Algoritmi strojnog učenja

Dogančić, Ema

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Ema Dogančić

ALGORITMI STROJNOG UČENJA

Diplomski rad

Voditelj rada:
doc. dr. sc. Zoran Škoda

Zagreb, studeni 2019.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Ovaj diplomski rad posvećujem svojoj obitelji, prijateljima i Domagoju. Hvala vam na podršci, kako tijekom izrade ovog rada, tako i tijekom cijelog studiranja.

Sadržaj

Uvod

Ljudski mozak je toliko zanimljiva pojava u prirodi da je proučavan već tisućama godina. Međutim, tek sredinom dvadesetog stoljeća neuroznanstvenicima je omogućeno razumijevanje mozga na razini neurona. Otkrivanjem načina na koji oni rade otvaraju se nova vrata u shvaćanju kako ljudski mozak, ali i ljudski um, obavljaju svoje zadaće. Fasciniranost umom tada motivira druge grane znanosti da u svoja istraživanja implementiraju nova medicinska otkrića. To je početak umjetnog neurona.

Razvojem umjetne inteligencije se ljudske sposobnosti nastoje simulirati na računalima. Kako bismo na računalo preslikali aktivnosti koje ljudi svakodnevno rade bez poteškoća, dolazi do ideje da se to napravi točno na način na koji to obavlja ljudski mozak i tako dolazi do razvoja neuronskih mreža.

Neuronske mreže razvijaju se od 40-ih godina 20. stoljeća, dok se u zadnjih dvadesetak posebno razvija jedan tip neuronskih mreža, a to su konvolucijske neuronske mreže koje predstavljaju glavni fokus ovog rada.

Prvo poglavlje prikazuje uvod u umjetnu inteligenciju i strojno učenje kako bismo približili tematiku čitatelju te ga kroz nekoliko povijesnih crtica proveli kroz razvoj stvaranja neuronskih mreža. Na kraju ovog poglavlja radi ilustracije malo drugačijeg pristupa umjetnoj inteligenciji navodimo genetički algoritam.

U idućem poglavlju navodimo osnovne pojmove vezane uz strojno učenje i motivaciju za njegovim razvojem.

U trećem poglavlju definiramo umjetne neurone po uzoru na biološke. Opisujemo razne aktivacijske funkcije te dajemo definiciju umjetne neuronske mreže. U nastavku detaljno opisujemo kako umjetne neuronske mreže obrađuju informacije.

Četvrto poglavlje fokus stavlja na učenje neuronskih mreža pomoću gradijentnog spusta. Detaljno raspisujemo *backpropagation* algoritam koji je osnovni algoritam za učenje neuronskih mreža i razlog ponovnog razvoja umjetnih neuronskih mreža od 1980-ih godina.

U zadnjem poglavlju detaljno opisujemo konvolucijske neuronske mreže. Dotičemo se povijesnog aspekta i objašnjavamo zašto je došlo do potrebe za razvojem konvolucijskih neuronskih mreža te predstavljamo matematičku operaciju konvoluciju koja se skriva u pozadini. U nastavku navodimo građu konvolucijskih mreža preko slojeva koji ju čine. Na kraju navodimo njihove prednosti u odnosu na klasične te diskutiramo primjenjivost.

Poglavlje 1

Uvod u umjetnu inteligenciju i strojno učenje

Umjetna inteligencija se proučava već desetljećima i još uvijek je jedna od najistraživanijih tema u računarstvu. 1950. godine Alan Turing[1] je napisao članak u kojem postavlja jedno naizgled jednostavno pitanje — "Mogu li strojevi razmišljati?". Turing predlaže metodu koja daje odgovor na to pitanje i ona se danas naziva Turingov test, a provodi se tako da jedna osoba koju nazivamo ispitivač postavlja pitanja i ako stroj uspije zavarati ispitivača da pomisli kako je on zapravo čovjek, tada je stroj prošao test i može se smatrati inteligentnim. Tek šest godina nakon toga, John McCarthy je u prijedlogu za konferenciju stvorio pojam "umjetne inteligencije" koji je puno godina kasnije definirao kao znanost i inženjerstvo inteligentnih strojeva [6].

Umjetna inteligencija danas ima brojna područja od kojih ćemo se u ovome radu osvrnuti na mali dio jednog od područja umjetne inteligencije — strojno učenje. Ono se velikim dijelom razvilo iz područja prepoznavanja uzoraka i statistike. Proučavajući ljude i njihov način razmišljanja te sposobnost učenja nailazi se na pitanje mogu li i računala učiti. Tako dolazimo do pojma strojnog učenja — kako na stroju simulirati učenje koje ljudi provode svakodnevno? Unutar strojnog učenja postoji više tipova učenja - nadzirano, nenadzirano, učenje pod kontrolom i isto tako više modela, odnosno algoritama, koji omogućuju takva učenja. Mi ćemo se u ovome radu fokusirati na model neuronskih mreža u okviru nadziranog učenja.

[1] Alan Turing (1912.–1954.) britanski matematičar, logičar i kriptoanalitičar. Smatra se ocem modernog računarstva.

1.1 Povijest neuronskih mreža

Inspirirani razvojem neuroznanosti, Warren McCulloch i Walter Pitts 1943. razvijaju prvi model umjetnog neurona. Imao je nekoliko ulaza koji su mogli poprimiti samo vrijednosti 0 ili 1. Aktivacijska funkcija je bila *step*-funkcija s nekim pragom θ, odnosno ako ulazi u sumi daju broj veći ili jednak θ, tada je izlaz neurona jednak 1, a u suprotnom je izlaz jednak 0. Pokazali su da se logički operatori AND i OR mogu prikazati preko takvih neurona, a onda se i sve logičke funkcije mogu prikazati kombinacijom tih neurona, to jest umjetnom neuronskom mrežom. Ali se javljalo nekoliko problema s ovako konstruiranim neuronom - ulazi i izlazi su binarni, prag θ može poprimiti samo neke vrijednosti i svi ulazi imaju jednake težine.

Idući model neurona razvio je Frank Rosenblatt 1958. godine iz teorije neuroznanstvenika Donalda Hebba i prijašnjeg rada McCullocha i Pittsa i nazvao ga je perceptron. Hebbova teorija govori kako se jačina sinapse selektivno pojačava ako se neuroni sa susjednih strana sinapse kontinuirano međusobno aktiviraju. Iz Hebbove teorije nastaje pravilo učenja koje govori na koji način se težine perceptrona trebaju modificirati kako bi mogao naučiti određenu funkciju. 1962. dokazana je konvergencija algoritma za učenje težina u perceptronu koji je iznio Rosenblatt. Perceptron je prvo bio zamišljen kao stroj, a ne kao računalni program. Njegova implementacija ugrađena je u hardver koji je bio napravljen za prepoznavanje slika.

Iako se na početku činio kao obećavajuć model, ubrzo je dokazano da se perceptron ne može učiti kako bi prepoznao puno klasa obrazaca. Marvin Minsky i Seymour Papert u svojoj knjizi *Perceptrons: An Introduction to Computational Geometry* iz 1969. godine dokazuju kako perceptron ne može naučiti funkciju XOR.

Perceptron je osmišljen slično kao i prvi model McCullocha i Pittsa, imamo nekoliko ulaza, ali je svakom ulazu pridijeljena neka težina. U jedinici se ulazi množe pripadajućim težinama i zbrajaju te ako daju vrijednost veću od nekog praga θ izlaz perceptrona je 1, a u suprotnom 0. Preciznije, za ulaze $x_1, \ldots, x_n$ i odgovarajuće težine $w_1, \ldots, w_n$ perceptron kao izlaz daje 1 ako je $\sum_{i=1}^{n} x_i w_i \geq \theta$, odnosno 0 ako je $\sum_{i=1}^{n} x_i w_i < \theta$. Drugim riječima, perceptron ulaz dijeli u dvije kategorije. To možemo promatrati i na sljedeći način: $\sum_{i=1}^{n} x_i w_i = \theta$ predstavlja hiperravninu koja prostor dijeli na dva dijela. Ako promatramo perceptron sa samo dva ulaza x_1 i x_2, tada $x_1 w_1 + x_2 w_2 = \theta$ zbog varijabilnosti parametara w_1, w_2 i θ predstavlja sve pravce u ravnini. To znači da perceptron možemo zamisliti kao pravac u ravnini i sve točke s jedne strane pravca pripadaju jednoj kategoriji, dok sve točke s druge strane pravca pripadaju drugoj kategoriji. Ako imamo skup točaka u ravnini i ako ih pravcem možemo podijeliti u dvije kategorije, kažemo da su točke linearno odvojive (linearly separable), inače kažemo da nisu linearno odvojive. Promotrimo sada perceptron za XOR. Imamo sljedeći skup točaka $(0, 0)$, $(0, 1)$, $(1, 0)$ i $(1, 1)$ gdje za $(0, 0)$ i $(1, 1)$ izlaz treba biti 0, a za ostale dvije točke 1. To možemo zapisati sljedećim

jednadžbama

$$\begin{aligned} 0w_1 + 0w_2 &< \theta, \\ 1w_1 + 1w_2 &< \theta, \\ 0w_1 + 1w_2 &\geq \theta, \\ 1w_1 + 0w_2 &\geq \theta, \end{aligned}$$

odnosno

$$\begin{aligned} \theta &> 0, \\ w_1 + w_2 &< \theta, \\ w_2 &\geq \theta, \\ w_1 &\geq \theta, \end{aligned}$$

iz čega odmah vidimo kontradikciju. To pokazuje da ove točke nisu linearno odvojive pa perceptron ne može izračunati funkciju XOR.

Zbog toga je istraživanje neuronskih mreža stagniralo neko vrijeme sve do otkrića i korištenja više slojeva u neuronskim mrežama. Otkriveno je da korištenjem dvaju ili više slojeva u perceptronu omogućuje značajno povećanje njegove snage u odnosu na korištenje perceptrona s jednim slojem. Nakon toga dolazi i do stvaranja novih neuronskih mreža kao što su *feedforward neural networks* s nelinearnim aktivacijskim funkcijama. 1960-ih godina također je razvijen i *backpropagation* algoritam koji omogućuje podešavanje težina skrivenih slojeva neuronske mreže kako bi se prilagodili novoj situaciji i upravo taj algoritam će biti centralni dio ovog rada. Zahvaljujući njemu dolazi do preporoda u neuronskim mrežama, iako tek dvadesetak godina poslije [17].

1.2 Genetički algoritmi

Genetički algoritmi su vrsta evolucijskih algoritama koje je predstavio John Holland 1960-ih godina. Kao i mnogi drugi algoritmi u umjetnoj inteligenciji, tako i ovi algoritmi ideju vuku iz biologije. Bazirani su na konceptu Darwinove teorije evolucije [4] koja se poziva na prirodni odabir. Kako bi se izbjegao eksponencijalni porast broja jedinki dolazi do borbe za opstanak u kojoj preživljavaju, i onda se dalje razmnožavaju, jedinke koje su bolje prilagođene uvjetima okoline.

Genetički algoritam započinje s nekim skupom koji zovemo populacija. Populacija se sastoji od određenog broja jedinki. Svaka jedinka je ocijenjena funkcijom podobnosti (engl. *fitness function*) koja podobnijim jedinkama pridružuje veće vrijednosti. Kao i u Darwinovoj teoriji evolucije, podobnije jedinke imaju veću vjerojatnost za opstankom, pa tako i za razmnožavanjem. Razmnožavanjem dolazi do rekombinacije gena dviju jedinki

i stvaranja nove generacije populacije koja će zbog rekombinacije biti malo prilagođenija okolini nego prošla generacija. Također, postoji i vjerojatnost da će se u novom potomku dogoditi mutacija koja će ga time malo izmijeniti. Mutacija je bitna jer unosi promjene u populaciju i time povećava prostor rješenja.

```
function GeneticAlgorithm(population, Fitness)
input : population = skup jedinki
        Fitness = funkcija podobnosti
output: jedinka
while (neka jedinka nije dovoljno podobna ili nije prošlo određeno vrijeme) do
    new_population = ∅
    for i = 1 to Size(population) do
        x = RandomSelection(population, Fitness)
        y = RandomSelection(population, Fitness)
        child = CrossOver(x, y)
        if (slučajna mala vjerojatnost) then
            child = Mutate(child)
        end
        new_population += child
    end
    population = new_population
end
return najpodobnija jedinka s obzirom na funkciju Fitness
```

Slika 1.1: Opći oblik genetičkog algoritma.

Ovaj opis možemo zapisati u obliku algoritma kao na slici 1.1, po uzoru na [25], gdje funkcija

- Fitness predstavlja funkciju podobnosti; za svaku jedinku evaluira koliko je ona podobna,
- RandomSelection iz populacije odabire jedinke s obzirom na funkciju podobnosti, što znači da jedinke s većom vrijednošću te funkcije imaju veću vjerojatnost biti odabrane,
- CrossOver križa dvije odabrane jedinke i vraća tako dobivenu jedinku,
- Mutate radi mutaciju jedinke, odnosno malu promjenu nad jedinkom unoseći na taj način varijaciju u novu generaciju.

Prirodna selekcija, rekombinacija i mutacija čine genetičke operatore. Oni se koriste kako bi se stvorile i održale genetičke različitosti (mutacija), kombinirala postojeća rješenja u novo (rekombinacija) te odabrala bolja rješenja (selekcija).

Genetički algoritmi primjenjuju se u velikom broju područja. Često se koriste za rješavanje NP-teških problema kao što su problem trgovačkog putnika i problem raspodjele poslova (job shop scheduling) [10].

Neka od područja u kojima se primjenjuju genetički algoritmi su softversko inženjerstvo za procjenu troškova softvera, raspodijeljeno računanje za raspoređivanje poslova, strojno učenje (za prepoznavanje uzoraka) i odabir značajki. Također se mogu primjenjivati i u učenju neuronskih mreža, ali i u mnogim drugim područjima [30].

Poglavlje 2

Osnove strojnog učenja

Glavni izvori za ovo poglavlje su [19, 29].

2.1 Učenje

Kako bismo mogli bolje razumjeti pojam strojnog učenja, pogledajmo prvo nekoliko primjera učenja koje se javlja u prirodi među životinjama.

Istraživanja su pokazala da štakori uče izbjeći otrovne mamce. Prilikom susreta s novom hranom prvo probaju malu količinu i ako ona izazove negativan učinak, štakori će ga povezati s tom vrstom hrane te ju više neće jesti. Vidimo da štakori koriste svoje iskustvo s prijašnjom hranom kako bi otkrili sigurnost nove. Ukoliko je prošlo iskustvo s nekom određenom hranom bilo negativno, štakori predviđaju da će i svako buduće susretanje također imati loš utjecaj.

Po uzoru na učenje štakora, mogli bismo zamisliti i algoritme koji bi učili na sličnom principu. Na primjer, želimo programirati računalo da filtrira neželjenu poštu. Prva ideja po uzoru na ovo učenje bila bi da zapamtimo sve prijašnje poruke koje su označene kao neželjene te kada se pojavi nova računalo će proći kroz sve one već zapamćene. Ako tamo pronađe identičnu poruku, onda će i novu označiti kao neželjenu, inače neće. Ovaj pristup često je koristan, ali ima veliki nedostatak koji sustavi za učenje ne bi smjeli imati, a to je da ne može točno označiti poruke koje dosad nije vidio. Sposobnost dobre procjene neviđenih podataka nazivamo generalizacijom i cilj strojnog učenja je razviti modele koji omogućuju dobru generalizaciju.

Ranije navedeni princip učenja štakora nije onakav kakav se na početku čini. Naime, pri susretu s određenom vrstom hrane koja uzrokuje negativan učinak, štakori će izbjegavati svaku novu vrstu hrane koja ih mirisom i okusom podsjeća na neku od odbojnih vrsta.

Sličan pristup možemo iskoristiti i kako bismo poopćili postupak filtriranja neželjene pošte na način da iz svake neželjene poruke izvučemo skup riječi koji se u poruci pojavljuje,

a asocira na neželjenu poruku te kada stigne nova poruka, računalo traži pojavljuju li se neke od sumnjivih riječi iz drugih poruka i u skladu s tim označuje je li poruka neželjena ili ne.

Iako u nekim situacijama ovaj pristup može biti dobar, to nije uvijek slučaj. Pogledajmo drugi primjer iz životinjskog učenja. Eksperiment je proveo američki psiholog B. F. Skinner, a radi se o tome da golubovi mogu biti praznovjerni (engl. *pigeon superstition*). Skinner je u eksperimentu postavio gladne golubove u kavez gdje su svakih 15 sekundi dobivali hranu. Golubovi su se kretali po kavezu i u trenutku stizanja hrane svaki od njih je obavljao neku radnju. Golubovi su obavljanje tih radnji povezivali s dobitkom hrane pa su ih počeli raditi sve češće. To jest, golubovi su vjerovali da dobivaju hranu zbog određenih postupaka, iako je ona ubacivana automatski. Ovu pojavu objašnjavamo praznovjernošću.

Jedan od ciljeva strojnog učenja je razviti algoritme koji će moći razlikovati praznovjerno učenje kao kod golubova od stvarnog učenja koje imaju štakori. Dok se ljudi mogu osloniti na zdrav razum kako bi filtrirali stvarno učenje od besmislenog, da bismo slično omogućili i računalima, trebali bismo definirati principe koje će računalo zaštiti od besmislenog učenja, što nije uvijek lako pa čak ni moguće.

Koja je razlika između učenja golubova i štakora? Provođeni su drugi eksperimenti na štakorima u kojima su negativni učinci hrane prikazani kao elektrošokovi ili zvučni signali. U tim slučajevima štakori nisu izbjegavali hranu koja izaziva takve učinke kao da imaju urođeno znanje koje im govori da veza između hrane i mučnine može biti međusobno uzrokovana, tako znaju da nema povezanosti između hrane i elektošokova ili zvuka. Za razliku od štakora, golubovi su bili spremni prilagoditi se bilo čemu kako bi dobili hranu. U procesu učenja štakori uspijevaju otkriti neke uzorke ignorirajući druge prividne povezanosti među događajima. Iz toga možemo zaključiti da je prethodno znanje jako bitno za razvijanje uspješnih algoritama učenja. Što je prethodno znanje jače, to će algoritam lakše učiti iz sljedećih primjera, ali će zato biti i manje fleksibilan.

2.2 Definicija učenja i primjeri

Navedimo jednu od općenitijih definicija učenja. Algoritam strojnog učenja je bilo koji algoritam koji ima sposobnost učenja iz podataka. Tom Mitchell u knjizi *Machine Learning* daje sljedeću definiciju učenja.

Definicija 2.2.1. *Kaže se da računalni program uči iz iskustva E s obzirom na neku klasu zadataka T i mjeru uspješnosti P, ako se njegova uspješnost na zadacima iz T, mjerena mjerom P, poboljšava s iskustvom E.*

Računalni program koji uči igrati šah može poboljšavati svoju uspješnost, mjerenu svojom sposobnošću da pobijedi na klasi zadataka koja uključuje igranje šaha, putem iskustva stečenog igranjem igara šaha protiv samog sebe. U ovom primjeru bi klasa T bila igranje

šaha, mjera uspješnosti P bi bila postotak pobijeđenih igara i iskustvo E bi bilo igranje igara šaha protiv samog sebe.

Pomoću ovakve definicije možemo uključiti većinu zadataka koje bismo smatrali zadacima učenja. Zadaci se obično opisuju u uvjetima kako bi algoritam trebao procesuirati primjer. Primjer uglavnom predstavljamo uređenom n-torkom $(x_1, \ldots, x_n)$, gdje svaki x_i nazivamo značajkom (engl. *feature, attribute*). Značajke dobivamo mjerenjima objekata ili nekih događaja koje želimo obrađivati algoritmom. Sada navodimo neke od najpoznatijih zadataka učenja. To su:

- klasifikacija — u ovom tipu zadataka želimo odrediti u koju od k klasa pripada neki podatak. Njega rješavamo tako da nađemo funkciju $f\colon \mathbb{R}^n \to \{1, \ldots, k\}$. Pridruživanjem $y = f(\vec{x})$, y je numerička reprezentacija neke od klasa, dok je $\vec{x}$ reprezentacija ulaznog podatka. Primjer klasifikacije je na temelju pacijentovih nalaza utvrditi radi li se o malignom ili benignom tumoru. Ovakav primjer nazivamo binarnom klasifikacijom zbog prisutnosti samo dviju klasa.

- regresija — u kojoj je cilj predvidjeti neku numeričku vrijednost na temelju ulaznog podatka. Ovaj zadatak rješavamo tako da nađemo funkciju $f\colon \mathbb{R}^n \to \mathbb{R}$. Slično je klasifikaciji, samo što umjesto nekog diskretnog izlaza sada možemo imati kontinuirani. Primjer regresije je na temelju nekih podataka o kućama predvidjeti cijenu kuće.

- grupiranje — u kojemu za razliku od prethodna dva tipa zadataka ne tražimo funkciju koja povezuje ulaz i izlaz podataka nego pokušavamo naći odnose između ulaznih podataka na način da ih podijelimo u podskupove, odnosno grupe, tako da slični podaci budu u istoj grupi, a drugačiji podaci budu u različitim grupama. Podaci u grupi dijele neko zajedničko obilježje.

2.3 Podjela strojnog učenja

Grubo strojno učenje, ovisno o načinu učenja, dijelimo na nadzirano učenje (engl. *supervised learning*), nenadzirano učenje (engl. *unsupervised learning*) i učenje podrškom (engl. *reinforcement learning*).

Nadzirano učenje i teorija statističkog učenja

Nadzirano učenje radi na principu učenja uz pomoć učitelja. Imamo ulazne podatke $\vec{x}$ i njihove pripadne izlazne vrijednosti y te na temelju uređenih parova ulaza i izlaza $(\vec{x}, y)$ pokušavamo naći funkciju h koja će ulazu x pridružiti vrijednost $h(\vec{x})$ uz što manju pogrešku, tj. tako da je $h(\vec{x}) \approx y$ koju onda možemo koristiti za predviđanje novih podataka.

Formalno, prema [29] u osnovnoj postavi statističkog učenja imamo sljedeće:

- domena ili skup ulaznih podataka: proizvoljan skup $\mathcal{X}$ je skup primjera. Elemente tog skupa još nazivamo i instancama.
- skup izlaznih podataka: proizvoljan skup $\mathcal{Y}$ je skup vrijednosti pridruženih ulaznim podacima.
- skup primjera za učenje (engl. *training set*): $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)\}$ je podskup Kartezijevog produkta $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. Predstavlja podatke kojima algoritam može pristupiti u ovom tipu učenju.
- izlaz: tražimo od algoritma da vrati *hipotezu* $h\colon \mathcal{X} \to \mathcal{Y}$.

Osnovna pretpostavka je da postoji fiksna nepoznata vjerojatnosna distribucija na $\mathcal{X}$. Označimo tu distribuciju s $P(x)$. Primjeri $\vec{x}_1, \ldots, \vec{x}_m \in \mathcal{X}$ su uzorkovani međusobno nezavisno iz distribucije $P(x)$. Za svaki $\vec{x}_i$ dobivamo vrijednost y_i u skladu s uvjetnom vjerojatnosnom distribucijom $P(y \mid x)$ koja je fiksna, ali nepoznata. Ovo je općenitiji slučaj koji uključuje slučaj kad koristimo funkciju $f(x) = y$ [33]. Tako dobivamo skup S. Označimo s $\mathcal{H}$ skup svih funkcija $f\colon \mathcal{X} \to \mathcal{Y}$, nazivamo ga *prostorom hipoteza*. Cilj je pretražiti prostor hipoteza i pronaći "najbolju" hipotezu $h \in \mathcal{H}$ tako da $h(\vec{x}) \approx y$.

Kako bismo mogli mjeriti koliko je neka hipoteza "dobra", moramo imati funkciju gubitka (engl. *loss function, cost function*) $L(\vec{x}, y, h(\vec{x}))$ kojom mjerimo koliku količinu korisnosti smo izgubili predviđanjem izlaza $h(\vec{x})$ kada je stvarna vrijednost izlaza y za dani $\vec{x}$.

Neka je $\mathcal{E}$ skup svih mogućih parova $\vec{x} \in \mathcal{X}$ i $y \in \mathcal{Y}$. Očekivani gubitak na svim primjerima za hipotezu h u odnosu na funkciju gubitka L prema [25] je

$$G_L(h) = \sum_{(\vec{x},y)\in\mathcal{E}} L(\vec{x}, y, h(\vec{x}))P(x, y), \tag{2.1}$$

gdje je $P(x, y)$ zajednička distribucija i vrijedi $P(x, y) = P(x)P(y \mid x)$. S obzirom da je distribucija $P(x, y)$ nepoznata, možemo samo odrediti procjenu očekivanog gubitka na primjerima koje imamo u skupu S

$$E_{L,S}(h) = \frac{1}{m} \sum_{(\vec{x},y)\in S} L(\vec{x}, y, h(\vec{x})). \tag{2.2}$$

ERM princip (engl. *empirical risk minimization principle*) kaže da trebamo uzeti hipotezu $\hat{h}$ koja minimizira $E_{L,S}(h)$

$$\hat{h} = \min_{h\in\mathcal{H}} E_{L,S}(h).$$

Klasifikacija i regresija su primjeri nadziranog učenja.

Nenadzirano učenje i učenje podrškom

Za razliku od nadziranog učenja gdje u skupu za učenje imamo i pripadne izlazne vrijednosti $y \in \mathcal{Y}$, u nenadziranom učenju imamo samo ulazne vrijednosti $\vec{x} \in \mathcal{X}$ i želimo pronaći veze između podataka. Pokušavamo naučiti vjerojatnosnu distribuciju $P(x)$ ili neka svojstva te distribucije [19]. Primjer nenadziranog učenja je grupiranje.

Učenje podrškom ima nešto drugačiji pristup od nadziranog i nenadziranog. Ono se bavi pitanjem kako agent koji je u interakciji sa svojim okolišem može naučiti odabirati najbolje akcije kako bi došao do nekog cilja.

Agent se nalazi u okolišu koji možemo opisati nekim skupom stanja S. U svakom trenutku on može učiniti neku od akcija iz skupa akcija A i iz trenutnog stanja prijeći u neko drugo. Izvođenjem akcije a_1 u stanju s_1 agent dobiva neku nagradu $r_1 = r(a_1, s_1)$ te prelazi u stanje s_2, zatim ponovno odabire neku akciju a_2, dobiva nagradu $r_2 = r(a_2, s_2)$ i tako dalje. Ovako dobivamo niz stanja s_i, akcija a_i i nagrada r_i. Cilj agenta je naučiti birati akcije koje će maksimizirati očekivanu kumulativnu nagradu

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots, \qquad 0 \leq \gamma < 1.$$

Poglavlje 3

Neuronske mreže

3.1 Biološka pozadina neuronskih mreža

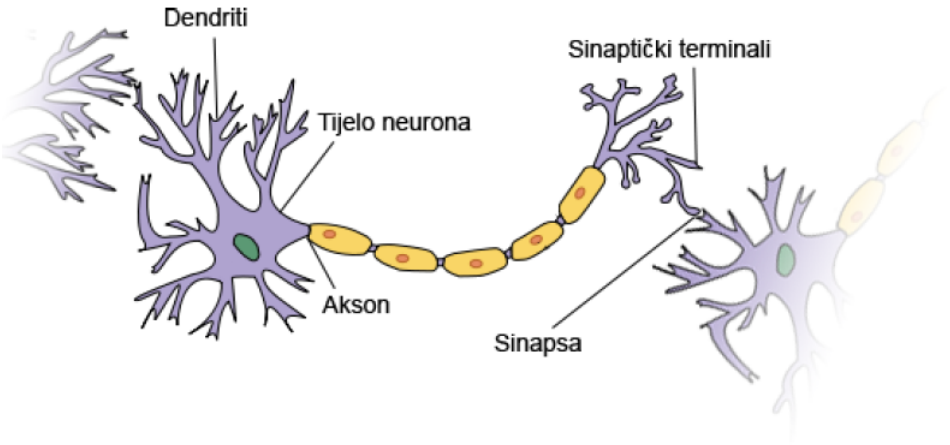
Umjetne neuronske mreže razvile su se po uzoru na živčani sustav. Živčani sustav se sastoji od mreže živčanih stanica koje nazivamo neuroni. Kao što možemo vidjeti na slici 3.1 neuron se sastoji od

- dendrita koji služe za primanje signala,
- tijela neurona koje obrađuje te signale,
- aksona koji prenose signal do sinaptičkih terminala i
- sinaptičkih terminala koji dalje signale prenose na iduće neurone.

Tijelo neurona obrađuje primljene signale na način da povećava ili smanjuje vrijednost električnog potencijala neurona. Ako ukupna vrijednost neurona pijeđe određeni prag, kažemo da se neuron "pali" i dolazi do slanja novog signala preko aksona do sinaptičkih terminala u kojima se stvaraju neurotransmiteri koje potencijal prenose na dendrite idućeg neurona i gornji postupak se ponovno odvija [15].

3.2 Umjetni neuroni i definicija neuronske mreže

Umjetni neuron funkcionira slično. Ima ulazne jedinice koje odgovaraju dendritima i prenose signale koji su opisani numerički. Oni se zatim množe nekim težinama i u tijelu neurona se zbrajaju te u odnosu na dobivenu vrijednost neuron daje izlazni signal koji se šalje idućim neuronima. Umjetni neuron predstavljen je slikom 3.2, a na njoj se vidi i računanje izlazne vrijednosti neurona:



Slika 3.1: Građa živčane stanice (neurona). Izvor [11].

prvo izračunamo težinsku sumu ulaza

$$net = \sum_{i=0}^{n} w_i x_i,$$

a zatim primijenimo funkciju φ da dobijemo izlaz neurona

$$o = \varphi(net) = \varphi\left(\sum_{i=0}^{n} w_i x_i\right),$$

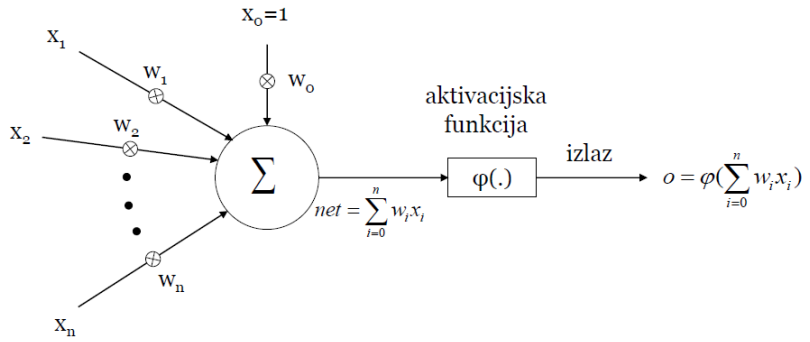
koji postaje ulaz za neurone idućeg sloja s kojima je povezan.

Nueron modeliramo funkcijom koja prima jedan ili više ulaza i na gore opisani način računa izlaz. Funkciju neurona φ nazivamo aktivacijska funkcija ili prijenosna funkcija. Neke od najčešćih aktivacijskih funkcija su

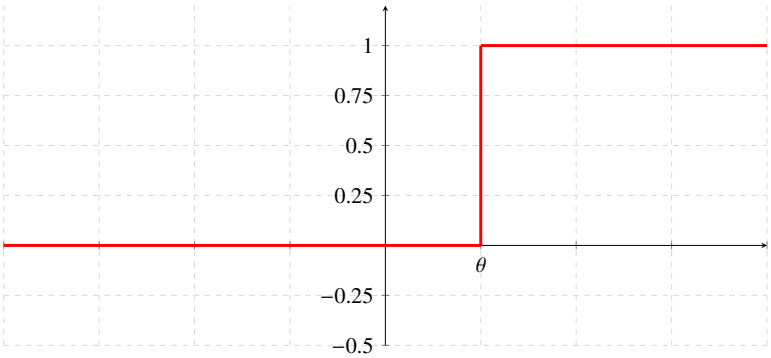
- *step*-funkcija

$$\varphi(x) = \begin{cases} 0, & \text{za } x < \theta \\ 1, & \text{za } x \geq \theta \end{cases},$$

gdje je $\theta \in \mathbb{R}$. Ona se često koristi u izlaznom sloju kod binarne klasifikacije, gdje izlaz 0 predstavlja jednu klasu, a izlaz 1 drugu klasu. Problem kod ove aktivacijske funkcije je što ako u cijeloj mreži koristimo samo nju, možemo naučiti klasificirati jedino linearno odvojive klase. Također, druge mane su da se ne može koristiti za



Slika 3.2: Općeniti izgled umjetnog neurona. Izvor [11].



Slika 3.3: Graf *step*-funkcije.

klasificiranje podataka u više od dvije klase te kako je gradijent funkcije jednak 0, nije korisna ako se primjenjuje *backpropagation* algoritam.

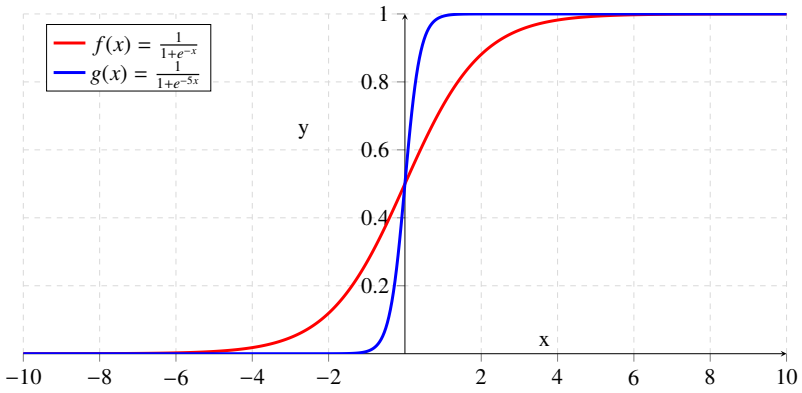
- Sigmoidalna ili logistička funkcija

$$\varphi(x) = \frac{1}{1 + e^{-ax}},$$

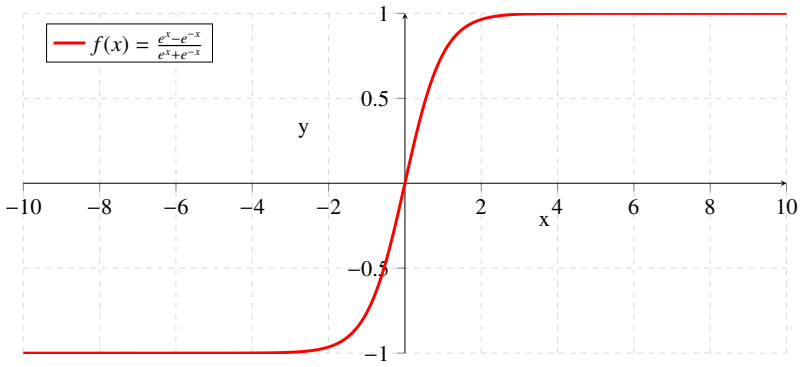
gdje je $a \in \mathbb{R}$ i određuje nagib funkcije. Najčešće se koristi $a = 1$. Prednost sigmoidalne funkcije je što je derivabilna u svim točkama, što je bitno u nekim algoritmima kao što ćemo vidjeti kasnije. U 1990-im godinama korištena je kao zadana aktivacijska funkcija za neuronske mreže [13].

- tangens hiperbolni

$$\varphi(x) = \text{th}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$



Slika 3.4: Graf sigmoidalne funkcije.



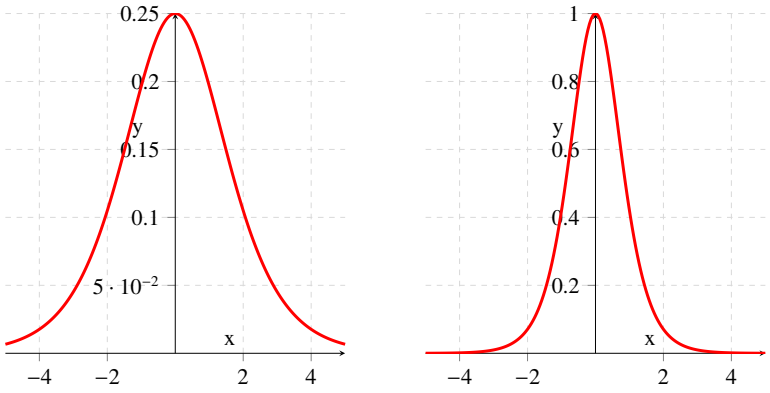
Slika 3.5: Graf tangensa hiperbolnog.

Lako vidimo da se tangens hiperbolni ne razlikuje puno od logističke funkcije. Naime, vrijedi

$$\text{th}(x) = 2\sigma(2x) - 1,$$

gdje σ predstavlja logističku funkciju. Prema [19] ako se moraju koristiti funkcije koje imaju krivulju u obliku slova S kao što su logistička funkcija i tangens hiperbolni, tada se preferira korištenje tangensa hiperbolnog. On se najviše počeo koristiti krajem 1990-ih te u 2000-im jer je često pokazivao bolje rezultate od logističke funkcije te je uz tu aktivacijsku funkcije lakše učenje mreže [13].

Aktivacijske funkcije tangens hiperbolni i logistička funkcija nailaze na isti problem prilikom korištenja *backpropagation* algoritma. Kako te funkcije preslikavaju cijeli $\mathbb{R}$ u intervale $[-1, 1]$, odnosno $[0, 1]$, jako veliki brojevi preslikani su u broj približno jednak 1, dok su jako mali brojevi preslikani u broj približno jednak -1, odnosno 0 te su zbog toga jako osjetljive jedino za ulaze oko 0. To može otežati učenje na temelju



(a) derivacija logističke funkcije

(b) derivacija tangensa hiperbolnog

Slika 3.6: Grafovi derivacija logističke funkcije i tangensa hiperbolnog.

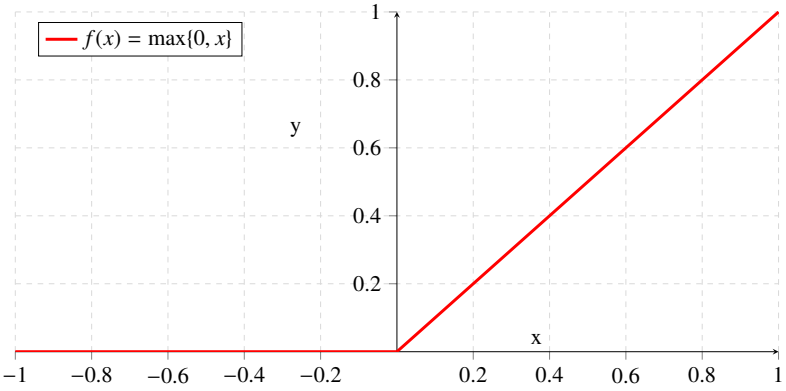
gradijentnog spusta pa se zbog toga sve češće te funkcije koriste samo na izlaznom sloju. Problem koji se javlja prilikom korištenja *backpropagation* algoritma u dubokim mrežama, to jest neuronskim mrežama s puno slojeva, naziva se problemom nestajanja gradijenta (engl. *vanishing gradient problem*). Ako pogledamo sliku 3.6 vidimo da derivacija logističke funkcije može postići najveću vrijednost 0.25, dok derivacija tangensa hiperbolnog može postići najveću vrijednost 1. S obzirom da kod propagiranja pogreške unazad do nižih slojeva dolazi do množenja gradijenata, vrijednosti postaju toliko male da se težine uopće ne mijenjaju i mreža ne uči [13].

- ReLU (engl. *Rectified Linear Unit*) definirana s

$$\varphi(x) = \max\{0, x\},$$

predstavljena je 2000. godine s biološkom motivacijom, a 2011. je po prvi put demonstrirano da njeno korištenje može omogućiti bolje učenje u dubokim neuronskim mrežama [8]. Ostale prednosti ReLU su jednostavno računanje gradijenata, lakša implementacija te ponašanje slično linearnim funkcijama jer je po dijelovima linearna [13]. Mana ReLU je što kada je ulaz negativan gradijent je 0 što može uzrokovati da se neke težine više nikad ne mijenjaju te u tim čvorovima mreža više ne uči. Postoje razne modifikacije ReLU koje smanjuju broj takvih čvorova. Zbog svojih prednosti ReLU i njene modifikacije se sada preporučuju za korištenje kao aktivacijske funkcije u skrivenim slojevima dubokih neuronskih mreža.

Kao što smo već rekli, neuronske mreže ponovno su se intenzivnije počele koristiti nakon otkrivanja slojeva i stvaranja *neuronskih mreža s propagacijom unaprijed* (eng. *fe-*



Slika 3.7: Graf ReLU aktivacijske funkcije.

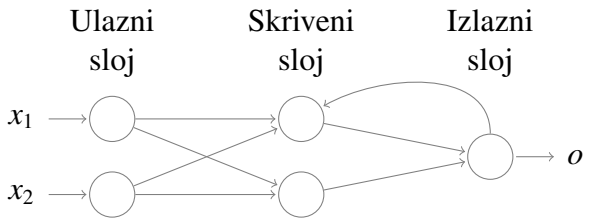
edforward neural network). Neuronske mreže možemo predstaviti usmjerenim grafom povezanih neurona, pa ovisno o grafu imamo više vrsta neuronskih mreža. Neuronske mreže s propagacijom unaprijed su one koje imaju usmjereni aciklički graf, odnosno veze između neurona idu samo u jednom smjeru — nema povratnih veza. Postoji još i druga vrsta neuronskih mreža koje nazivamo *neuronskom mrežom s povratnom vezom* (engl. *recurrent neural network*) jer su predstavljene usmjerenim grafom koji može imati cikluse. Ove mreže za razliku od onih s propagacijom unaprijed mogu modelirati kratkotrajno pamćenje što ih čini modelom sličnijem mozgu i stoga zanimljivijim [25]. Primjer neuronske mreže s povratnom vezom vidi se na slici 3.8. U svim primjerima neuronskih mreža kružići predstavljaju neurone, a strelice usmjerene bridove.

Definicija 3.2.1. *Neuronska mreža s propagacijom unaprijed (FNN) je uređena četvorka* (V, E, σ, w) *gdje su* V *i* E *vrhovi, odnosno bridovi, usmjerenog acikličkog grafa,* $w\colon E \to \mathbb{R}$ *funkcija težina, te* $\sigma\colon \mathbb{R} \to \mathbb{R}$ *aktivacijska funkcija.*

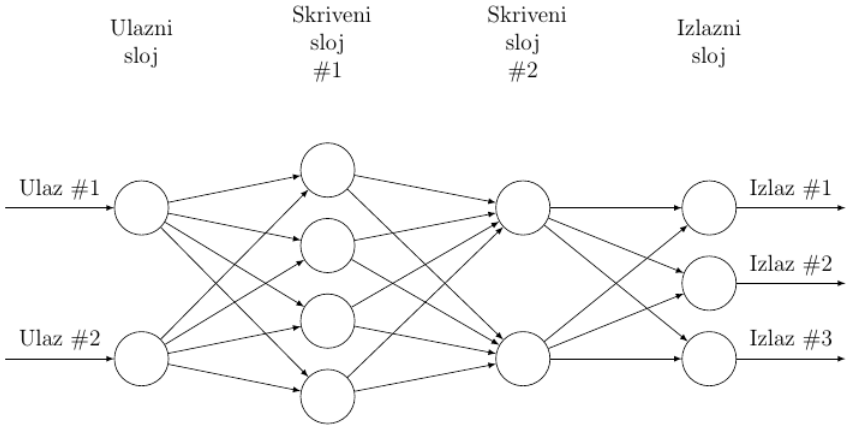
Vrhovi odgovaraju neuronima mreže, a bridovi vezama između tih neurona. Svaki neuron modeliran je kao skalarna funkcija σ. Svaki usmjereni brid povezuje izlaz jednog neurona s ulazom drugog neurona. Ulaz neurona dobije se kao težinski zbroj u odnosu na težine w izlaza svih neurona iz kojih bridovi ulaze u dani neuron [29].

Definicija 3.2.2. *Neka je* (V, E, σ, w) *neuronska mreža s propagacijom unaprijed. Ako možemo skup vrhova* V *particionirati tako da* $V = \bigcup_{l=0}^{L} V_l$ *i svaki brid iz* E *povezuje neki vrh u* V_{l-1} *s nekim vrhom iz* V_l*, za* $l \in \{1, \ldots, L\}$*, onda kažemo da je* (V, E, σ, w) *slojevita neuronska mreža s propagacijom unaprijed [29].*

Ubuduće ćemo promatrati samo takve neuronske mreže i koristit ćemo sljedeće nazive:



Slika 3.8: Primjer neuronske mreže s povratnom vezom.



Slika 3.9: Primjer slojevite neuronske mreže s propagacijom unaprijed. Izvor [23].

- sloj V_0 nazivamo ulaznim slojem (eng. *input layer*) i on se sastoji od $n + 1$ neurona, gdje je n dimenzija ulaznog prostora, a zadnji neuron je konstantan i on uvijek daje izlaz 1,
- slojeve $V_1 \dots V_{L-1}$ nazivamo skrivenim slojevima (eng. *hidden layer*),
- sloj V_L nazivamo izlaznim slojem (eng. *output layer*).

Kažemo da je L broj slojeva ili dubina neuronske mreže. S obzirom da neuroni ulaznog sloja ne predstavljaju prave neurone, jer na njega ne primjenjujemo aktivacijsku funkciju, nego vrijednosti koje u njega ulaze predstavljaju ulaz neuronske mreže, ulazni sloj ne računamo u broj slojeva.

Primjer neuronske mreže s propagacijom unaprijed se vidi na slici 3.9. Ta mreža ima tri izlaza, dva ulaza i dva skrivena sloja s po četiri i dva neurona.

Jednom kad smo specificirali neuronsku mrežu parametrima (V, E, σ, w), označimo s n broj neurona u ulaznom sloju bez *biasa*, a s K broj neurona u izlaznom sloju, tada skup

funkcija $h\colon \mathbb{R}^n \to \mathbb{R}^K$ predstavlja prostor hipoteza za učenje. Obično prostor hipoteza prediktora neuronske mreže definiramo nakon što fiksiramo graf (V, E) i aktivacijsku funkciju σ te prostor hipoteza ovisi samo o funkciji težina w

$$\mathcal{H}_{V,E,\sigma} = \{h_{V,E,\sigma,w} | w\colon E \to \mathbb{R}\}$$

i zato funkcija težina predstavlja parametre modela koje želimo naučiti.

Funkcija w svakom bridu pridružuje realni broj koji nazivamo težinom tog brida. Kako je E konačan skup bridova, funkciju w možemo zamišljati kao vektor $\mathbf{w} \in \mathbb{R}^{|E|}$ težina. Označimo funkciju koju računa neuronska mreža, kad je funkcija težina w definirana s $\mathbf{w}$, kao $h_{\mathbf{w}}\colon \mathbb{R}^n \to \mathbb{R}^K$.

3.3 Propagacija unaprijed

Sad kada smo uveli sve pojmove potrebne za razumijevanje neuronske mreže možemo prijeći na to kako neuronska mreža daje izlaz za određeni ulazni podatak.

Kao što smo imali u poglavlju 2, tako i ovdje imamo skup ulaznih podataka $\mathcal{X}$, skup izlaznih podataka $\mathcal{Y}$, skup primjera za učenje $S = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ ili $S = \{x^{(1)}, \ldots, x^{(m)}\}$ ovisno radi li se o nadziranom ili nenadziranom učenju. Skup primjera za učenje obično se dijeli na dva skupa: skup za učenje i skup za testiranje. Prvi skup služi za učenje mreže, odnosno podešavanje težinskih faktora. Skup za testiranje služi kako bismo ustanovili koliko su podešene težine dobre. Funkciju gubitka ili pogreške L možemo definirati na različite načine, ovisno o zadatku za kojeg učimo neuronsku mrežu. Ovdje ćemo koristiti oznaku $L(\theta)$ gdje θ predstavlja parametre neuronske mreže koje podešavamo, a to su težine. Promotrimo sada neke od najčešćih oblika funkcije pogreške.

Kada se radi o zadatku klasifikacije u K različitih klasa izlazne vrijednosti $y^{(i)}$ mogu biti jednodimenzionalni podaci koji se obično preslikavaju u K-dimenzionalne jedinične vektor stupce u kojima mjesto na kojem je jedinica govori kojoj klasi podatak $x^{(i)}$ pripada, pa zato koristimo $y^{(i)}$ kao vektor, a isto tako i neuronska mreža tada daje funkciju h čiji je izlaz također K-dimenzionalan vektor stupac i najčešće predstavlja vjerojatnost da ulazni podatak pripada određenoj klasi. Tada se koristi sljedeći oblik funkcije troška

$$L(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[y_k^{(i)} \log \left(h(x^{(i)}) \right)_k + \left(1 - y_k^{(i)} \right) \log \left(1 - (h(x^{(i)})) \right)_k \right],$$

gdje je K broj klasa, odnosno broj neurona u izlaznom sloju, a $y_k^{(i)}$ predstavlja k-tu vrijednost vektora $y^{(i)}$ i $(h(x^{(i)}))_k$ predstavlja k-tu vrijednost vektora $h(x^{(i)})$.

Drugi najčešći oblik funkcije troška se koristi u zadacima poput regresije ali i u mnogim drugim i iznosi

$$L(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left((h(x^{(i)}))_k - y_k^{(i)} \right)^2,$$

kod regresije je najčešće $K = 1$.

Kako bismo mogli računati vrijednost funkcije pogreške na skupu podataka, moramo prvo dobiti izlaz neuronske mreže $h_{\mathbf{w}}(x^{(i)})$ za ulazni podatak. Već smo objasnili kako se računa izlaz neurona, s tim da treba naglasiti da za ulazni sloj neuron samo vraća ulaznu vrijednost. Formalnije, označimo s $a_i^{(j)}$ aktivaciju, odnosno izlaz neurona i u sloju j. Tada $a^{(0)}$ predstavlja vektor ulaznih podataka. Kako bismo lakše zapisali računanje aktivacija za preostale slojeve, uvodimo još oznaka.

U slojevitoj neuronskoj mreži s propagacijom unaprijed pridružene vrijednosti težina možemo reprezentirati matricama težina za bridove između svaka dva susjedna sloja. $\mathbf{w}^{(j)}$ označava matricu težina bridova koji izlaze iz neurona u sloju $j-1$ i ulaze u neurone u sloju j tako da su u i-tom retku težine bridova koji ulaze u i-ti neuron u sloju j, a ako gledamo po stupcima matrice, u k-tom stupcu su težine bridova koji izlaze iz k-tog neurona u sloju $j-1$. Ako sa s_j označimo broj neurona u sloju j, tada je dimenzija $\mathbf{w}^{(j)}$ jednaka $s_j \times s_{j-1}$.

Tada se izlaz neurona u ostalim slojevima definira kao

$$a^{(j)} = \sigma\left(w^{(j)} a^{(j-1)} + b^{(j)} \right), \quad 1 \leq j \leq L, \tag{3.1}$$

gdje je

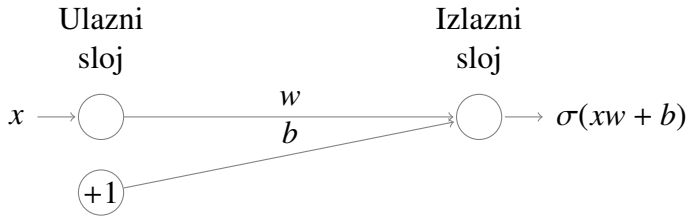
$$\sigma\left(\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \right) = \begin{bmatrix} \sigma(z_1) \\ \sigma(z_2) \\ \vdots \\ \sigma(z_n) \end{bmatrix}.$$

Često ćemo izraz (3.1) pisati i kao

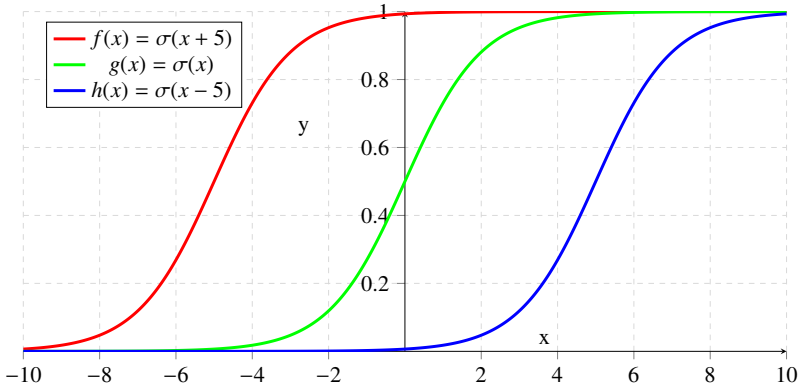
$$a^{(j)} = \sigma\left(z^{(j)} \right), \qquad \text{uz } z^{(j)} = w^{(j)} a^{(j-1)} + b^{(j)}. \tag{3.2}$$

Izlaz neuronske mreže je aktivacija $a^{(L)}$.

$b^{(j)}$ nazivamo *bias*om sloja j, za $1 \leq j \leq L$. Možemo ga grafički predstaviti neuronom unutar kojeg piše +1 što predstavlja da je njegova vrijednost uvijek jednaka 1, a nalazi se u sloju $j-1$ te iz njega izlazi po jedan brid za svaki neuron u sloju j, ali nema ulaznih bridova. Težinu pripadnog brida koji ulazi u i-ti neuron j-tog sloja predstavljamo izrazom $b_i^{(j)}$. *Bias* nam omogućuje pomicanje aktivacijske funkcije po x-osi. To možemo jednostavno predočiti korištenjem mreže sa samo jednim ulazom i izlazom. Za ulaz x, izlaz je $\sigma(wx)$. Ako je σ logistička funkcija, za različite vrijednosti težine w dobivamo logističke funkcije koje sve za ulaz 0 daju 0.5, ali nisu jednako strme što se vidi na slici 3.4. Na ovaj način ne možemo dobiti funkciju koja daje vrijednost u 0 različitu od 0.5. Da bismo to postigli,



Slika 3.10: Nuronska mreža s jednim ulazom, jednim izlazom i *bias*om.



Slika 3.11: Sigmoidalne aktivacijske funkcije ovisno o promjeni vrijednosti b.

moramo imati *bias*. Tada mreža sa samo jednim ulazom, jednim izlazom i *bias*om izgleda kao na slici 3.10.

Vidimo da je izlaz u tom slučaju jednak $\sigma(wx + b)$. Na taj način ćemo aktivacijsku funkciju moći pomjeriti ulijevo ili udesno, što se vidi na slici 3.11, i tako postići veći raspon mogućih izlaznih funkcija [21].

Poglavlje 4

Učenje neuronske mreže

U prošlom poglavlju smo pokazali na koji način neuronska mreža provodi računanje izlaza s trenutno odabranim težinama $\mathbf{w}$. Kako te težine neuronska mreža može sama naučiti, vidjet ćemo na primjeru algoritma *backpropagation*, a za njega moramo prvo proučiti algoritam optimizacije koji se naziva gradijentni spust.

4.1 Gradijentni spust

Težine neuronske mreže podešavamo s ciljem poboljšavanja hipoteze. Hipotezu smatramo boljom što je funkcija troška manja. Funkcija troška L ima najmanju vrijednost u točki w^* ako je $L(w^*) = \min_{\mathbf{w}} L(\mathbf{w})$. To bi značilo da ako pronađemo w^*, pronašli smo i najbolju hipotezu. Postupak traženja minimuma ili maksimuma neke funkcije f nazivamo optimizacijom. Nekad je minimum funkcije moguće naći i analitički, ali u funkcijama s jako puno varijabli, kao što su funkcije troška u neuronskim mrežama, to nije uvijek slučaj. Zato koristimo iterativne metode traženja minimuma funkcije. Jedna od najjednostavnijih takvih metoda je gradijentni spust.

Za diferencijabilne funkcije $f\colon \mathbb{R} \to \mathbb{R}$, $f'(c)$ predstavlja koeficijent smjera tangente na točku $(c, f(c))$. Ako promatramo dvije točke na grafu funkcije f, $(c, f(c))$ i $(d, f(d))$, tada je koeficijent smjera sekante kroz te dvije točke jednak

$$\frac{f(d) - f(c)}{d - c}.$$

Ako $d \to c$, tada se točka $(d, f(d))$ približava točki $(c, f(c))$ i sekanta kroz te dvije točke postaje tangenta, pa se tako i koeficijent smjera sekante postaje koeficijent smjera tangente, to jest,

$$\lim_{d \to c} \frac{f(d) - f(c)}{d - c} = f'(c).$$

Ako je $d = c + \epsilon$ za neki mali ϵ, tada imamo

$$\frac{f(c+\epsilon) - f(c)}{\epsilon} = \frac{f(c+\epsilon) - f(c)}{c + \epsilon - c} = \frac{f(d) - f(c)}{d - c} \approx f'(c),$$

iz čega dobivamo

$$f(c + \epsilon) = f(c) + \epsilon f'(c).$$

Vidimo da možemo smanjiti vrijednost funkcije f ako se iz točke c pomičemo malim koracima ϵ u smjeru negativne derivacije. Slično vrijedi i za funkcije više varijabli, promatramo gradijent funkcije. Neka je $g\colon \mathbb{R}^n \to \mathbb{R}$, tada je gradijent funkcije g jednak $\nabla g(c) = \left(\frac{\partial g}{\partial x_1}(c), \ldots, \frac{\partial g}{\partial x_n}(c)\right)$. Vrijedi da funkcija g iz točke c raste najbrže u smjeru gradijenta $\nabla g(c)$, a pada najbrže u smjeru antigradijenta $(-\nabla g(c))$.

Na toj ideji se temelji gradijentni spust. Minimiziramo funkciju g tako da krenemo od proizvoljne točke x_0 i zatim računamo nove točke prema izrazu

$$x_{k+1} = x_k - \alpha_k \nabla g(x_k), \qquad k = 0, 1, \ldots$$

gdje je vrijednost α_k duljina koraka kojim se pomičemo u smjeru antigradijenta [28]. Često uzimamo fiksnu veličinu α_k koju tada označavamo samo s α. Ovaj postupak ponavljamo do konvergencije. Kriterij konvergencije je dostignut unaprijed zadan broj iteracija ili male promjene u susjednim x_k. Važno je napomenuti da gradijentni spust pronalazi minimum ako je funkcija konveksna jer tada ima samo jedan globalni minimum, u suprotnom može zapeti u nekom od lokalnih minimuma. Također, duljina koraka ne smije biti niti prevelika niti premala [14].

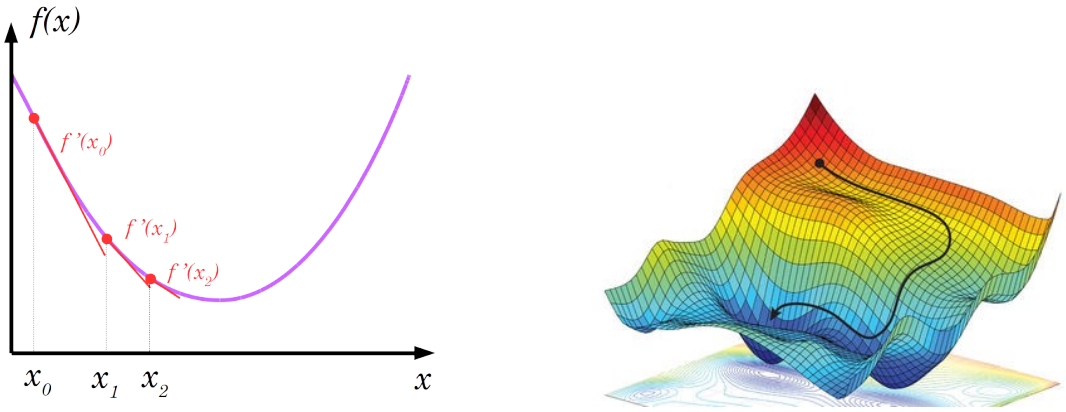
Radi ilustracije, na slici 4.1 prikazano je kako radi metoda gradijentnog spusta na funkciji jedne i više varijabli.

Verzije gradijentnog spusta u strojnom učenju

Kad koristimo gradijentni spust u strojnom učenju, želimo minimizirati funkciju troška koja ovisi o nekim parametrima. Tada ažuriramo vrijednosti parametara pomoću izračunatog gradijenta funkcije troška.

U strojnom učenju za većinu algoritama imamo po nekoliko verzija, ovisno koliki dio skupa za učenje koristimo u jednom koraku. Tako razlikujemo sljedeće verzije gradijentnog spusta [19]

- deterministički ili grupni gradijentni spust (engl. *batch gradient descent*) — za računanje gradijenta u svakom koraku koristimo cijeli skup za učenje. Izračunamo gradijent za svaki primjer, sumiramo ih te na kraju podijelimo brojem primjera i podešavamo parametre pomoću te vrijednosti. Konvergencija prema jedinstvenom globalnom minimumu je zagarantirana ako uzmemo dovoljno malenu duljinu koraka, ali može biti



Slika 4.1: Primjeri gradijentnog spusta na funkciji jedne varijable (lijevo) i funkciji dviju varijabli (desno). Izvor [5, 35]

jako spora jer za svaki korak moramo proći po svim primjerima iz skupa za učenje, a koraka može biti puno [25].

- stohastički gradijentni spust (engl. *stochastic gradient descent*) — u svakom koraku promatramo samo jedan primjer, tj. ažuriramo parametre pomoću gradijenta funkcije troška izračunatom na jednom primjeru. Često je brži od determinističkog gradijentnog spusta, ali uz fiksnu duljinu koraka ne garantira konvergenciju jer može oscilirati oko globalnog minimuma bez da se zaustavi u njemu [25].

- gradijentni spust s mini-grupama (engl. *mini-batch gradient descent*) — nešto između stohastičke i deterministične varijante jer koristimo više od jednog primjera za učenje, ali manje od cijelog skupa za učenje. Skup za učenje podijelimo u nekoliko grupa jednake veličine te u svakom koraku računamo gradijent za sve primjere iz trenutne grupe i ažuriramo parametre. Veličina grupe najčešće je potencija broja 2 od 32 do 256 [19].

4.2 *Backpropagation* algoritam

Gradijentnim spustom lako možemo promijeniti težine u izlaznom sloju, ali se postavlja pitanje kako mijenjati težine u skrivenim slojevima. Tu u priču dolazi algoritam s propagacijom pogreške unatrag (engl. *backpropagation*). Kako ne znamo pogreške u skrivenim slojevima, nego samo znamo onu u izlaznom sloju, tu poznatu propagiramo unatrag prema skrivenim slojevima, odakle i naziv algoritma.

Funkciju troška L promatramo samo na jednom primjeru

$$L(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{s_L}\left(a_k^{(L)} - y_k\right)^2.$$

Zanima nas kako ako malo promijenimo težine mreže, možemo promijeniti funkciju troška. To je upravo parcijalna derivacija $\frac{\partial L}{\partial w_{j,k}^{(l)}}$, gdje l označava sloj, a j i k neurone sloja l, odnosno sloja $l-1$. Kad izračunamo sve parcijalne derivacije, onda mijenjamo težine kao kod gradijentnog spusta.

Krenimo od zadnjeg sloja. Kako bismo izračunali $\frac{\partial L}{\partial w_{j,k}^{(L)}}$ koristimo lančano pravilo za deriviranje i dobivamo sljedeći izraz [27]

$$\frac{\partial L}{\partial w_{j,k}^{(L)}} = \frac{\partial L}{\partial a_j^{(L)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\frac{\partial z_j^{(L)}}{\partial w_{j,k}^{(L)}}, \tag{4.1}$$

gdje za parcijalne derivacije iz (4.1) imamo

$$\begin{aligned}
\frac{\partial L}{\partial a_j^{(L)}} &= \frac{\partial}{\partial a_j^{(L)}}\left(\frac{1}{2}\sum_{k=1}^{s_L}\left(a_k^{(L)} - y_k\right)^2\right) = \left(a_j^{(L)} - y_j\right),\\
\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} &= \frac{\partial}{\partial z_j^{(L)}}\sigma\left(z_j^{(L)}\right) = \sigma'\left(z_j^{(L)}\right),\\
\frac{\partial z_j^{(L)}}{\partial w_{j,k}^{(L)}} &= \frac{\partial}{\partial w_{j,k}^{(L)}}\left(\sum_{i=1}^{s_{L-1}} w_{j,i}^{(L)} a_i^{(L-1)} + b_j\right) = a_k^{(L-1)}.
\end{aligned} \tag{4.2}$$

Nakon uvrštavanja parcijalnih derivacija iz (4.2) u (4.1) dobivamo izraz

$$\frac{\partial L}{\partial w_{j,k}^{(L)}} = \left(a_j^{(L)} - y_j\right)\sigma'\left(z_j^{(L)}\right)a_k^{(L-1)}. \tag{4.3}$$

Slično, za vrijednosti $\frac{\partial L}{\partial b_j}$ dobivamo

$$\frac{\partial L}{\partial b_j^{(L)}} = \frac{\partial L}{\partial a_j^{(L)}}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\frac{\partial z_j^{(L)}}{\partial b_j^{(L)}}, \tag{4.4}$$

gdje jedino različitu vrijednost dobivamo za zadnji član

$$\frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = \frac{\partial}{\partial b_j^{(L)}}\left(\sum_{i=1}^{s_{L-1}} w_{j,i}^{(L)} a_i^{(L-1)} + b_j\right) = 1,$$

pa tako imamo

$$\frac{\partial L}{\partial b_j^{(L)}} = \left(a_j^{(L)} - y_j\right)\sigma'\left(z_j^{(L)}\right). \tag{4.5}$$

Dalje prema [31] izvodimo $\frac{\partial L}{\partial w_{j,k}^{(l)}}$, za $l = 1, \ldots, L-1$. Uvijek promatramo dva susjedna sloja: l i $l+1$. $\frac{\partial L}{\partial w_{j,k}^{(l)}}$ možemo, kao i u (4.1), pomoću lančanog pravila zapisati na sljedeći način

$$\frac{\partial L}{\partial w_{j,k}^{(l)}} = \frac{\partial L}{\partial a_j^{(l)}}\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}\frac{\partial z_j^{(l)}}{\partial w_{j,k}^{(l)}}, \tag{4.6}$$

gdje zadnja dva člana dobijemo istim postupkom kao i u (4.2), to jest, imamo

$$\begin{aligned}\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} &= \frac{\partial}{\partial z_j^{(l)}}\sigma\left(z_j^{(l)}\right) = \sigma'\left(z_j^{(l)}\right),\\ \frac{\partial z_j^{(l)}}{\partial w_{j,k}^{(l)}} &= \frac{\partial}{\partial w_{j,k}^{(l)}}\left(\sum_{i=1}^{s_{l-1}} w_{j,i}^{(l)} a_i^{(l-1)} + b_j\right) = a_k^{(l-1)}.\end{aligned} \tag{4.7}$$

Jedino se mijenja član $\frac{\partial L}{\partial a_j^{(l)}}$. Kada malo promijenimo $a_j^{(l)}$, u idućem sloju se mijenjaju svi $a_i^{(l+1)}$, zbog toga imamo

$$\frac{\partial L}{\partial a_j^{(l)}} = \sum_{i=1}^{s_{l+1}} \frac{\partial L}{\partial a_i^{(l+1)}}\frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}}\frac{\partial z_i^{(l+1)}}{\partial a_i^{(l)}}, \tag{4.8}$$

odnosno nakon što raspišemo nepoznate parcijalne derivacije,

$$\frac{\partial L}{\partial a_j^{(l)}} = \sum_{i=1}^{s_{l+1}} w_{i,j}^{(l+1)}\sigma'\left(z_i^{(l+1)}\right)\frac{\partial L}{\partial a_i^{(l+1)}}, \tag{4.9}$$

gdje koristimo vrijednosti $\frac{\partial L}{\partial a_i^{(l+1)}}$ izračunate prije.

Dakle, kada spojimo jednadžbe iz (4.9) i (4.7) u (4.6) dolazimo do izraza

$$\frac{\partial L}{\partial w_{j,k}^{(l)}} = a_k^{(l-1)}\sigma'\left(z_j^{(l)}\right)\sum_{i=1}^{s_{l+1}} w_{i,j}^{(l+1)}\sigma'\left(z_i^{(l+1)}\right)\frac{\partial L}{\partial a_i^{(l+1)}}. \tag{4.10}$$

Analogno, za parcijalne derivacije za biase imamo

$$\frac{\partial L}{\partial b_j^{(l)}} = \sigma'\left(z_j^{(l)}\right)\sum_{i=1}^{s_{l+1}} w_{i,j}^{(l+1)}\sigma'\left(z_i^{(l+1)}\right)\frac{\partial L}{\partial a_i^{(l+1)}}. \tag{4.11}$$

Ako sada s $\Delta_j^{(l)}$ označimo izraz $\frac{\partial L}{\partial a_j^{(l)}}\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}$, tada izraze za parcijalne derivacije možemo zapisati kompaktnije kao u algoritmu na slici 4.2.

function BACKPROPAGATIONALGORITHM(*examples, network*)
inputs : *examples* = skup primjera za učenje, gdje je ulaz vektor x, a izlaz vektor y,
network = neuronska mreža s propagacijom unaprijed s L slojeva,
parametrima $w_i j$ i b_i te aktivacijskom funkcijom σ
output: neuronska mreža s podešenim parametrima
while *(nije postignut neki kriterij zaustavljanja)* **do**
  foreach $w_{ij} \in$ *network* **do**
    w_{ij} = slučajni mali broj
  end
  foreach *primjer* $(x, y) \in$ *examples* **do**
    `/* Propagacija unaprijed radi izračunavanja izlaza */`
    foreach *čvor* $i \in$ *ulazni sloj* **do**
      $a_i^{(0)} = x_i$
    end
    for $l = 1$ **to** L **do**
      foreach *čvor* $j \in$ *sloj* l **do**
        $z_j^{(l)} = b_j^{(l)} + \sum_{i=1}^{s_{l-1}} w_{i,j}^{(l)} a_i^{(l-1)}$
        $a_j^{(l)} = \sigma\left(z_j^{(l)}\right)$
      end
    end
    `/* Propagacija unazad radi izračunavanja parcijalnih derivacija */`
    foreach *čvor* $j \in$ *izlazni sloj* **do**
      $\Delta_j^{(L)} = \left(a_j^{(L)} - y_j\right) \sigma'\left(z_j^{(L)}\right)$
    end
    for *sloj* $l = L - 1$ ***to*** 1 **do**
      foreach *čvor* $j \in$ *sloj* l **do**
        $\Delta_j^{(l)} = \sigma'\left(z_j^{(l)}\right) \sum_{i=1}^{s_{l+1}} w_{i,j}^{(l+1)} \Delta_i^{(l+1)}$
      end
    end
    `/* Ažuriranje parametara mreže koristeći delte */`
    foreach *težina* $w_{j,k}^{(l)} \in$ *network* **do**
      $w_{j,k}^{(l)} := w_{j,k}^{(l)} - \alpha \Delta_j^{(l)} a_k^{(l-1)}$
    end
    foreach *bias* $b_j^{(l)} \in$ *network* **do**
      $b_j^{(l)} := b_j^{(l)} - \alpha \Delta_j^{(l)}$
    end
  end
end
return *network*

Slika 4.2: Stohastička verzija *backpropagation* algoritma prema [25].

Poglavlje 5

Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (CNN) su poseban tip neuronskih mreža za obradu podataka koji imaju topologiju nalik mreži. Najčešće se kao ulazni podaci koriste slike, koje možemo predstaviti kao 2D mrežu piksela, ili vremenske serije, koj možemo predstaviti kao 1D mrežu s vrijednostima u određenim vremenskim intervalima [19].

5.1 Povijest CNN-a

1950-ih i 1960-ih dva neuropsihologa Hubel i Wiesel pokazali su da vidni korteks mačke sadrži neurone koji pojedinačno reagiraju na male dijelove vidnog polja. Ako uzmemo u obzir da su oči fokusirane na jednu točku, dio vidnog polja unutar kojeg vidni podražaji uzrokuju aktivnost neurona naziva se njegovim područjem osjetljivosti. Susjedni neuorni zaduženi su za slična i preklapajuća područja osjetljivosti [2].

U svom radu iz 1959. godine govore o dva osnovna tipa stanica u primarnom vidnom korteksu, a to su jednostavne i složene stanice. Jednostavne stanice reagiraju na rubove i rešetke u određenom smjeru unutar područja osjetljivosti. Složene stanice također reagiraju na rubove i rešetke određene orijentacije, ali imaju stupanj prostorne invarijancije, što znači da oni unutar svog područja osljetljivosti mogu reagirati na više vizualnih podražaja odjednom. Na primjer, jednostavne stanice mogu reagirati jedino na horizontalne linije na dnu vidnog polja, dok će složene stanice reagirati na horizontalne linije na dnu, sredini ili vrhu vidnog polja.

1962. su objasnili da složene stanice postižu prostornu invarijanciju zbrajajući izlaze nekoliko jednostavnih stanica koje reagiraju na rubove iste orijentacije, ali imaju različita područja osjetljivosti.

Ovaj koncept zbrajanja izlaza jednostavnih stanica je osnova konvolucijskih neuronskih mreža [16].

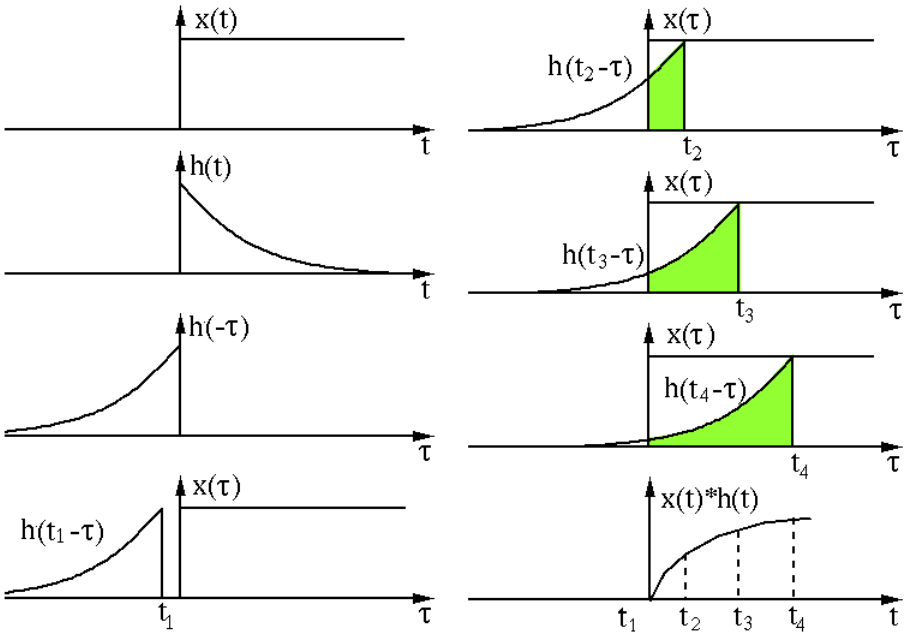
5.2 Motivacija

Zamislimo da želimo napraviti neuronsku mrežu koja će prepoznavati rukom pisanu znamenku 8 sa slika. Kako bismo to mogli napraviti trebamo velik broj slika koje prikazuju razne rukom pisane znamenke. Za takve potrebe postoji MNIST skup podataka koji sadrži 60 tisuća crno-bijelih slika koje prikazuju rukom pisane znamenke dimenzija 28x28. Svaki piksel slike možemo predstaviti brojem od 0 koji predstavlja bijeli piksel do 1 koji predstavlja crni piksel, dok vrijednosti između 0 i 1 predstavljaju postepeno tamnije nijanse sive. Na taj način sliku možemo predati neuronskoj mreži kao polje od $28 \cdot 28$ brojeva. Dodamo skrivene slojeve i ovakvu neuronsku mrežu možemo naučiti algoritmom propagacije pogreške unatrag tako da s velikom točnošću prepoznaje znamenke.

Problem koji se kod ovakvih neuronskih mreža javlja za ovaj primjer je što ako mreži damo sliku na kojoj znamenka nije centrirana, mreža neće dobro prepoznati znamenku. To možemo riješiti tako da provjeravamo manje dijelove slike sve dok ne nađemo centriranu znamenku. Drugi način je da dodamo u skup za učenje još slika na kojima ćemo promijeniti poziciju znamenke pa će mreža biti u stanju naučiti znamenke i kada nisu centrirane, ali tada bismo morali u mreži imati više skrivenih slojeva kako bi mogla naučiti kompliciranije uzorke. Ovakva ideja javila se još krajem 1960-ih i tada dolazi do pojma *dubokih neuronskih mreža*, a to su mreže koje imaju više skrivenih slojeva. U to vrijeme su ovakve mreže imale puno poteškoća s učenjem zbog svoje veličine, ali u današnje vrijeme napredovanjem tehnologije taj problem je uglavnom riješen. Iako možemo mrežu dovoljno povećati kako bismo riješili problem na ovaj način, nema smisla da mreža prepoznaje znamenku 8 na vrhu slike kao nešto različito od znamenke 8 na dnu slike. Želimo napraviti neuronsku mrežu dovoljno pametnom da zna kako je znamenka 8 bilo gdje na slici isti objekt [18].

Rješenje za taj problem daju upravo konvolucijske neuronske mreže koje su se i za ovaj MNIST skup podataka pokazale kao najbolje rješenje te daju grešku od 0.21 % [7].

Jedan od glavnih problema neuronskih mreža s kojima smo se susreli u prijašnjim poglavljima je potpuna povezanost neurona zbog čega za veće dimenzije slika u boji dolazimo do ogromnog broja težina koje trebamo podesiti. Na primjer, ako imamo sliku dimenzija 1000×1000 u boji, tada bismo imali 3 milijuna neurona u ulaznom sloju i ako bismo koristili 1000 neurona u skrivenom sloju, već za ta dva sloja dobili bismo 3 milijarde težina koje trebamo naučiti. S tako puno parametara je teško naći dovoljno podataka za učenje kako bismo izbjegli prenaučenost mreže, a također i učenje mreže koja ima toliko parametara zahtijeva puno memorije i snage računala. Drugi problem kod obrade slika je taj što prilikom pretvaranja dvodimenzionalne slike u jednodimenzionalno polje dolazi do gubitka prostornih informacija. Pikseli koji se nalaze blizu jedni drugima važniji su od piksela koji su na skroz različitim mjestima na slici i pomažu nam definirati značajke slike. Zbog toga želimo iskoristiti prostornu povezanost piksela slike koju ne možemo iskoristiti koristeći dosad promatrane neuronske mreže, ali možemo koristeći konvolucijske neuronske mreže.



Slika 5.1: Primjer računanja konvolucije dviju funkcija. Izvor [34].

5.3 Konvolucija

Jedna jako bitna matematička operacija je konvolucija. S njom se susrećemo u mnogim područjima matematike i inženjerstva kao što su obrada signala i slika.

Definicija 5.3.1. *Konvolucija dvije funkcije $f, g\colon I \to \mathbb{R}$, gdje je I interval u $\mathbb{R}$, je funkcija $f * g$ definirana s*

$$(f * g)(t) = \int_I f(\tau)g(t-\tau)\mathrm{d}\tau. \tag{5.1}$$

Konvolucija je komutativna, asocijativna i distributivna. Sada ćemo prikazati jedan način na koji možemo interpretirati konvoluciju. Imamo dvije realne funkcije realne varijable f i g. Kao u definiciji, funkciju $g(\tau)$ zrcalimo oko y-osi da dobijemo $g(-\tau)$ i zatim ju translatiramo po x-osi za neku vrijednost t te dobivamo $g(t-\tau)$. Kako bismo dobili vrijednost $(f * g)(t)$ za sve t iz domene, translatiranje grafa funkcije g za vrijednost t možemo zamisliti kao da se graf zrcaljene funkcije g giba jednolikom brzinom u smjeru x-osi i u svakom trenutku računamo površinu presjeka grafova $f(\tau)$ i $g(t-\tau)$. Iznos te površine je upravo vrijednost konvolucije u trenutku t.

U kontekstu konvolucijskih neuronskih mreža prvi argument (funkcija f) je *ulaz*, drugi argument (funkcija g) je *jezgra* i često se naziva *filtrom*, dok rezultat (funkciju $f * g$) nazivamo *mapom značajki*.

Ako funkcije f i g nisu kontinuirane nego diskretne, to jest, definirane su na skupu cijelih brojeva $\mathbf{Z}$, tada definiciju možemo iskazati na sljedeći način

$$(f * g)(t) = \sum_{\tau=-\infty}^{+\infty} f(\tau) g(t - \tau) \tag{5.2}$$

i nazivamo ju *diskretnom konvolucijom* [3].

U strojnom učenju kao ulazne podatke često imamo višedimenzionalna polja, ali isto tako možemo imati i jezgru koja je višedimenzionalno polje. Jezgra predstavlja parametre koje želimo podesiti prilikom učenja neuronske mreže. Kako svaki element ulaznog podatka i jezge moramo za vrijeme učenja spremiti odvojeno, obično pretpostavljamo da su funkcije f i g nula svuda, osim na konačnom skupu. Zbog toga u praksi možemo implementirati beskonačnu sumu kao sumu na konačnom broju elemenata višedimenzionalnog polja.

Često koristimo konvolucije preko više osi u isto vrijeme. Na primjer, ako je ulazni podatak dvodimenzionalna slika I, vjerojatno ćemo koristiti i dvodimenzionalni filtar K pa konvolucija glasi

$$S(i, j) = (I * K) = \sum_m \sum_n I(m, n) K(i - m, j - n). \tag{5.3}$$

Kako je konvolucija komutativna, izraz (5.3) možemo zapisati kao

$$S(i, j) = (K * I) = \sum_m \sum_n I(i - m, j - n) K(m, n), \tag{5.4}$$

što je nekad lakše implementirati u bibliotekama strojnog učenja.

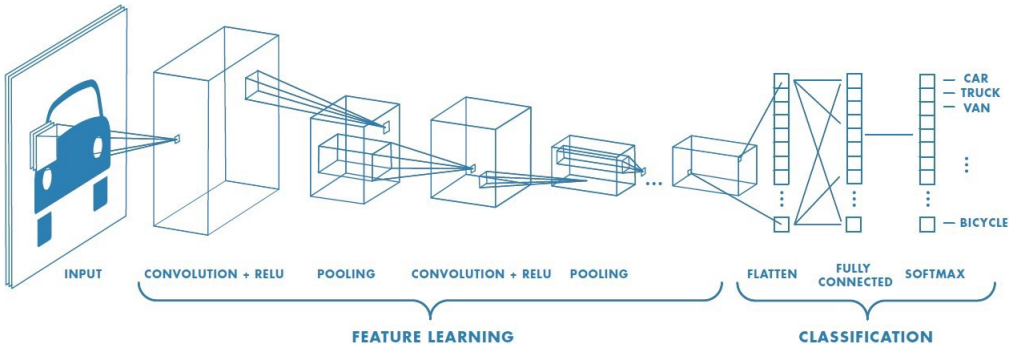
Uz konvoluciju je usko vezana i operacija unakrsna korelacija. Jedina razlika je da nema zrcaljenja jezgre. Zrcaljenje jezgre daje nam svojstvo komutativnosti koje uglavnom nije toliko bitno svojstvo prilikom implementiranja neuronske mreže. Jako puno biblioteka vezanih za neuronske mreže umjesto konvolucije implementiraju unakrsnu korelaciju

$$S(i, j) = (I * K) = \sum_m \sum_n I(i + m, j + n) K(m, n) \tag{5.5}$$

i nju nazivaju konvolucijom [19].

5.4 Arhitektura konvolucijskih neuronskih mreža

Arhitekturu konvolucijske neuronske mreže ćemo objasniti po uzoru na predavanja [24]. Kao i kod običnih neuronskih mreža, tako i u konvolucijskim neuronskim mrežama, osnovne jedinice su neuroni odnosno mape značajki i slojevi. Konvolucijska neuronska



Slika 5.2: Primjer arhitekture konvolucijske neuronske mreže. Izvor [26].

mreža osim ulaznog i izlaznog sloja može imati još tri tipa slojeva, a to su konvolucijski sloj, sloj sažimanja te potpuno povezani sloj. Konvolucijska neuronska mreža započinje ulaznim slojem koji je obično neka dvodimenzionalna ili trodimenzionalna mapa značajki. Zatim se izmjenjuju konvolucijski slojevi i slojevi sažimanja te na kraju dolazi potpuno povezani sloj. Primjer možemo vidjeti na slici 5.2.

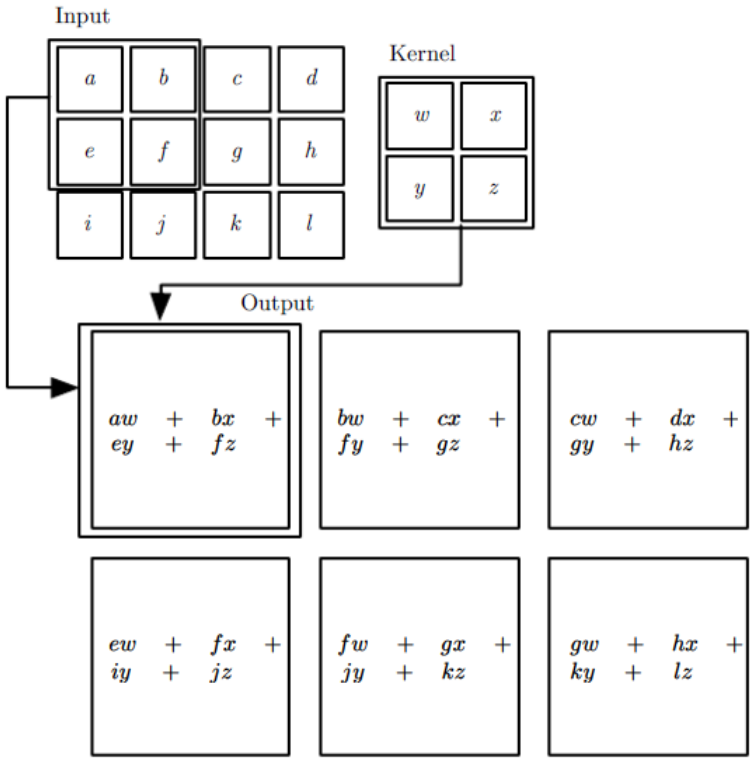
Konvolucijski sloj

Konvolucijski sloj je glavna jedinica konvolucijskih neuronskih mreža i u njemu se odvija većina posla.

Ulaz u konvolucijski sloj predstavlja neki objekt, najčešće slika, u više dimenzija $h \times w \times d$, gdje je h visina, w širina te d dubina. Ako se radi o slici, širina i visina predstavljaju širinu i visinu slike, dok dubina predstavlja broj kanala pa tako ako je slika u boji, dubina je 3, a ukoliko je slika crno-bijela, dubina je 1. Ako je dubina jednaka 1, obično se tada dimenzije pišu u obliku $h \times w$.

Svaki konvolucijski sloj dalje ima određen broj filtara unaprijed fiksirane visine i širine. Oni predstavljaju parametre koje konvolucijska mreža podešava. Filtri su, kao i ulaz, višedimenzionalni, a dubina im uvijek odgovara dubini ulaza, dok su im visina i širina obično puno manje od visine i širine ulaza te su najšeće neparni brojevi.

Matematička operacija konvolucija odvija se nad ulazom i filtrima na način prikazan na slici 5.3. Krenemo od gornjeg lijevog kuta i zamišljamo da filtar kližemo preko ulaza udesno i u svakom koraku računamo zbroj umnožaka na presjeku te to predstavlja jednu vrijednost u izlazu. Kad dođemo do desnog ruba, filtar kližemo natrag na lijevi kraj i prema dolje. Vidimo da ćemo na ovaj način, čim filtar ima širinu ili visinu veću od dva, za izlaz dobiti rezultat manjih dimenzija od ulaza. Kad bismo na ovaj način radili baš operaciju matematičke konvolucije, a ne unakrsne korelacije, morali bismo prvo zrcaliti filtar po stupcima i retcima te primijeniti isti postupak.



Slika 5.3: Prikaz konvolucije bez zrcaljenja (unakrsna korelacija) ulaza dimenzija 4×3 i filtra dimenzija 2×2. Njihovom konvolucijom dobivamo izlaz dimenzija 3×2. Izvor [19].

Kako bismo odredili konvolucijski sloj moramo unaprijed odrediti vrijednosti nekih parametara koje ćemo objasniti u nastavku. Njih nazivamo *hiperparametrima*, a to su širina ruba P, veličina filtra F, broj filtara N te veličina pomaka S.

Nadopunjavanje

Na slici 5.3 vidimo da iz ulaza dimenzija 4×3 dobivamo izlaz dimenzija 3×2. Takvom primjenom konvolucije u svakom idućem konvolucijskom sloju kao izlaz dobivamo slike sve manjih dimenzija. Također, ako pogledamo lijevi gornji element ulaza a sa slike 5.3, vidimo da on u izlaznoj slici sudjeluje samo u stvaranju jednog elementa, za razliku od elementa f koji se javlja u 4 elementa izlazne slike. Na ovaj način dolazi do gubljenja informacija s rubova jer se oni koriste rjeđe od središnjih elemenata. Kako bismo to izbjegli oko ulazne slike (matrice) dodajemo rub širine P koji popunimo nulama i njega nazivamo nadopunjavanje (engl. *padding*).

Širina ruba P može biti proizvoljna, ali najčešće se koriste sljedeće dvije varijante

- Važeće nadopunjavanje (engl. *valid padding*) — ulaz ne nadopunjavamo, nego koristimo izvornu sliku. Tada je $P = 0$.
- Jednako nadopunjavanje (engl. *same padding*) — ulaz nadopunjavamo na način da izlaz ima iste dimenzije kao i ulaz.

Pomak

Pomak (engl. *stride*) S nam govori za koliko piksela ćemo pomaknuti filtar udesno i prema dolje pri računanju konvolucije ulaza i filtra. Pomak veći od 1 uvijek smanjuje dimenzije ulazne mape. Često se pomak gleda kao samo jedna vrijednost S pa se za istu vrijednost pomičemo i udesno i dolje, ali možemo gledati različite pomake S_w i S_h kojima bismo mogli odrediti posebno vrijednost za koju se pomičemo po širini S_w, odnosno po visini S_h.

Filtar

U procesu obrade slika često se javlja izraz filtar. Kroz desetljeća napravljeni su mnogi filtri koji prepoznaju razne stvari. Neki od najpoznatijih su

- filtar srednje vrijednosti čija je vrijednost dana matricom

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}, \tag{5.6}$$

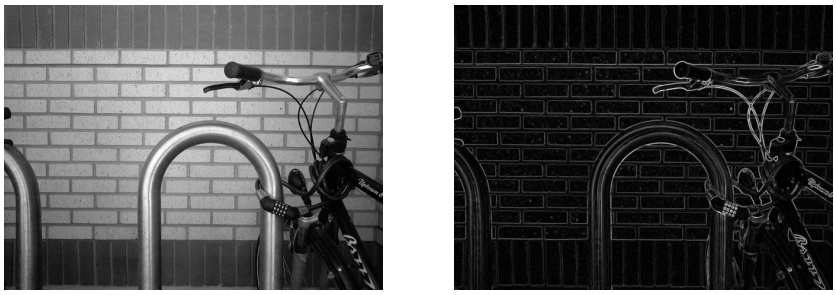
  i čiji je učinak zamućivanje originalne slike jer svaki piksel zamijeni prosjekom susjednih piksela.

- Gaussov filtar sličan je filtru srednje vrijednosti, ali ovaj puta uzimamo težine tako da pikseli oko središnjeg imaju veće težine od rubnih i možemo ga zapisati kao

$$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}. \tag{5.7}$$

- filtri za izoštravanje slike — ima ih više verzija. Jedna od najčešćih je

$$\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix} \tag{5.8}$$



Slika 5.4: Primjer primjene Sobel filtra. Izvor [9].

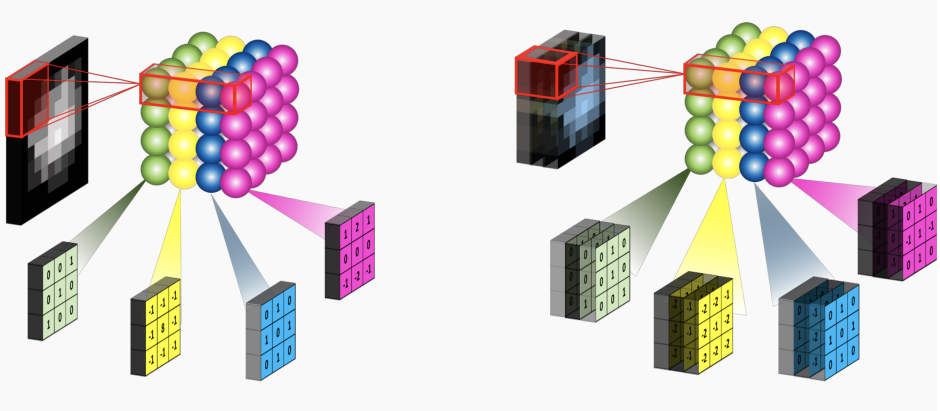
- filtri za detekciju rubova — njih također ima više različitih verzija, a najpoznatiji su Sobel. Koristimo dva filtra (po jedan za svaki smjer), a njihova vrijednost je

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{5.9}$$

  K_x detektira vertikalne rubove, dok K_y detektira horizontalne rubove. Izračunamo konvoluciju između crno-bijele slike i K_x te K_y posebno i dobijemo vrijednosti x i y redom. Tada novu vrijednost piksela dobijemo kao $\sqrt{x^2 + y^2}$. Primjer detekcije rubova vidi se na slici 5.4.

Iz ovog vidimo da se filtri mogu koristiti kako bi odredili određene elemente slike i upravo zato se filtri koriste u prepoznavanju objekata sa slika. Sad se postavlja pitanje kako odrediti koji ćemo filtar koristiti od tolikih mogućnosti. Ako imamo problem nadziranog učenja, tada ne moramo ručno odrediti vrijednosti filtara koje ćemo koristiti i otkrivati koje vrijednosti bi mogle dati najbolje rezultate za naš skup podataka, nego ćemo učiti našu konvolucijsku mrežu da nauči upravo najbolje filtre. Vrijednosti filtara predstavljaju parametre za učenje, kao što su težine kod klasične neuronske mreže. Na primjer, možemo u prvom konvolucijskom sloju imati nekoliko filtara dimenzija 3×3 sa slučajno postavljenim vrijednostima te onda gradijentnim spustom podestiti te vrijednosti.

Kako bismo odredili filtar, moramo odrediti njegovu visinu i širinu te dubinu. Za dubinu vrijedi da mora biti jednaka dubini ulazne mape značajki. Za širinu i visinu obično se uzima jednaka vrijednost te se tada označava s F i naziva veličina filtra, ali kao i za pomak, možemo uzeti različite vrijednosti za visinu i širinu F_h i F_w. Vrijednost F se najčešće (ne nužno) uzima neparna jer tada filtar ima centralni element što nekad može olakšati pričanje o poziciji filtra. Također, ako je veličina filtra neparna, možemo uzimati simetrična nadopunjavanja nulama.



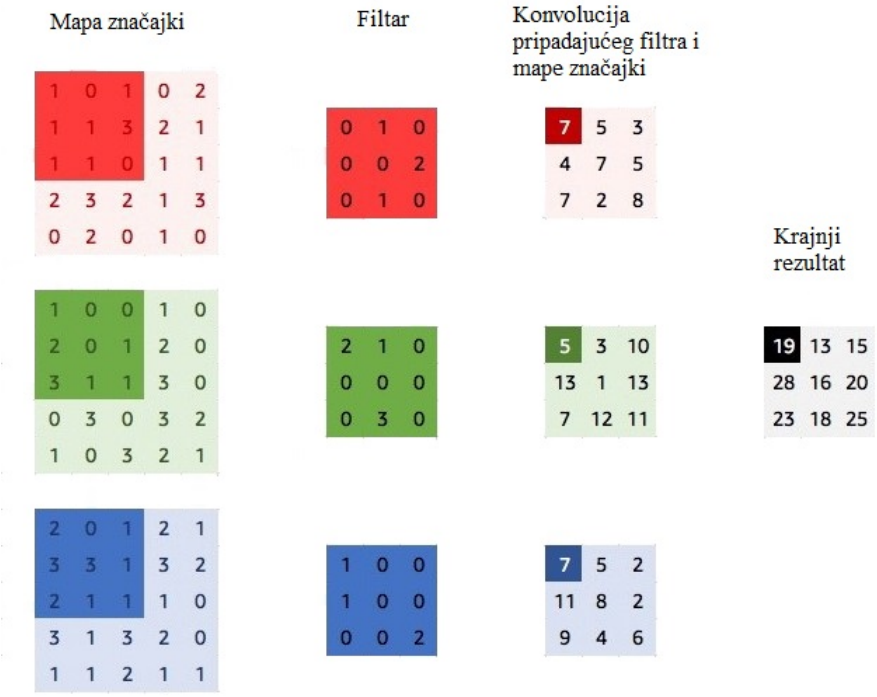
Slika 5.5: *Lijevo:* vidimo ulaznu mapu kao crno-bijelu sliku koju konvoluiramo s četiri dvodimenzionalna filtra. Za svaku konvoluciju s filtrom dobivamo jednu 2D mapu značajki pa ukupno zbog četiri filtra dobivamo mapu značajki s dubinom četiri. *Desno:* vidimo ulaznu mapu značajki kao sliku u boji, odnosno 3D mapu značajki. Nju konvoluiramo također s četiri, ovaj put trodimenzionalna filtra jer je i dubina ulaza tri. Ponovno za konvoluciju sa svakim 3D filtrom dobivamo po jednu 2D mapu značajki, što na kraju predstavljamo kao jednu izlaznu mapu značajki dubine četiri. Izvor [32].

Osim dimenzija filtra, određujemo još jedan parametar, a to je broj filtara. Broj filtara određuje dubinu izlazne mape značajki konvolucijskog sloja. Konvolucijom 3D mape značajki s 3D filtrom, dobivamo 2D izlaznu mapu značajki. Zato dubinu izlazne mape značajki povećavamo brojem filtara koje koristimo što vidimo na slici 5.5.

Uloga filtra je detektirati neke značajke s ulazne mape značajki. Tako primjenom više filtara konvolucijska neuronska mreža može detektirati različite značajke koje će zatim proslijediti idućem sloju u kojem će se iz otkrivenih značajki detektirati još neke nove i tako dalje. Zato konvolucijska neuronska mreža obično u prvim slojevima prepoznaje neke općenitije značajke poput bridova, dok u kasnijim slojevima može prepoznati značajke poput ljudskog lica. Iz tog razloga, također, obično broj filtara raste s idućim slojevima kako bi se mogli otkriti kompleksniji oblici.

Pokažimo na primjeru kako se računa konvolucija ulaza i filtra kada imaju dubinu veću od jedan. Pretpostavimo da imamo na ulazu sliku u boji. Tada je dubina tri te dubina filtra također mora biti jednaka tri. Na slici 5.6 vidimo ulaznu mapu značajki dimenzija $5\times5\times3$ te filtar dimenzija $3\times3\times3$ i oni konvolucijom daju izlaznu mapu značajki dimenzija $3\times3\times1$.

Sada zamislimo da odvojimo slojeve slike po boji — crvena, zelena i plava boja predstavljaju po jednu dvodimenzionalnu mapu značajki. Na isti način razdvojimo i filtar po dubini te tada konvoluiramo po jednu dvodimenzionalnu mapu značajki s pripadnim dvodimenzionalnim filtrom kao na slici 5.3. Za svaku od pripadnih dvodimenzionalnih konvolu-



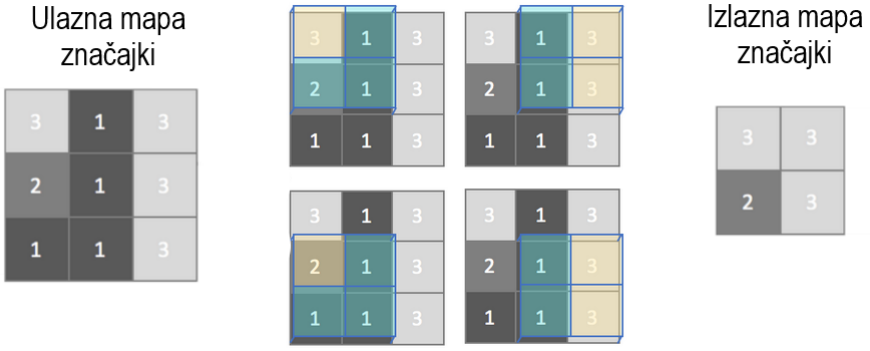
Slika 5.6: Primjer računanja konvolucije mape značajki dimenzija 5×5×3 i filtra dimenzija $3 \times 3 \times 3$. Krajnji rezultat predstavlja sumu matrica u prethodnom stupcu. Izvor [22].

cija dobivamo po jedan realni broj, zatim sve brojeve zbrojimo i dobivamo prvu vrijednost izlazne mape značajki.

Sloj sažimanja

Konvolucijski slojevi primjenom raznih naučenih filtara stvaraju mape značajki koje izdvajaju različite značajke s ulazne slike. Problem je u tome što se te značajke izdvoje na točno određenom mjestu na kojem se nalaze na slici. Zbog toga ćemo prilikom malih promjena pozicije značajki na slici (kao što su rotacija i translacija) dobiti novu značajku. Cilj je povećati prostornu invarijantnost, a to postižemo tako da napravimo verziju mape značajki niže kvalitete kako bismo izdvojili bitne strukturalne elemente od detalja koji možda nisu bitni za naš zadatak [12]. Za to nam služe slojevi sažimanja (engl. *pooling layer*).

Osim povećanja prostorne invarijantnosti, slojevi sažimanja koriste se i radi smanjivanja dimenzija mape značajki s tim da se smanjuju samo visina i širina dok dubina ostaje ista. Na taj način smanjujemo broj parametara koje mreža mora naučiti.



Slika 5.7: Sažimanje maksimalnom vrijednošću. Izlazna mapa značajki predstavlja vrijednost nakon odabira maksimalnih vrijednosti. Maksimalne vrijednosti u svakom 2×2 okviru označene su žutom bojom. Vrijednosti parametara su $F = 2$ i $S = 1$. Izvor [20].

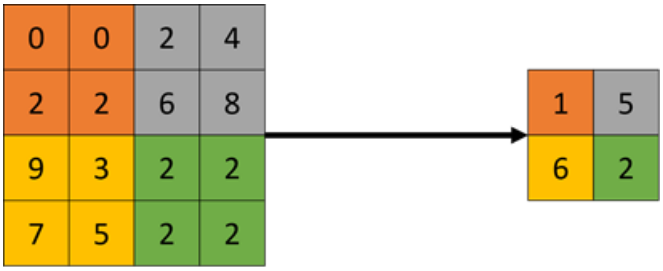
Kao kod konvolucijskih slojeva, definiramo veličinu filtra F te pomak S, nadopunjenje nulama obično se ne koristi kod sažimanja. Postupak je sličan kao kod računanja konvolucije — na slici promatramo $F \times F$ podataka i predstavljamo ih jednom vrijednošću pomoću neke funkcije sažimanja (engl. *pooling function*), zatim se pomaknemo S koraka pa ponovimo postupak. Razlika je što u sloju sažimanja nemamo matricu parametara koje želimo naučiti, samo su nam bitna veličina filtra i pomak te funkcija sažimanja.

Postoji nekoliko tipova slojeva sažimanja ovisno o funkciji sažimanja, a to su

- sažimanje maksimalnom vrijednošću (engl. *max-pooling*) gdje je funkcija sažimanja maksimum po svim elementima unutar $F \times F$ okvira. Primjer vidimo na slici 5.7. Ovaj način se u zadnje vrijeme najčešće koristi [24].
- Sažimanje srednjom vrijednošću gdje unutar svakog $F \times F$ okvira računamo srednju vrijednost svih elemenata. Primjer na slici 5.8.
- Sažimanje L_2 normom gdje $F \times F$ okvir predstavljamo vrijednošću koju dobijemo tako da uzmemo sumu kvadrata svih elemenata unutar tog okvira.
- Sažimanje težinskim usrednjavanjem gdje težine opadaju s udaljenošću od centralnog elementa.

Potpuno povezani sloj

Potpuno povezani sloj (engl. *fully connected layer*) možemo predstaviti klasičnom potpuno povezanom neuronskom mrežom koja može imati proizvoljan broj skrivenih slojeva. Ulaz



Slika 5.8: Sažimanje srednjom vrijednošću s parametrima $F = 2$ i $S = 2$. Ovo je najčešći odabir parametara za slojeve sažimanja; svaka dimenzija se smanjuje za faktor dva pa se ukupna dimenzija smanjuje na jednu četvrtinu prijašnje. Izvor [1].

u potpuno povezani sloj je jednodimenzionalno polje dobiveno od višedimenzionalne mape značajki iz prethodnog sloja (engl. *flattening*), najčešće iz sloja sažimanja. Izlaz iz potpuno povezanog sloja ujedno predstavlja i izlaz cijele mreže. Podsjetimo, potpuno povezana neuronska mreža za svaki sloj $1 \leq l \leq L$ ima matricu težina dimenzija $s_l \times s_{l-1}$, gdje je s_j broj neurona u sloju j.To znači da kod potpuno povezanog sloja imamo puno parametara za naučiti. Zbog toga se potpuno povezani slojevi najčešće koriste na samom kraju kada se dimenzija mape značajki smanji primjenom slojeva sažimanja.

5.5 Propagacija unaprijed

Neka je l konvolucijski sloj ili sloj sažimanja u konvolucijskoj neuronskoj mreži. Tada sloju l pridružujemo sljedeće vrijednosti

$$\begin{aligned} F^{[l]} &= \text{veličina filtra u sloju } l, \\ P^{[l]} &= \text{širina ruba u sloju } l, \\ S^{[l]} &= \text{pomak u sloju } l, \\ N^{[l]} &= \text{broj filtara u sloju } l. \end{aligned} \tag{5.10}$$

Neka je ulazna mapa značajki u sloj l dimenzija $H^{[l-1]} \times W^{[l-1]} \times D^{[l-1]}$.

Tada je izlazna mapa značajki iz sloja l dimenzija $H^{[l]} \times W^{[l]} \times D^{[l]}$ gdje za vrijednosti $H^{[l]}$, $W^{[l]}$ i $D^{[l]}$ vrijedi

$$
\begin{aligned}
H^{[l]} &= \left\lfloor \frac{H^{[l-1]} + 2P^{[l]} - F^{[l]}}{S^{[l]}} + 1 \right\rfloor, \\
W^{[l]} &= \left\lfloor \frac{W^{[l-1]} + 2P^{[l]} - F^{[l]}}{S^{[l]}} + 1 \right\rfloor, \\
D^{[l]} &= N^{[l]}.
\end{aligned}
\tag{5.11}
$$

Kao i kod klasičnih neuronskih mreža, tako i ovdje u svakom sloju imamo aktivaciju $A^{[l]}$ koja predstavlja izlaz sloja. Aktivaciju u konvolucijskom sloju dobijemo na sljedeći način

- prvo napravimo konvolciju ulazne mape značajki, odnosno izlazne mape značajki iz sloja $l-1$, a to je $A^{[l-1]}$ s filtrom $K^{[l]}$ iz sloja l i dobivamo $\left(A^{[l-1]} * K^{[l]}\right)$,
- zatim na dobivenu mapu značajki dodamo *bias* $b^{[l]}$ što je vektor dimenzije $N^{[l]}$. Za svaki filtar imamo po jednu realnu vrijednost — prilikom konvolucije i-te mape značajki s i-tim filtrom dobivamo dvodimenzionalnu značajku sa širinom $W^{[l]}$ i visinom $H^{[l]}$ koju predstavljamo matricom te na svaki element te matrice pridodajemo vrijednost $b_i^{[l]}$.
- Na kraju na svaki element mape značajki primijenimo aktivacijsku funkciju σ da dobijemo konačni izlaz sloja l.

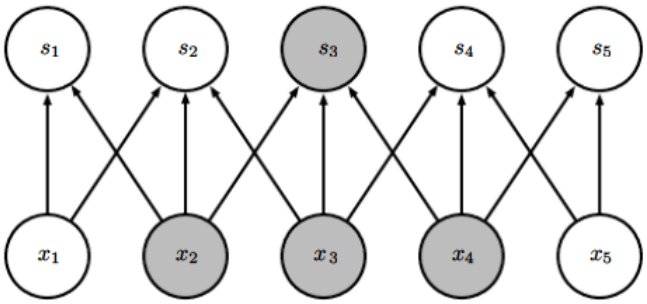
U sloju sažimanja nemamo *bias*e, niti aktivacijsku funkciju. Izlaz je samo računanje konvolucije ulazne mape značajki s fiksiranim filtrom. Zato sloj sažimanja nema parametre.

Aktivacija u potpuno povezanom sloju identična je kao i kod klasičnih neuronskih mreža.

5.6 Prednosti konvolucijskih mreža nad klasičnim neuronskim mrežama

Prema [19] konvolucije uvode tri bitne ideje koje unaprjeđuju sustav za učenje. To su raspršena povezanost, dijeljenje parametara i ekvivarijantnost reprezentacije na translacije.

Kod konvolucijskih neuronskih mreža konvolucijski sloj i sloj sažimanja primaju kao ulaz te daju kao izlaz neku trodimenzionalnu mapu značajki dimenzija $h \times w \times d$. Tada taj objekt možemo predstaviti pomoću d matrica dimenzija $h \times w$. Možemo zamisliti da tada sloj koji kao izlaz daje taj objekt ima $h \cdot w \cdot d$ neurona i svaki od njih predstavlja po jedan element tih matrica te oni predstavljaju ulaz u idući sloj. Na te elemente djelujemo



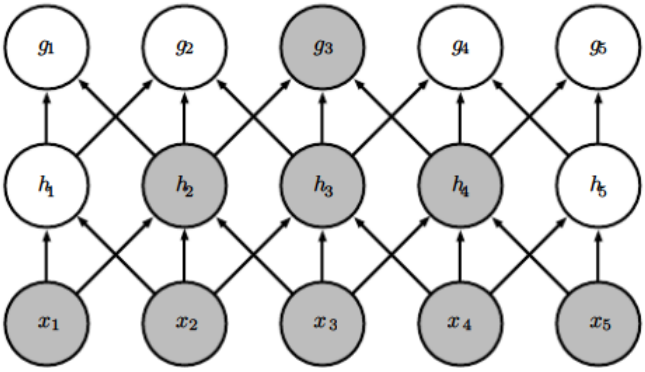
Slika 5.9: Raspršena povezanost — ako promatramo jedinicu s_3, na nju utječu samo x_2, x_3 i x_4 kada je izlaz konvolucija s filtrom veličine tri. Izvor [19].

filtrom kako bismo dobili novu mapu značajki, dok elementi filtra predstavljaju parametre koje mreža uči.

U klasičnim neuronskim mrežama, izlaz sloja računamo tako da množimo aktivaciju prošlog sloja s matricom težina gdje svaka težina predstavlja posebnu vezu između neurona prethodnog i trenutnog sloja. Svaki neuron je povezan sa svakim i za njih kažemo da imaju gustu povezanost (engl. *dense*). U konvolucijskim neuronskim mrežama to nije slučaj — imamo rijetku, odnosno raspršenu povezanost što se vidi na slici 5.9, izlazi su povezani samo s nekolicinom ulaza. Svojstvo raspršenosti postižemo odabirom filtara određene veličine koji su puno manji od ulaza.

Na primjer, čak i ako ulazna slika ima na tisuće piksela, možemo prepoznati rubove ili neke druge bitne značajke, pomoću filtara koji zauzimaju samo desetke piksela. Zbog toga neovisno o veličini ulazne mape značajki imamo određen broj parametara za učenje što bitno utječe na efikasnost. Parametri zauzimaju manje memorije i trebamo manje operacija kako bismo izračunali izlaz. Ako imamo m ulaznih neurona i n izlaznih neurona, onda u klasičnim neuronskim mrežama imamo $n \times m$ parametara i algoritam zahtijeva $O(n \times m)$ operacija po primjeru, dok u konvolucijskom sloju možemo ograničiti broj parametara ovisno o veličini filtra k pa smanjujemo broj operacija na $O(n \times k)$, gdje je k fiksan i uglavnom puno manji od m. Iako su jedinice između susjednih slojeva raspršeno povezane, odnosno direktne veze su rijetke, jedinice u dubokim slojevima mogu indirektno biti povezane sa svim ili velikim dijelom ulazne slike kao na slici 5.10. To omogućuje mreži da gradi kompliciranije interakcije između objekata, ali pomoću blokova koji su raspršeno povezani.

Svojstvo dijeljenja parametara odnosi se na filtar. U klasičnim neuronskim mrežama prilikom računanja izlaza sloja svaki element matrice težina koristi se samo jednom. U konvolucijskim neuronskim mrežama jedan filtar se koristi više puta kako se pomičemo po slici prilikom konvolucije i na taj način umjesto da učimo skup parametara za svaku lokaciju u ulaznoj slici, učimo samo jedan skup parametara. Dijeljenjem parametara dobivamo



Slika 5.10: Raspršena povezanost u dubljim slojevima. Izvor [19].

još manji model nego ako koristimo samo raspršenu povezanost bez dijeljenja parametara.

Forma dijeljenja parametara koja se koristi u konvolucijskim neuronskim mrežama uzrokuje da je svaki sloj ekvivarijantan s obzirom na translacije.

Kažemo da je funkcija f ekvivarijantna s obzirom na funkciju g ako $f(g(x)) = g(f(x))$. Drugim riječima, funkcija je ekvivarijantna ako se s promjenom ulaza, izlaz mijenja na isti način.

U slučaju konvolucije, ako je g bilo koja funkcija koja translatira ulaz, tada je konvolucija ekvivarijantna s obzirom na funkciju g.

Konvolucija nije ekvivarijantna na neke druge transformacije ulaza kao što su skaliranje ili rotacija.

Konvolucijske neuronske mreže u praksi

U matematičkoj teoriji umjetne inteligencije postoji teorem koji se zove teorem univerzalne aproksimacije. On kaže kako neuronska mreža s propagacijom unaprijed koja sadrži samo jedan skriveni sloj s konačnim brojem neurona može aproksimirati proizvoljnu neprekidnu funkciju na kompaktnim podskupovima od $\mathbb{R}^n$, uz blage pretpostavke na aktivacijsku funkciju. To znači da neuronska mreža može prikazati velik broj zanimljivih funkcija kada im se daju odgovarajući parametri.

S ovim teoremom više se ne pitamo može li se neka neprekidna funkcija izračunati neuronskom mrežom, nego koji je dobar način za njezino računanje. Drugo pitanje koje se postavlja je zašto bismo koristili duboke neuronske mreže s desecima i više slojeva ako se funkcija može izračunati i sa samo jednim. Iako je u teoriji to moguće, u praksi postoji puno razloga za korištenje dubokih neuronskih mreža.

Iako svaka arhitektura mreže može uz dovoljno vremena i resursa riješiti većinu problema, neke arhitekture to mogu u puno kraćem vremenu te je bitno odabrati dobru arhi-

tekturu mreže.

Prva arhitektura konvolucijskih neuronskih mreža naziva se LeNet-5, a razvio ju je Yann LeCun za prepoznavanje rukom pisanih znamenaka 1998. godine. Ona ima slojeve slagane redom jedan konvolucijski pa jedan sloj sažimanja i tako nekoliko puta te na kraju potpuno povezani sloj. LeNet-5 predstavlja standardni predložak konvolucijskih neuronskih mreža.

U novije vrijeme se u praksi slojevi ne slažu uvijek kao jedan konvolucijski sloj pa jedan sloj sažimanja, nego postoje razne varijacije. Na primjer, koristi se prvo više konvolucijskih slojeva pa tek onda jedan sloj sažimanja. Kako slojevi sažimanja smanjuju dimenziju mape značajki, korištenjem više konvolucijskih slojeva zaredom možemo izvući više reprezentacija podataka prije nego što izgubimo dio prostornih informacija sažimanjem. U velikom broju modernih konvolucijskih neuronskih mreža, kao što su VGG, Inception i ResNet, broj uzastopnih konvolucijskih slojeva može dostići velike brojeve.

Druga mogućnost je slaganje konvolucijskih slojeva u paraleli što se koristi u arhitekturi Inception.

Osim odabira slaganja i broja slojeva također možemo birati između raznih aktivacijskih funkcija, ali kao najbolja se u praksi pokazuje ReLU jer u dubokim neuronskim mrežama može lako doći do problema nestajanja gradijenta, što uz ovu funkciju nije slučaj.

Glavni cilj prilikom učenja neuronske mreže je postići generalizaciju. Ako naša mreža ne generalizira dobro problem može biti

- podnaučenost (engl. *underfitting*) — mreža nije dovoljno složena kako bi točno uhvatila složenost svojstvenu problemu koji pokušavamo riješiti; možemo reći da je mreža prejednostavna, ne postoji skup parametara koji bi mogao objasniti skup za učenje,

- prenaučenost (engl. *overfitting*) — mreža ima puno parametara i zbog dugog učenja dolazi do toga da počne odgovarati specifičnim primjerima iz skupa za učenje čime više ne odgovara generalnim uzorcima; možemo reći da je mreža prekompleksna.

Kako duboke neuronske mreže imaju jako puno parametara za podešavanje, lako može doći do problema prenaučenosti. Jedan od načina na koji rješavamo taj problem je povećanje skupa za učenje. Duboke konvolucijske neuronske mreže daju bolje rezultate ako su učene na velikom skupu podataka jer je tada manja mogućnost da će mreža naučiti sve specifičnosti skupa za učenje.

Drugi način je isključivanje neurona (engl. *dropout*). Neuroni se prilikom učenja, u potpuno povezanom sloju, isključuju iz mreže s nekom vjerojatnošću p. To radimo na način da aktivaciju neurona postavimo na 0. Isključivanjem određenih neurona odbacujemo informacije koje su dosad naučene za taj neuron pa mreža ne može točno naučiti sve specifičnosti skupa za učenje i time se smanjuje prenaučenost mreže, ali i vrijeme učenja.

Iz ovih nekoliko primjera vididmo da na puno načina možemo utjecati na to koliko će naša konvolucijska neuronska mreža biti dobra. Stoga, i zbog prednosti nad klasičnim neuronskim mrežama, ne iznenađuje činjenica da se zadnjih godina konvolucijske mreže sve više istražuju te se razvijaju modeli s novim arhitekturama i novim idejama, a s razvojem novih arhitektura dolazi do sve veće primjene konvolucijskih mreža. Zbog toga se u današnje vrijeme s njima susrećemo u velikom broju problema poput raspoznavanja lica, klasificiranja slika, analiziranja dokumenata, prepoznavanja govora i mnogim drugim.

Bibliografija

[1] *Average-Pooling*, https://embarc.org/embarc_mli/doc/build/html/MLI_kernels/pooling_avg.html, posjećena 5. 11. 2019.

[2] *Convolutional Neural Network*, https://en.wikipedia.org/wiki/Convolutional_neural_network, posjećena 22. 10. 2019.

[3] *Convolution*, https://en.wikipedia.org/wiki/Convolution, posjećena 26. 10. 2019.

[4] *Genetic algorithm*, https://en.wikipedia.org/wiki/Genetic_algorithm, posjećena 15. 9. 2019.

[5] *Gradient descent*, https://www.neural-networks.io/en/single-layer/gradient-descent.php, posjećena 30. 8. 2019.

[6] *The History of Artificial Intelligence*, https://courses.cs.washington.edu/courses/csep590/06au/projects/history-ai.pdf, posjećena 29. 7. 2019.

[7] *MNIST database*, https://en.wikipedia.org/wiki/MNIST_database, posjećena 26. 10. 2019.

[8] *Rectifier (neural networks)*, https://en.wikipedia.org/wiki/Rectifier_(neural_networks), posjećena 2. 9. 2019.

[9] *Sobel Operator*, https://en.wikipedia.org/wiki/Sobel_operator, posjećena 2. 11. 2019.

[10] B. H. Arabi, *Solving NP-complete Problems Using Genetic Algorithms*, (2016), 43–48.

[11] M. Bošnjak, *Neuronske mreže*, 2011.

[12] J. Brownlee, *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*, https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/, posjećena 4. 11. 2019.

[13] J. Brownlee, *A Gentle Introduction to the Rectified Linear Unit (ReLU)*, https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/, posjećena 2. 9. 2019.

[14] B. Dalbelo Bašić i J. Šnajder, *Linearni diskriminativni modeli*, https://www.fer.unizg.hr/_download/repository/SU-8-LinearniDiskriminativniModeli.pdf, posjećena 4. 10. 2019.

[15] B. Dalbelo Bašić, M. Čupić i J. Šnajder, *Umjetne neuronske mreže*, 2008.

[16] R. Draelos, *A Short History of Convolutional Neural Networks*, https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/, posjećena 22. 10. 2019.

[17] K. D. Foote, *A Brief History of Machine Learning*, https://www.dataversity.net/a-brief-history-of-machine-learning/, posjećena 29. 7. 2019.

[18] A. Geitgey, *Deep Learning and Convolutional Neural Networks*, https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721#, posjećena 28. 10. 2019.

[19] I. Goodfellow, Y. Bengio i A. Courville, *Deep Learning*, The MIT Press, 2016.

[20] J. Jordan, *Convolutional neural networks*, https://www.jeremyjordan.me/convolutional-neural-networks/, posjećena 4. 11. 2019.

[21] N. Kohl, *Role of Bias in Neural Networks*, https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks, posjećena 17. 10. 2019.

[22] T. Lane, *Multi-Channel Convolutions explained with... MS Excel!*, https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108, posjećena 2. 11. 2019.

[23] T. Levanić, *Neuronske mreže*, https://web.math.pmf.unizg.hr/nastava/matsoft/DobreDZ/2015-16/HTML/TomislavLevanic, posjećena 8. 8. 2019.

[24] A. Ng, *Convolutional Neural Networks*, https://www.coursera.org/learn/convolutional-neural-networks, posjećena 28. 10. 2019.

[25] S. Russell i P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Upper Saddle River, NJ, 2010.

[26] S. Saha, *A Comprehensive Guide to Convolutional Neural Networks*, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, posjećena 5. 11. 2019.

[27] G. Sanderson, *Backpropagation calculus*, https://youtu.be/tIeHLnjs5U8, posjećena 20. 9. 2019.

[28] R. Scitovski, N. Truhar i Z. Tomljanović, *Metode optimizacije*, https://www.mathos.unios.hr/mo/Metode_optimizacije_2014.pdf.

[29] S. Shalev-Shwartz i S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, New York, NY, USA, 2014.

[30] S. Sharma, *Applications of Genetic Algorithm in Software Engineering, Distributed Computing and Machine Learning*, International Journal of Computer Applications and Information Technology **9** (2017), 208–212.

[31] D Stansbury, *Derivation: Error Backpropagation and Gradient Descent for Neural Networks*, https://theclevermachine.wordpress.com/tag/backprop-derivation//, posjećena 23. 9. 2019.

[32] M. Stewart, *Simple Introduction to Convolutional Neural Networks*, https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac, posjećena 2. 11. 2019.

[33] V. N. Vapnik, *Statistical Learning Theory*, Wiley-Interscience, 1998.

[34] R. Wang, *Digital Convolution – E186 Handout*, http://fourier.eng.hmc.edu/e161/lectures/convolution/index.html, posjećena 26. 10. 2019.

[35] C. M. Wild, *It's Only Natural: An Excessively Deep Dive Into Natural Gradient Optimization*, https://towardsdatascience.com/its-only-natural-an-excessively-deep-dive-into-natural-gradient-optimization-75d464b89dbb, posjećena 5. 11. 2019.

Sažetak

Ovaj diplomski rad bavi se klasičnim i konvolucijskim neuronskim mrežama. Cilj rada bio je opisati i objasniti način na koji se pokušava simulirati rad ljudskih neuronskih mreža na računalne sustave. Sve to s ciljem stvaranja programa koji mogu imitirati ljudsko razmišljanje i djelovanje.

Iz tog razloga smo detaljno opisali klasične umjetne neuronske mreže s propagacijom unaprijed pri čemu smo posebnu pozornost posvetili *backpropagation* algoritmu. U zadnjem poglavlju prikazujemo konvolucijske neuronske mreže, njihovu građu pomoću konvolucijskih slojeva, slojeva sažimanja i potpuno povezanih slojeva te hiperparametara mreže. Dalje govorimo o prednostima konvolucijskih nad klasičnim neuronskim mrežama.

Na samom kraju rada spominjemo neke od poznatijih konvolucijskih neuronskih mreža te govorimo o problemu prenaučenosti koji se često javlja u dubokim neuronskim mrežama.

Summary

This thesis addresses feed-forward and convolutional neural networks. The task of the thesis was to explain and describe the way human neural networks are simulated onto computer systems for the purpose of creating computer programs that are able to imitate human behavior.

We provided a detailed description of feed-forward neural networks with emphasis on backpropagation algorithm. The last chapter is focused on presenting convolutional neural networks, their architecture consisting of convolutional layers, pooling layers, fully connected layers and hyperparameters in the network. Furthermore, we discuss advantages of convolutional neural networks over feed-forward neural networks.

At the very end of the thesis we mention some of the more famous convolutional neural networks and we're discussing the problem of overfitting, which often occurs in deep neural networks.

Životopis

Rođena sam 20. lipnja 1995. godine u Vinkovcima. Živim u Privlaci u blizini Vinkovaca i tu pohađam osnovnu školu. 2010. godine upisujem srednju školu — Gimnaziju Matije Antuna Reljkovića, smjer prirodoslovno-matematički. Četiri godine nakon upisujem preddiplomski studij Matematika, smjer inženjerski na Prirodoslovno-matematičkom fakultetu koji završavam u roku. Iste godine upisujem diplomski studij, smjer Računarstvo i matematika na kojem dobivam nagradu za najbolju studenticu smjera.