožiti s p. Može se pokazati da je to jednako računanju geometrijske sredine predikcije svih 2^H modela.

Kod mreža s više slojeva, *dropout* kod ulaznih slojeva je manji, koristi se $p = 0.9$ (samo 10% ulaznih neurona se gasi). Neki autori navode da koriste u svim skrivenim slojevima $p = 0.5$, dok neki postepeno smanjuju p od 0.9 (na ulazu), do 0.5 u zadnjem

skrivenom sloju.

Ako se duboka mreža lako prenauči, *dropout* će osjetno smanjiti grešku. No, ako se mreža ne može prenaučiti, onda je potrebno koristiti mrežu s više parametara (širu ili dublju).

4. Semantička segmentacija

Semantička segmentacija je proces pridavanja labele svakom pikselu slike. Labele su obično oznake objekata koji su predstavljeni slikom. Semantička segmentacija spaja klasične probleme detekcije objekata, segmentacije slika i višeklasne klasifikacije u jedan proces.

Prepoznavanje nekog tipa objekta na slici se uglavnom temelji na tome da tražimo objekte slične teksture i sličnih obrisa, ali ponekad niti te dvije stvari ne govore točno koji je to objekt. Takvi zadaci znaju biti teški i za ljudske označivaće, stoga automatizacija takvog zadatka računalima nije nimalo jednostavna.

Postavljaju se dva vrlo važna pitanja: kako dobro prikazati slikovne informacije u sustavu za označavanje scene i kako te informacije iskoristiti za interpretaciju. Budući da je ideja konvolucijskih mreža da rade kao detektori značajki, konvolucijske mreže se u praksi koriste i za izvlačenje značajki i za klasifikaciju.

Razumijevanje scena je jako važno kod mnogih primjena, a posebno kod robotike. Jedan od važnih ciljeva u području robotike je omogućiti robotima da se autonomno kreću kroz prostor koji dijele s ljudima. Najvažnija stvar u implementaciji takvog sustava je prepoznavanje i intepretacija okoline.

slika

4.1. Testni skupovi

Skupovi podataka korišteni u evaluaciji...

4.1.1. Stanford Background Dataset

Vrste oznaka, veličine slika, zastupljenost oznaka - histogram

4.1.2. SIFT Flow

Vrste oznaka, veličine slika, zastupljenost oznaka - histogram Standardni dataset, popis rezultata

5. Implementacija

U okviru ovog rada implementirana je radna okolina[1] (engl. *framework*) za semantičku segmentaciju. U pozadini se koristi alat *Theano*[2] [1] [2] koji služi za definiranje, optimiranje i prevođenje simboličkih izraza u C++ ili CUDA programski kod.

Neke od prednosti korištenja alata *Theano* su:

- omogućuje paraleno korištenje CPU-a i GPU-a što ima za posljedicu da se algoritmi i programi mogu pisati i analizirati na nekom lošijem računalu, dok se ne ustvrdi da rade dobro, a kasnije pokretati na računalima s moćnim grafičkim karticama bez kojih je obzbiljiniji rad s velikim skupovima podataka i dubokim mrežama praktički nezamisliv
- optimiranje simboličkih izraza, što znači da se nijedan izraz neće računati više od jednom
- aritmetička pojednostavljenja, na primjer: $(x \cdot y)/x$ postaje y
- ugrađeno računanje gradijenata
- ugrađene metode za poboljšanje numeričke stabilnosti određenih matematičkih izraza
- ugrađen konvolucijski operator $(*)$

Mana korištenja *Theana* je upravo to što to nije programski jezik, već se u programskom jeziku *Python* pišu izrazi za *Theano*. Za takvo pisanje izraza je potreban malo drugačiji način razmišljanja koji zahtijeva privikavanje.

Iako su neki alati, poput *Caffe-a* i *PyLearna-a*, jednostavniji za korištenje, mnogo je teže nadograđivati funkcionalnosti jer to iziskuje pisanje C++ i CUDA koji mora biti kompatibilan s već napisanim kodom, dok se u *Theanu* samo definira simbolički izraz za novu funkcionalnost. U trenutku početka rada nisu postojali alati za semantičku segmentaciju, a eventualna proširenja postojećih alata za klasifikaciju bila su u razvoju.

[1] https://github.com/iborko/theano-conv-semantic

[2] Theano, http://deeplearning.net/software/theano/

Parametri algoritma poput arhitekture mreže, stope učenja, načina pretprocesiranja, vrste algoritma gradijentne optimizacije, i slično, se definiraju u konfiguracijskoj datoteci u JSON formatu koja se predaje kao argument skriptama za generiranja ulaza (pretprocesiranje) i učenje:

```
python generate.py network.conf
python train.py network.conf
```

gdje je `network.conf` konfiguracijska datoteka u JSON formatu.

TODO: dodati appendix s primjerom jedne konfiguracijske datoteke.

5.1. Podatci

Svaki skup podataka semantički segmentiranih slika koji se koristi mora imati svoju skriptu za ucitavanje u mapi `dataset`. Tamo već postoje skripte `loader_iccv.py` i `loader_sift.py` za učitavanje pripadnih datasetova. Svaka `loader_*.py` skripta mora imati `load_dataset(path)` funkciju koja vraća listu (ili iterator) `Sample` objekata. Klasa `Sample` je definirana u `dataset/sample.py` datoteci i sadrži naziv primjera, ulaznu sliku i segmentiranu sliku. Ulazne slike su u obliku *numpy*[3] tenzora s 3 dimenzije: visina, sirina i broj kanala, a izlazne slike su u obliku *numpy* matrica gdje je element matrice oznaka (redni broj) pripadnog piksela.

5.2. Predprocesiranje

Pomoću konfiguracijske datoteke moguće je definirati niz funkcija koja se koriste tokom predprocesiranja. Neki od primjera funkcija:

yuv pretvara RGB kanale u YUV kanale

int_to_float pretvara intezitete iz raspona od 0 do 255 (cijeli brojevi) u raspon 0 do 1 (decimalni brojevi)

normalize normalizira sliku tako da ima srednju vrijednost inteziteta 0 i standardnu devijaciju 1

[3] *numpy*, ekstenzija jezika *Python* koja dodaje podršku za višedimenzionalne brojevne nizove i podržava razne matematičke operacije nad njima, http://www.numpy.org

5.3. Arhitektura mreže

TODO

5.4. Stopa učenja

Kako utječe na učenje, kad mijenjati, kako.

5.5. Uvjeti zaustavljanja

Načini određivanja trenutka zaustavljanja

6. Rezultati

TODO

6.1. Mjere

Pixel accuracy, mean class accuracy, IOU (intersection over union).

7. Zaključak

Zaključak.

LITERATURA

[1] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, i Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[2] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, i Yoshua Bengio. Theano: a CPU and GPU math expression compiler. U *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Lipanj 2010.

[3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, i Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, 2014. URL `http://arxiv.org/abs/1412.7062`.

[4] Clément Farabet, Camille Couprie, Laurent Najman, i Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1915–1929, 2013.

[5] K. Fukushima. A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:189–203, 2003.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, i Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. U *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[7] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, 2013. URL `http://arxiv.org/abs/1308.0850`.

[8] G E Hinton i R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, Srpanj 2006.

[9] Alex Krizhevsky, Ilya Sutskever, i Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. U F. Pereira, C.J.C. Burges, L. Bottou, i K.Q. Weinberger, urednici, *Advances in Neural Information Processing Systems*, svezak 25, stranice 1097–1105, 2012.

[10] Y. LeCun, L. Bottou, Y. Bengio, i P. Haffner. Gradient-based learning applied to document recognitio. *Proceedings of the IEEE*, 29(3):2278–2324, 1998.

[11] Yann Lecun, Leon Bottou, Genevieve B. Orr, i Klaus-Robert Müller. Efficient backprop. 1998.

[12] Jonathan Long, Evan Shelhamer, i Trevor Darrell. Fully convolutional networks for semantic segmentation. *CVPR (to appear)*, Studeni 2015.

[13] David G. Lowe. Object recognition from local scale-invariant features. U *Proceedings of the International Conference on Computer Vision*, svezak 2 od *ICCV*, stranice 1150–. IEEE Computer Society, 1999. ISBN 0-7695-0164-8.

[14] Andrew L Maas, Awni Y Hannun, i Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. *Proc. ICML*, 30, 2013.

[15] Warren Mcculloch i Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[16] Marvin Minsky i Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[17] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, i Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, 2013. URL `http://arxiv.org/abs/1312.6229`.

[18] T. Serre, L. Wolf, S. Bileschi, i M. Riesenhuber. Robust object recognition with cortex-like mechanisms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):411–426, 2007.

[19] K. Simonyan i A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, i Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, Siječanj 2014. ISSN 1532-4435.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, i Andrew Rabinovich. Going deeper with convolutions. *CoRR*, 2014. URL `http://arxiv.org/abs/1409.4842`.

[22] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Doktorska disertacija, Harvard University, 1974.

[23] Bing Xu, Naiyan Wang, Tianqi Chen, i Mu Li. Empirical evaluation of rectified activations in convolutional network. 2015.

[24] Ning Zhang, Jeff Donahue, Ross Girshick, i Trevor Darrell. Part-based R-CNNs for fine-grained category detection. U *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

[25] Jan Šnajder i Bojana Dalbelo Bašić. *Strojno učenje*. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2012.

Semantička segmentacija prirodnih scena dubokim neuronskim mrežama

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: računalni vid, neuronske mreže, konvolucijske mreže, duboko učenje, semantička segmentacija

Semantic segmentation of outdoor scenes using deep neural networks

Abstract

Abstract.

Keywords: computer vision, neural networks, convolutional networks, deep learning, semantic segmentation