Primjena neuronskih mreža u obradi slika

Levanić, Tomislav

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Tomislav Levanić

PRIMJENA NEURONSKIH MREŽA U OBRADI SLIKA

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, veljača 2018.

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. _____

Mojoj obitelji

Sadržaj

Uvod

Zavidni rezultati neuronskih mreža u brojnim zadacima umjetne inteligencije pokazuju kako je to područje s izrazito velikim potencijalom. Neovisno o tome radi li se o obradi i analizi zvuka, slika, video snimki, teksta ili kojeg drugog ulaznog podatka, neuronske mreže sve značajnije prednjače kao najbolji odabir algoritma za učenje. Ključne u postizanju nadljudskih rezultata u obradi slika su konvolucijske neuronske mreže i one predstavljaju glavni dio ovoga rada.

U prvom dijelu rada dajemo definiciju takozvanih acikličkih neuronskih mreža te svih pojmova potrebnih za njihovu definiciju. Time postavljamo teorijski temelj za definicije neuronskih slojeva potrebnih za definiciju konvolucijske neuronske mreže, koju odmah potom i definiramo.

Drugi dio rada bavi se primjenom konvolucijske neuronske mreže u klasifikaciji slika. Osim toga, dio rada posvećujemo i raznim problemima u kojima su se konvolucijske neuronske mreže pokazale kao izvrsno rješenje.

Kratka povijest neuronskih mreža

Iako su pravi procvat doživjele tek početkom 21. stoljeća, ideja o računalu baziranom na principu rada ljudskoga uma pojavila se još 40-tih godina 20. stoljeća.

Povijest neuronskih mreža možemo podijeliti na dva razdoblja. Prvo je razdoblje od 1940-ih pa sve do 1970-ih godina, kada je područje umjetnih neuronskih mreža zatihnulo. Drugo razdoblje započelo je 1980-ih godina i proteže se sve do danas. U nastavku dajemo kratak opis oba razdoblja. Nešto detaljnija povijest može se pronaći u [34].

Rođenje neuronskih mreža

Kao početak ovog razdoblja često se uzima članak [21] autora McCullocha i Pittsa objavljen 1943. godine, u kojem je pokazano da čak i najjednostavnije neuronske mreže mogu, u principu, izračunati proizvoljnu aritmetičku ili logičku funkciju.

Prvo uspješno neuro-računalo (`Mark I perceptron`) razvijeno je tijekom 1957. i 1958., a u njegovom razvoju sudjelovali su Frank Rosenblatt, Charles Wightman i drugi. Nešto kasnije, Bernard Widrow, u suradnji sa svojim studentima, razvija modele `ADALINE` i `MADALINE`. `MADALINE` je prva mreža korištena za praktičan problem, a koristila se u sustavu kontrole leta gotovo 50 godina.

Brojni uspjesi u ovom području postavili su izrazito visoka očekivanja za neuronske mreže pa su se pojavila nerijetka predviđanja da će se već kroz nekoliko godina pojaviti računala sposobna čitati, pisati i pričati. Baš takva visoka očekivanja oslabila su matematičku strogoću te je pristup cijelom području bio većinom eksperimentalan, što je s vremenom postao problem. Nedostatak dobrih ideja ovu je granu doveo u teško stanje. Kampanja u svrhu diskreditiranja neuronskih mreža, koju su vodili Marvin Minsky i Seymour Papert, postigla je svoj cilj preusmjeravanja financijskih sredstava s područja neuronskih mreža na područje umjetne inteligencije. Osnova za tu kampanju bila je nesposobnost tadašnjih modela da, između ostalog, nauče logičku funkciju `XOR`.

Unatoč smanjenju financijske podrške i interesa za neuronske mreže, razvoj se nastavio i u 1970-im godinama. Iako bez velikih otkrića, ovo je razdoblje u kojem su postavljeni čvrsti temelji za svojevrsnu renesansu ovoga područja.

Preporod neuronskih mreža

Početkom 1980-ih mnogi znanstvenici ponovo predlažu razvoj neuro-računala i neuronskih mreža. Poznati svjetski fizičar, John Hopfield, nakon iskazanog interesa za područje neuronskih mreža izdaje dva izrazito čitljiva članka o tom području [12, 13]. Ti članci, uz brojna predavanja, populariziraju cijelo područje iznova te je od tada ovo područje u konstantnom razvoju.

Ulaskom u 21. stoljeće i razvitkom hardvera dogodio se veliki globalni procvat cijelog područja te se iz dana u dan razvijaju novi modeli i rješavaju sve složeniji problemi.

Dio I

Teorija

Poglavlje 1

Osnovni model neuronske mreže

Glavnu temu ovog rada predstavljaju konvolucijske neuronske mreže te se podrazumijeva poznavanje i razumijevanje osnovnih pojmova iz područja neuronskih mreža. Baš zbog toga, ovo poglavlje ne sadrži detaljnu analizu svakog od sastavnih dijelova acikličke (eng. *feedforward*) neuronske mreže, već služi za postavljanje matematičkih temelja potrebnih za definiciju konvolucijske neuronske mreže.

1.1 Nadzirano učenje

Nadzirano učenje (eng. *supervised learning*) je zadatak učenja na označenim podacima. Drugim riječima, za određene ulazne podatke znamo očekivani izlaz te tu funkciju želimo generalizirati.

Definicija 1.1.1. *Neka su X i Y skupovi. Funkciju $T : X \to Y$ definiranu sa*

$$T = \{(x, y) \mid y \text{ je očekivani izlaz za ulaz } x \in X\}$$

zovemo ***skup podataka****.*

Skup podataka definirali smo kao funkciju koju želimo naučiti, odnosno generalizirati na neki nadskup njezine domene A. U praksi je trening skup podataka zapravo diskretan skup (ručno) označenih podataka, odnosno diskretna funkcija. Neka se npr. radi o trening skupu podataka fotografija lica koji koristimo u svrhu raspoznavanja osobe na slici. Tada, za svaku sliku $x \in A$, vrijednost $T(x)$ predstavlja identitet osobe, npr. jedinstveno ime i prezime ili identifikacijski broj.

Napomena 1.1.2. *Nadzirano učenje dijeli se na* ***klasifikaciju*** *i* ***regresiju****. Nadzirano učenje je klasifikacija ako je svaki element slike trening skupa podataka diskretna/nebrojčana vrijednost. Ako se pak radi o kontinuiranoj/brojčanoj vrijednosti govorimo o regresiji.*

Osim nadziranog učenja postoje još i nenadzirano (eng. unsupervised learning) i učenje podrškom (eng. reinforcement learning).

Za nenadzirano učenje možemo koristiti istu definiciju za skup podataka, uz uvjet da se radi o konstantnoj funkciji, odnosno neoznačenim podacima.

Učenje podrškom predstavlja način učenja nagrađivanjem i kažnjavanjem izvršenih akcija.

1.2 Neuron

Neuron, kao sastavna jedinica ljudskog mozga, predstavlja prvi korak u razumijevanju neuronske mreže. Prvi matematički model neurona daju McCulloch i Pitts u [21]. Radi se o modelu baziranom na biološkom neuronu. Ipak, mnogi detalji u vezi biološkog neurona su još uvijek nepoznanica pa se u praksi radi o izrazito jednostavnim matematičkim modelima. U nastavku dajemo definiciju neurona po uzoru na McCulloch–Pittsov model.

Definicija 1.2.1. *Neka je $n \in \mathbb{N}$. Funkciju $N^f : \mathbb{R}^n \to \mathbb{R}$ definiranu sa*

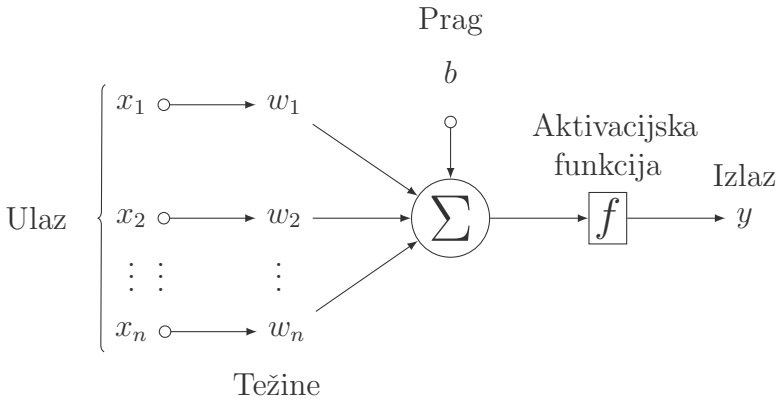
$$N^f(x_1, \ldots, x_n) = f\left(\sum_{i=1}^{n} x_i w_i - b\right)$$

nazivamo ***McCulloch–Pittsov neuron****. Realne brojeve $w_1, w_2, \ldots, w_n$ zovemo* ***težine****, realan broj b* ***prag (eng. bias)****, a funkciju $f : \mathbb{R} \to \mathbb{R}$* ***aktivacijska funkcija****.*

Težine možemo interpretirati kao kvantifikaciju važnosti svake od komponenti ulaznog vektora. Naime, suma unutar funkcije f je upravo skalarni produkt vektora $(x_1, \ldots, x_n)$ i $(w_1, \ldots, w_n)$. Uloga aktivacijske funkcije potaknuta je prirodom rada biološkog neurona, gdje se signal neurona prosljeđuje sljedećim neuronima ako i samo ako je ulazni podražaj tog neurona dovoljno snažan. U tu svrhu je u originalnoj definiciji McCulloch–Pittsovog neurona korištena funkcija signum, dok se danas, zbog načina učenja, koriste razne diferencijabilne funkcije. Primijetimo da je vrijednost praga baš ona vrijednost skalarnog produkta za koju se događa promjena vrijednosti funkcije signum, te je po tome i dobila naziv. Na slici 1.1 vidimo grafički prikaz neurona iz prethodne definicije.

Napomena 1.2.2. *Aktivacijska se funkcija u praksi odvaja od definicije samog neurona zbog jednostavnijeg poopćenja na razne vrste modela neurona i slojeva.*

Uzevši prethodnu napomenu u obzir, dajemo konačnu definiciju neurona.



Slika 1.1: Model neurona

Definicija 1.2.3. *Neka je $n \in \mathbb{N}$ i neka su $w_1, \ldots, w_n, b \in \mathbb{R}$ težine i prag, respektivno. Funkciju $N : \mathbb{R}^n \to \mathbb{R}$ definiranu sa*

$$N(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i w_i - b$$

nazivamo ***neuron****.*

1.3 Sloj

Ideja neurona je detektirati pojavu nekog apstraktnijeg svojstva na bazi ulaza, odnosno jednostavnijih svojstava. Složeni podaci imaju mnoštvo apstraktnih svojstava pa je sljedeći korak u izgradnji matematičkog modela neuronske mreže upravo definicija sloja. Intuitivno, sloj neurona predstavlja uređenu n-torku neurona koji imaju isti ulaz te svaki od njih detektira neko apstraktno svojstvo ulaznih podataka.

Definicija 1.3.1. ***Ulazni sloj*** *definiramo kao identitetu, odnosno funkciju* $\mathrm{id} : \mathbb{R}^n \to \mathbb{R}^n$ *definiranu sa*

$$\mathrm{id}(x) = x, \quad x \in \mathbb{R}^n.$$

Primijetimo da ulazni sloj predstavlja upravo ulazne podatke, te ga definiramo kao sloj, jer se u literaturi tako i navodi. U skladu s definicijama ostalih slojeva, koje navodimo u nastavku, i ulazni sloj definirali smo kao funkciju.

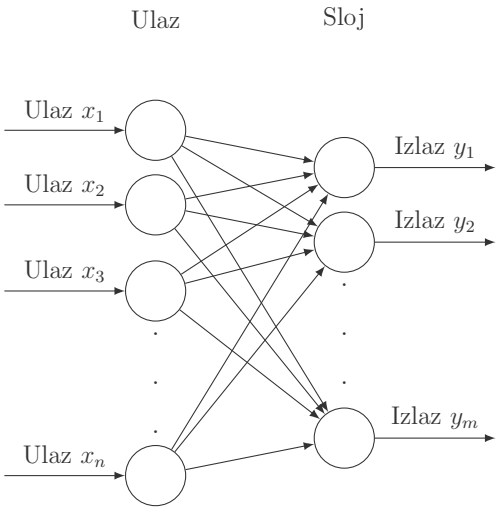
Analogno intuiciji u uvodnom dijelu, sada definiramo potpuno povezan sloj.

Definicija 1.3.2. *Neka su* $N_1^f, \ldots, N_m^f : \mathbb{R}^n \to \mathbb{R}$ *neuroni. Funkciju*

$$F(x) = (N_1^f(x), \ldots, N_m^f(x)), \quad x \in \mathbb{R}^n$$

nazivamo ***potpuno povezan sloj****.*

Naziv potpuno povezan proizlazi iz činjenice da svaki od neurona u sloju za ulaz ima cijeli vektor. Naime, ako to prikažemo u obliku grafa, svaki element ulaznog vektora vizualno je povezan sa svakim od neurona u sloju, i obratno. Takav graf može se vidjeti na slici 1.2.



Slika 1.2: Potpuno povezan sloj

Napomena 1.3.3. *Ako promotrimo definiciju potpuno povezanog sloja lako uočavamo matrični zapis funkcije* $F : \mathbb{R}^n \to \mathbb{R}^m$

$$F\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_n^1 & b^1 \\ w_1^2 & w_2^2 & \cdots & w_n^2 & b^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_1^m & w_2^m & \cdots & w_n^m & b^m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix},$$

gdje su $b^k \in \mathbb{R}$ *pragovi, a* $w_1^k, \cdots, w_n^k \in \mathbb{R}$ *težine* k*-tog neurona u sloju, za vrijednosti* $k = 1, 2, \ldots, m$*.*

Iz definicije neurona znamo da je svaki neuron definiran pragom te vektorom težine jednake dimenzionalnosti kao i ulazni vektor. Ako promotrimo problem obrade slika lako je vidjeti da se radi o prostorima visokih dimenzija. Na primjer, za sliku u boji dimenzija 640×480 radi se o 921600-dimenzionalnom vektorskom prostoru. Osim memorijskih ograničenja, valja imati na umu da je za izračun izlaza svakog od neurona potrebno isto toliko množenja i zbrajanja realnih brojeva, što predstavlja značajan problem. Takav pristup obradi slika u praksi nije upotrebljiv te se, baš zbog toga, koriste takozvani konvolucijski slojevi, koje ćemo definirati kasnije.

U napomeni 1.2.2 spomenuli smo da se, u praksi, iz McCulloch–Pittsove definicije neurona izbacuje aktivacijska funkcija radi jednostavnijeg poopćenja. Sada definiramo aktivacijski sloj, kojim se postiže analogan efekt nelinearnosti kao u originalnoj definiciji neurona.

Definicija 1.3.4. *Neka je $f : \mathbb{R} \to \mathbb{R}$ aktivacijska funkcija. Za proizvoljni $m \in \mathbb{N}$,* ***aktivacijski sloj*** *$A^f : \mathbb{R}^m \to \mathbb{R}^m$ definiramo sa*

$$A^f(x) = (f(x_1), \ldots, f(x_m)) \in \mathbb{R}^m, \quad x \in \mathbb{R}^m.$$

Primijetimo da za potpuno povezan sloj neurona F i aktivacijski sloj A^f, funkcija $A^f \circ F$ predstavlja potpuno povezan sloj McCulloch–Pittsovih neurona. Kao što smo već spomenuli, u praksi se ne koriste samo potpuno povezani slojevi. S obzirom da gotovo sve vrste slojeva u svojoj definiciji zahtijevaju aktivacijsku funkciju, ovo omogućuje lakše korištenje već definiranih aktivacijskih slojeva, odnosno funkcija.

Napomena 1.3.5. *U praksi se ne govori o aktivacijskim slojevima, već samo o aktivacijskim funkcijama. Naime, za aktivacijske funkcije se podrazumijeva definicija kao kod aktivacijskog sloja. U nastavku rada, izraz aktivacijska funkcija može se odnositi i na aktivacijsku funkciju i na aktivacijski sloj, ovisno o potrebi.*

1.4 Neuronska mreža

Nakon definicije potpuno povezanog sloja, dajemo i definiciju višeslojne neuronske mreže.

Definicija 1.4.1. *Neka je S_1 ulazni sloj, te $S_2 \ldots, S_k$ slojevi neurona takvi da je*

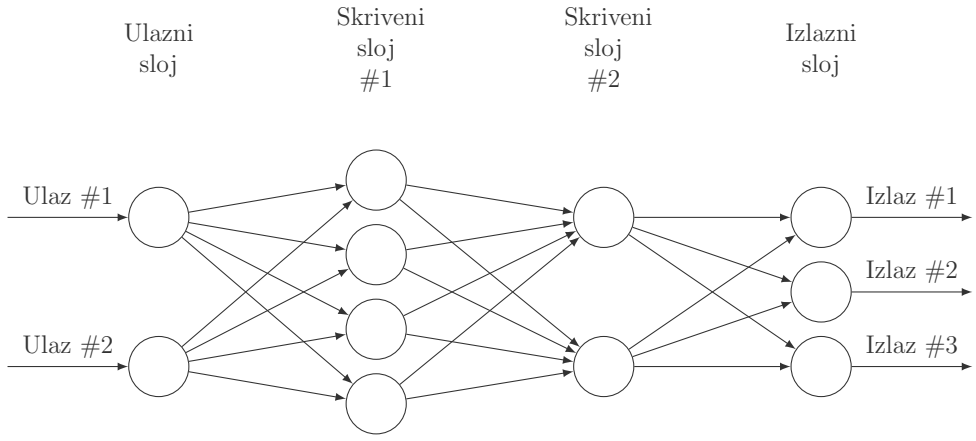
$$\mathcal{N} = S_k \circ \cdots \circ S_1$$

dobro definirana. Funkciju $\mathcal{N}$ nazivamo ***aciklička neuronska mreža (eng. feed-forward neural network)****.*

Sloj S_k zovemo ***izlazni sloj****, a uređenu $(k-2)$-torku $(S_2, \ldots, S_{k-1})$ nazivamo* ***skriveni sloj****.*

Zbog ovakve definicije neuronske mreže kao kompozicije slojeva, osiguran je nužan uvjet kompatibilnosti dimenzionalnosti izlaznih i ulaznih vektora između susjednih slojeva. Izravno iz definicije jasan je i sekvencijalni postupak izračunavanja vrijednosti neuronske mreže za neki ulazni vektor.

Primjer jednostavne neuronske mreže s ulaznim vektorom dimenzije 2 može se vidjeti na slici 1.3, gdje se skriveni sloj sastoji od jednog potpuno povezanog sloja s 4 neurona te jednog potpuno povezanog sloja s 2 neurona. Izlazni sloj sastoji se od 3 neurona.



Slika 1.3: Primjer modela jednostavne neuronske mreže

Napomena 1.4.2. *Primijetimo da se graf na slici 1.3 sastoji od ulaznog sloja te potpuno povezanih slojeva, odnosno, na njemu nisu prikazani aktivacijski slojevi. Aktivacijski slojevi nisu prikazani zbog jednostavnosti grafa, no nelinearna aktivacijska funkcija izrazito je bitan dio neuronske mreže. Naime, može se pokazati da je višeslojna neuronska mreža s linearnim aktivacijskim funkcijama zapravo linearan klasifikator, odnosno, ne postoji prednost takve mreže naspram samo jednog potpuno povezanog sloja.*

Budući da se radi o nadziranom učenju, imamo skup podataka za treniranje. Za neki ulazni vektor, neuronska mreža izračuna izlazni vektor kojeg možemo usporediti s očekivanim rezultatom te dobiti preciznost, tj. kvalitetu procjene. Tu dolazi ideja funkcije gubitka koja predstavlja grešku procjene.

Definicija 1.4.3. *Neka je T skup podataka za treniranje, $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$ neuronska mreža i $f_T : \mathbb{R}^m \to \mathbb{R}$. Realnu funkciju f_T zovemo* ***funkcija gubitka****.*

Primijetimo da je kompozicija funkcija $\mathcal{N}$ i f_T iz prethodne definicije valjana, tj. vrijedi $\mathcal{N} \circ f_T : \mathbb{R}^n \to \mathbb{R}$. Tako definirana funkcija izravno daje realan broj za svaki ulaz iz trening skupa podataka. Taj realan broj interpretiramo kao svojevrsnu veličinu pogreške. Osnovna ideja funkcije gubitka je odrediti pogrešku za određeni ulaz te tako postaviti temelje za učenje, odnosno minimizaciju te pogreške. Neka je $T : A \to A_T$ skup podataka i $\mathcal{N} : A \to A_T$ neuronska mreža. Jedan od načina na koji možemo definirati funkciju gubitka je

$$f_T(x) = \frac{1}{2}\|T(x) - \mathcal{N}(x)\|^2, \quad x \in A.$$

Očito se takva definicija zasniva na normi razlike izlaza neuronske mreže i očekivanog izlaza, što je izrazito prirodno. Funkcije gubitka nerijetko su složenije, no gotovo uvijek imaju intuitivnu ideju u svojoj pozadini.

1.5 Učenje

Sada znamo kako izgraditi neuronsku mrežu te izračunati vrijednost i grešku mreže za određene ulazne podatke. Ono što preostaje je problem određivanja vrijednosti pragova i težina za neurone. Taj proces zovemo učenje, a ključnu ulogu kod učenja neuronskih mreža igra iterativni optimizacijski algoritam zvan gradijentni spust.

Stohastički gradijentni spust

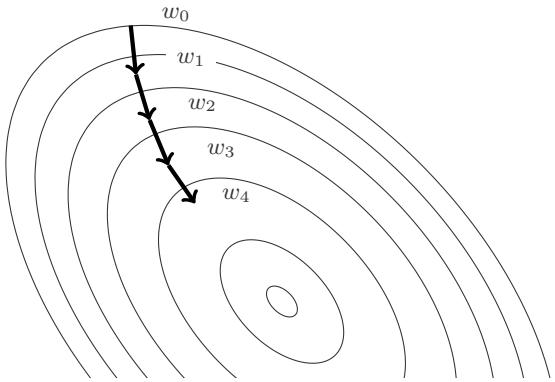
Definicija 1.5.1. *Neka je $f : \mathbb{R}^n \to \mathbb{R}$ diferencijabilna funkcija, $x \in \mathbb{R}^n$ i $\alpha > 0$. Iterativni algoritam definiran sa*

$$\begin{aligned} \theta_0 &= x, \\ \theta_{m+1} &= \theta_m - \alpha \nabla f(\theta_m), \quad m \in \mathbb{N}, \end{aligned}$$

nazivamo ***gradijentni spust (eng. gradient descent)****. Pozitivan realan broj α zovemo* ***brzina učenja (eng. learning rate)****.*

Intuitivno, jer $-\nabla f(\theta_m)$ označava smjer najstrmijeg pada u točki θ_m, u svakom se koraku algoritma približavamo (lokalnom) minimumu. Vizualno to možemo vidjeti na slici 1.4.

U strojnom učenju minimiziramo funkciju gubitka na cijelome trening skupu, odnosno, njezin prosjek. To je idealno, no često ipak presporo, jer se radi s izrazito velikim trening skupovima. Rješenje tog problema daje ideja **stohastičkog gradijentnog spusta**. U tom slučaju, u svakoj iteraciji algoritma koristi se samo jedan primjerak iz trening seta. To omogućuje bržu, ali često i neprecizniju konvergenciju



Slika 1.4: Grafički prikaz gradijentnog spusta

funkcije gubitka. Zbog toga se u praksi najčešće koristi kompromis, gdje se, umjesto jednog primjerka ili cijelog skupa, koristi neki malen podskup (eng. ***mini-batch***) koji omogućava i efikasno računanje i preciznu konvergenciju.

Danas su popularna razna proširenja i varijacije stohastičkog gradijentnog spusta, kao što su **AdaGrad** [6], **RMSProp** [28] i **Adam** [16], koji koriste ideje prilagodljive, odnosno, adaptivne brzine učenja.

Izravno računanje gradijenta na neuronskoj mreži nije lagan zadatak. Za zadnji sloj mreže možemo izravno izračunati pogrešku, na njemu računanje gradijenta ne predstavlja problem te je trivijalno primijeniti algoritam gradijentnog spusta, no za ostale slojeve to nije istina. Upravo taj problem rješava **algoritam propagiranja unazad (eng. *backpropagation*)**. Prije samog objašnjenja, potrebne su neke definicije kao matematička podloga.

Tenzori i vektorizacija

Tenzor je poopćenje skalara i vektora. Na primjer, tenzor reda 0 je skalar, tenzor reda 1 vektor, a tenzor reda 2 matrica. Slično kao što se vektor može intepretirati kao niz skalara i matrica kao niz vektora, tako se i tenzor reda 3 može interpretirati kao niz matrica, tj. tenzora reda 2. Specijalno, tenzor reda 3 koristi se kao struktura za opis `RGB` fotografije, gdje je kanal svake boje prikazan kao matrica. U nastavku dajemo nešto precizniju definiciju tenzora.

Definicija 1.5.2. ***Tenzor*** *reda* 0 *je skalar. Tenzor* T_n *reda* n*, za* $n > 0$*, definiramo kao*

$$T_n = (T_{n-1}^1, \ldots, T_{n-1}^m),$$

gdje su T_{n-1}^i*, za* $i = 0, 1, \ldots, m$*, tenzori reda* $(n-1)$*, međusobno istih dimenzija.*

Napomena 1.5.3. *Analogno oznakama za vektore i matrice, prostor istodimenzionalnih tenzora reda n nad poljem* $\mathbb{R}$ *označavamo sa*

$$\mathbb{R}^{d_1 \times \cdots \times d_n}, \quad d_1, \ldots, d_n \in \mathbb{N}.$$

Definicija 1.5.4. *Neka su*

$$T_n = (T_{n-1}^1, \ldots, T_{n-1}^m) \in \mathbb{R}^{d_1 \times \cdots \times d_{n-1} \times m}$$

i

$$S_n = (S_{n-1}^1, \ldots, S_{n-1}^k) \in \mathbb{R}^{d_1 \times \cdots \times d_{n-1} \times k}$$

tenzori reda n. ***Konkatenaciju tenzora*** *definiramo formulom*

$$T_n \oplus S_n = (T_{n-1}^1, \ldots, T_{n-1}^m, S_{n-1}^1, \ldots, S_{n-1}^k) \in \mathbb{R}^{d_1 \times \cdots \times d_{n-1} \times (m+k)}.$$

Definicija 1.5.5. *Neka je* $T_n = (T_{n-1}^1, \ldots, T_{n-1}^m)$ *tenzor reda n.* ***Vektorizaciju tenzora*** *definiramo sa*

$$\begin{aligned} &\operatorname{vec}(T_0) = (T_0), \\ &\operatorname{vec}(T_n) = \operatorname{vec}(T_{n-1}^1) \oplus \cdots \oplus \operatorname{vec}(T_{n-1}^m). \end{aligned}$$

Algoritam propagiranja unazad

Proces učenja neuronskih mreža oslanja se na gradijentni spust, pa samim time i vektorsku analizu. Ključne su definicije gradijenta vektorske funkcije jedne varijable, Jacobijeve matrice vektorske funkcije više varijabli te derivacija kompozicije funkcija. Pripadne definicije i teoremi mogu se pronaći u [20].

Derivacije funkcija po tenzorima mogu se svesti na gore navedene slučajeve, uz pomoć vektorizacije tenzora.

Neka je $\mathcal{N} = S_k \circ \cdots \circ S_1$ neuronska mreža i f pripadna funkcija gubitka. Za $i \in 1, 2, \ldots, k$, označimo s $\boldsymbol{x}^i$ ulazni tenzor, a s $\boldsymbol{w}^i$ tenzor svih parametara (težina i pragova) sloja S_i. Primijetimo da se za potpuno povezane slojeve radi o ulaznom vektoru $\boldsymbol{x}^i$ i matrici parametara $\boldsymbol{w}^i$, no zbog generalizacije govorimo o tenzorima.

Za primjenu algoritma gradijentnog spusta S_i potrebno je izračunati $\frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{w}^i)}$. Iz jednakosti

$$\frac{\partial f}{\partial \boldsymbol{x}^T} = \left(\frac{\partial f}{\partial \boldsymbol{x}}\right)^T$$

i pravila za derivaciju kompozicije funkcija slijedi izraz

$$\frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{w}^i)^T} = \frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{x}^{i+1})^T} \frac{\partial \operatorname{vec}(\boldsymbol{x}^{i+1})}{\partial \operatorname{vec}(\boldsymbol{w}^i)^T}. \tag{1.1}$$

Intuitivno, jer je $\boldsymbol{x}^{i+1}$ zapravo izlazni tenzor sloja S_i, izraz $\frac{\partial \operatorname{vec}(\boldsymbol{x}^{i+1})}{\partial \operatorname{vec}(\boldsymbol{w}^i)^T}$ odgovara na pitanje promjene izlaznog tenzora u ovisnosti na promjenu tenzora parametara istog sloja. Taj izraz, dakle, ovisi samo o funkciji S_i te ga nije teško izračunati. Ono što preostaje je prvi dio izraza. Za sloj S_i vrijedi i jednakost

$$\frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{x}^i)^T} = \frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{x}^{i+1})^T} \frac{\partial \operatorname{vec}(\boldsymbol{x}^{i+1})}{\partial \operatorname{vec}(\boldsymbol{x}^i)^T}. \tag{1.2}$$

Drugi dio izraza je i u ovom slučaju intuitivan, jer odgovara na pitanje promjene izlaznog tenzora sloja u ovisnosti na promjenu ulaznog tenzora sloja, te ga se može izravno računati. Prvi je dio identičan kao i u (1.1) i može se izračunati primjenom formule (1.2) za sloj S_{i+1}.

Sada se prirodno nameće postupak i redoslijed izračuna formula (1.1) i (1.2) za svaki od slojeva neuronske mreže. Označimo s $\boldsymbol{x}^{i+1}$ ulazni tenzor funkcije gubitka. Znamo izračunati $\frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{x}^{i+1})}$, a onda i $\frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{x}^{i+1})^T}$, odakle iz (1.2), slijedi

$$\frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{x}^i)^T} = \frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{x}^{i+1})^T} \frac{\partial \operatorname{vec}(\boldsymbol{x}^{i+1})}{\partial \operatorname{vec}(\boldsymbol{x}^i)^T}.$$

Izraz $\frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{w}^i)}$ računa se analogno. Sada je ostalo primijetiti da se za S_{i-1} vrijednosti (1.1) i (1.2) računaju jednostavno, koristeći upravo izračunatu $\frac{\partial f}{\partial \operatorname{vec}(\boldsymbol{x}^i)^T}$. Postupak je trivijalno nastaviti, redom za sve slojeve do S_1, čime smo za svaki sloj dobili jednostavan način izračuna formule (1.1), koja je potrebna za primjenu algoritma gradijentnog spusta.

1.6 Generalizirana definicija sloja

Na osnovi prethodnog poglavlja o učenju dajemo općenitije definicije sloja i neuronske mreže.

Definicija 1.6.1. *Neka je T skup podataka za treniranje i $f_T : \mathbb{R}^{d_1 \times \cdots \times d_p} \to \mathbb{R}$. Funkciju f_T zovemo funkcija gubitka.*

Definicija 1.6.2. *Neka je $f : \mathbb{R}^{d_1 \times \cdots \times d_p} \to \mathbb{R}$ funkcija gubitka i*

$$F : \mathbb{R}^{x_1 \times \cdots \times x_n} \times \mathbb{R}^{w_1 \times \cdots \times w_k} \to \mathbb{R}^{y_1 \times \cdots \times y_m}$$

funkcija. Uređenu trojku

$$\left(F, \frac{\partial f}{\partial w}, \frac{\partial f}{\partial x}\right)$$

*nazivamo **sloj**. Tenzor $x \in \mathbb{R}^{x_1 \times \cdots \times x_n}$ zovemo ulaz, dok tenzor $w \in \mathbb{R}^{w_1 \times \cdots \times w_k}$ zovemo parametri sloja.*

Napomena 1.6.3. *Izrazi* $\frac{\partial f}{\partial w}$ *i* $\frac{\partial f}{\partial x}$ *iz prethodne definicije nisu nužno parcijalne derivacije, već samo funkcije koje intuitivno predstavljaju te funkcije. Ova definicija dopušta korištenje slojeva koji nisu bazirani na derivabilnoj funkciji, kao što ćemo vidjeti kod aktivacijske funkcije* `ReLU`*. Ipak, najčešće je cilj doista koristiti derivabilne funkcije za što precizniju implementaciju gradijentnog spusta.*

Definicija 1.6.4. *Neka su* $S_1, \ldots, S_k$ *slojevi i* $F_1, \ldots, F_k$ *pripadne funkcije, odnosno, prvi elementi uređenih trojki, takvi da je*

$$\mathcal{N} = F_k \circ \cdots \circ F_1$$

dobro definirana. Funkciju $\mathcal{N}$ *nazivamo aciklička neuronska mreža (eng. feedforward neural network).*

Sada znamo sve što je potrebno definirati za definiciju i implementaciju slojeva neuronskih mreža.

Potpuno povezan sloj

Neka je f funkcija gubitka. U definiciji 1.3.2 dali smo definiciju potpuno povezanog sloja. Ovdje proširujemo istu u smislu promjene interpretacije težina i praga. Naime, parametri će se, također, promatrati kao ulazne varijable, što je zahtjev koji slijedi izravno iz algoritma propagiranja pogreške unazad. Time smo dobili definiciju funkcije F iz uređene trojke sloja.

Primijetimo sada da u formulama (1.1) i (1.2) vrijedi $F(\boldsymbol{x}^i, \boldsymbol{w}^i) = \boldsymbol{x}^{i+1}$ pa je zapravo potrebno izračunati vrijednosti $\frac{\partial F(\boldsymbol{x}^i, \boldsymbol{w}^i)}{\mathrm{vec}(\boldsymbol{w}^i)^T}$ i $\frac{\partial F(\boldsymbol{x}^i, \boldsymbol{w}^i)}{\mathrm{vec}(\boldsymbol{x}^i)^T}$, što se svodi na jednostavnu sumaciju elemenata, jer se F može interpretirati kao k-multilinearna funkcija. Zatim, izravno iz definicije k-multilinearne funkcije slijedi da su parcijalne derivacije, zapravo, derivacije linearnih funkcija, što je lako izračunati. Prvi član formula dobije se rekurzivnom primjenom, odnosno, propagiranjem unazad na ostalim slojevima mreže. Ovime su dane definicije svih članova uređene trojke definicije sloja, odnosno, sve što je potrebno za programsku implementaciju potpuno povezanog sloja.

1.7 Univerzalni aproksimator

Teorem 1.7.1 (Teorem o univerzalnom aproksimatoru)**.** *Za svaku neprekidnu funkciju na kompaktnom podskupu od* $\mathbb{R}^n$*, postoji neuronska mreža, sastavljena od jednog potpuno povezanog sloja i sigmoidne aktivacijske funkcije (v. 2.2), koja ju aproksimira.*

Dokaz. Vidi [3]. □

Napomena 1.7.2. *Nije bitno da aktivacijska funkcija bude baš sigmoidna funkcija, već da se radi o nelinearnoj funkciji.*

S obzirom da je i takav jednostavan model mreže univerzalni aproksimator, nameće se pitanje odakle potreba za višeslojnim neuronskim mrežama. Prethodni teorem dokazuje egzistenciju takve mreže, no ne govori ništa o potrebnom broju neurona u potpuno povezanom sloju te mreže koji može biti nerazumno velik. Pokazalo se da više jednostavnih slojeva postiže puno bolji omjer moći aproksimacije i potrebnog broja operacija, odnosno, vremenske složenosti mreže. Upravo to je glavni razlog zašto se umjesto ovog, jednostavnog, modela preporuča korištenje dubokih neuronskih mreža [9].

Poglavlje 2

Konvolucijska neuronska mreža

Ovaj rad bazira se na primjeni neuronskih mreža u obradi slika. Slike u boji sastoje se od velikog broja piksela i 3 kanala boja. Čak i slike relativno malih dimenzija predstavljaju problem velike vremenske i prostorne složenosti. Upravo taj problem rješavaju konvolucijski slojevi kojima se bavimo u ovom poglavlju. Osim konvolucijskih i potpuno povezanih slojeva, za konstrukciju konvolucijske neuronske mreže potrebni su još neki slojevi koje definiramo u nastavku.

2.1 `ReLU` i modifikacije

Sloj i funkcija `ReLU`

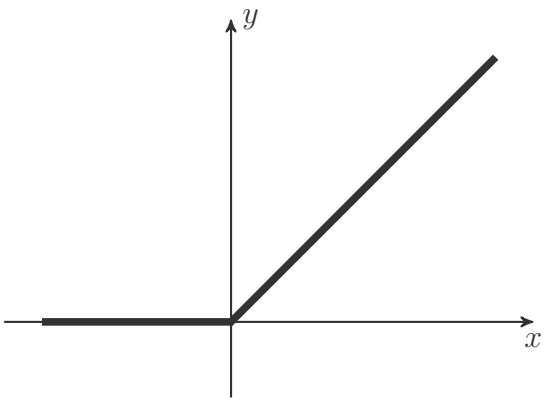
Definicija funkcije `ReLU` (eng. *Rectified Linear Unit*) dana je sa

$$\texttt{ReLU}\,(x) = \max\{0, x\}.$$

Graf ove funkcije i podrijetlo naziva odrezana linearna funkcija mogu se vidjeti na slici 2.1. Višedimenzonalnija verzija, koja se koristi kod neuronskih mreža, jednostavno je proširenje ove definicije. Naime, $\max\{0, x\}$ se primijeni na svakom skalaru x ulaznog tenzora. Primijetimo da ovaj sloj nema parametara te nema potrebe za učenjem, odnosno, skup parametara w je prazan pa možemo definirati $\frac{\partial f}{\partial w} = 0$. Za funkciju gubitka f, treći element uređene trojke iz definicije sloja definira se sa

$$\frac{\partial f}{\partial x} = \begin{cases} \frac{\partial f}{\partial y}, & \text{za } x > 0, \\ 0, & \text{inače,} \end{cases}$$

gdje je y ulaz sljedećeg sloja u mreži. Intuitivno, za $x > 0$ zanemarujemo ovaj sloj, jer je linearan s nagibom 1. Definicija je, kao i za F, dana s realnom funkcijom



Slika 2.1: Graf funkcije `ReLU`

realne varijable, no proširenje za ulazni tenzor proizvoljnog reda opet je trivijalno, po skalarima. Ovime je potpuno definiran `ReLU` sloj. Gašenje negativnih vrijednosti iz prethodnog sloja, odnosno, ulaznog tenzora, intuitivno je veoma korisno. Naime, neke uzorke koji se pojavljuju kod ulaznih vrijednosti zapravo želimo zanemariti, jer ne utječu na konačan rezultat te ih, postavljanjem na 0, funkcija `ReLU` jednostavno zanemari. Više o motivaciji definicije kao rješenju problema nestajanja gradijenta (eng. *vanishing gradient*) može se pronaći u [8].

Propusni `ReLU`

Iako je gašenje negativnih vrijednosti iznimno korisno svojstvo funkcije `ReLU`, ono može predstavljati svojevrstan problem pa se prilikom korištenja funkcije `ReLU` može pojaviti problem umirućeg `ReLU`. Naime, ako je ulazni tenzor `ReLU` sloja gotovo u potpunosti negativan, to će zaustaviti proces učenja zbog načina na koji propagiranje unazad radi. Iako se to rijetko događa i problem se može riješiti ponovnim pokretanjem algoritma, odnosno, drugačije inicijaliziranim početnim parametrima, postoje modifikacije funkcije `ReLU` koje pokušavaju riješiti ovaj problem na jednostavne načine.

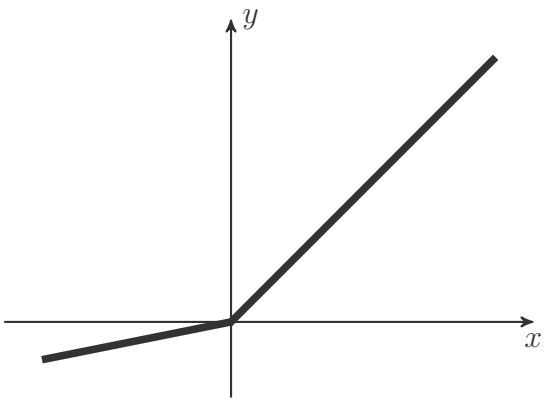
Jedna od modifikacija je propusni `ReLU` (eng. *Leaky* `ReLU`), s oznakom `LReLU` [19]. Definicija modifikacije slična je definiciji funkcije `ReLU` i glasi

$$\texttt{LReLU}\,(x) = \max\{\alpha x, x\}, \quad \alpha \in \mathbb{R}^+.$$

Realan broj α je hiperparametar[1] koji se odabire pri definiciji sloja. Primijetimo da za $\alpha = 0$ dobijemo upravo funkciju `ReLU`. Vrijednost parametra α je najčešće u

[1]Hiperparametar je parametar na kojem se ne provodi proces učenja. Primjer jednog hiperparametra je brzina učenja (eng. *learning rate*).

okolini broja 0.2, odnosno, dovoljno velika da se izbjegne problem umirućeg `ReLU`, a s druge strane dovoljno malena da se zadrži svojstvo gašenja negativnih vrijednosti. Na slici 2.2 može se vidjeti graf funkcije `LReLU` za $\alpha = 0.2$.



Slika 2.2: Graf funkcije `LReLU`

Slično kao i `ReLU`, funkcija `LReLU` nema parametara pa možemo definirati $\frac{\partial f}{\partial w} = 0$. Neka je f opet funkcija gubitka. Posljednji element definicije sloja dan je formulom

$$\frac{\partial f}{\partial x} = \begin{cases} \frac{\partial f}{\partial y}, & \text{za } x > 0, \\ 0, & \text{za } x = 0, \\ \alpha \frac{\partial f}{\partial y}, & \text{inače.} \end{cases}$$

Primijetimo da je opet, kao i kod analogne definicije za `ReLU`, ova funkcija bazirana na derivaciji s proširenjem u točki 0, gdje ona nije derivabilna.

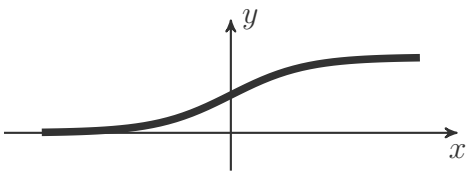
Osim ove modifikacije, postoji i parametrizirana verzija `LReLU`, pod nazivom `PReLU`, gdje vrijednost α nije hiperparametar, već parametar kojeg možemo učiti [11]. Još jedna alternativa je i proširenje eksponencijalnom funkcijom `ELU` [2]. Sve one imaju svoje prednosti i mane, koje često ovise o arhitekturi mreže koja se trenira.

2.2 `softmax`

Sigmoidna funkcija

Funkcija $\texttt{sigmoid} : \mathbb{R} \to [0, 1]$ definirana je izrazom

$$\texttt{sigmoid}\,(x) = \frac{e^x}{e^x + 1}.$$



Slika 2.3: Graf funkcije `sigmoid`

Graf sigmoidne funkcije prikazan je na slici 2.3. Iako se sigmoidna funkcija rijetko koristi kao aktivacijska funkcija, može se koristiti nakon posljednjeg sloja, odnosno, za normalizaciju izlaznih vrijednosti na segment $[0, 1]$. Takav broj izrazito se prirodno interpretira kao vjerojatnost.

Ukoliko se radi o problemu klasifikacije u više, međusobno disjunktnih, kategorija, mogu se javiti problemi, jer se može dogoditi slučaj gdje ulazna vrijednost sa stopostotnom vjerojatnošću spada pod više kategorija. To je posljedica činjenice da se sigmoidna funkcija proširuje na proizvoljan tenzor jednostavnom primjenom po elementima. Upravo taj problem rješava poopćenje sigmoidne funkcije.

Sloj i funkcija σ (`softmax`)

Normalizirana eksponencijalna funkcija s oznakom σ generalizacija je logističke funkcije, koja n-dimenzionalni vektor iz $\mathbb{R}^n$ preslikava u $[0, 1]^n$, uz uvjet da je suma elemenata izlaznog vektora jednaka 1. Njezina definicija glasi

$$\sigma\left((x_1, \ldots, x_n)\right) = \left(\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \ldots, \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}}\right).$$

Definicija se jednostavno proširuje na tenzore proizvoljnog reda. Opet, jer ni ova funkcija nema parametara za učenje, možemo definirati $\frac{\partial f}{\partial w} = 0$. Za potrebe definicije trećeg elementa uređene trojke sloja slijedi definicija Kroneckerove delta funkcije.

Definicija 2.2.1. ***Kroneckerova delta funkcija*** $\delta : \mathbb{N} \times \mathbb{N} \to \{0, 1\}$ *definira se izrazom*

$$\delta_{ij} = \begin{cases} 1, & za\ i = j \\ 0, & inače \end{cases}, \quad i, j \in \mathbb{N}.$$

Parcijalna derivacija funkcije σ po elementima dana je izrazom

$$\frac{\partial \sigma(x)_i}{\partial x_j} = \sigma(x)_i \left(\delta_{ij} - \sigma(x)_j\right),$$

odakle slijedi $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} \frac{\partial \sigma}{\partial x}$, gdje je $y = \sigma(x)$ ulazni vektor sljedećeg sloja. Ovime je u potpunosti definiran `softmax` sloj.

2.3 Konvolucijski sloj

Konvolucija

Za potrebe definicije operatora konvolucije slijedi definicija zrcaljenja matrica.

Definicija 2.3.1. *Sa $Z_n \in \mathbb{R}^{n\times n}$ označavamo matricu oblika*

$$\begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix},$$

odnosno, matricu s jedinicama na sporednoj dijagonali.

Primijetimo da AZ_m zrcali matricu $A \in \mathbb{R}^{n\times m}$ vertikalno, dok ju Z_nA zrcali horizontalno.

Definicija 2.3.2. *Neka su $A, B \in \mathbb{R}^{n\times m}$ matrice. Operator **konvolucije matrica istih dimenzija** $*$ definiramo kao*

$$A * B = \langle \mathrm{vec}(A) \,|\, \mathrm{vec}(Z_nBZ_m) \rangle ,$$

*gdje je $\langle \cdot | \cdot \rangle$ skalarni produkt vektora. Matricu A zovemo **slika**, dok matricu B nazivamo **filter**.*

Prethodna definicija prilično je jednostavna, naime, filter B zrcalimo oko centra, pomnožimo matrice po elementima te potom sumiramo dobivene elemente. U definiciji se radi o matricama istih dimenzija, no to želimo proširiti tako da dozvolimo filter manjih dimenzija od slike. Proces konvolucije matrica različitih dimenzija prikazan je na slici 2.4.

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 8 & 7 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 14 & 18 \\ 26 & 28 & 28 \end{bmatrix}$$

Slika 2.4: Primjer konvolucije matrica

Definicija 2.3.3. *Neka je $n \leq m$. Za $i = 1, 2, \ldots, m-n+1$, definiramo matricu $R_i^{n\times m} \in \mathbb{R}^{n\times m}$ tako da za definiciju $R_i^{n\times m}$ kao niz vektora*

$$R_i = (r^1, r^2, \ldots, r^i, \ldots, r^{i+n-1}, \ldots, r^m)$$

vrijedi

$$(r^i, \ldots, r^{i+n-1}) \in \mathbb{R}^{n\times n}$$

je jedinična matrica i

$$r^j = (0, \ldots, 0) \in \mathbb{R}^n, \qquad (j < i) \vee (j \geq i+n).$$

Naposlijetku definiramo i $S_i^{m\times n} = (R_i^{n\times m})^T$.

Primjer 2.3.4. *Primjeri matrica iz prethodne definicije su*

$$R_2^{3\times 4} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad i \quad S_1^{3\times 2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

Primijetimo sada da vrijedi

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 5 \\ 7 & 8 \\ 10 & 11 \end{bmatrix},$$

što je upravo podmatrica srednje matrice dimenzija 3×2, s prvim elementom na indeksu $(2, 1)$.

Primijetimo da kombinacija zadnje dvije definicije daje definiciju operatora konvolucije, gdje su dimenzije slike veće ili jednake dimenzijama filtera, odnosno, procesa prikazanog na slici 2.4.

Kod izračuna gradijenata potrebnih za definiciju sloja bit će prikladnija definicija konvolucije pomoću matričnog množenja, koju gradimo u nastavku. Označimo prvo s $\operatorname{div}(a, b) = \lfloor a/b \rfloor$ i $\operatorname{mod}(a, b) = a - \operatorname{div}(a, b)$ cjelobrojno dijeljenje i ostatak pri cjelobrojnom dijeljenju, respektivno.

Definicija 2.3.5. *Neka je $M \in \mathbb{R}^{n\times m}$. Za $a, b \in \mathbb{N}$ definiramo operator φ sa*

$$\varphi(M) = (\boldsymbol{x}^0, \boldsymbol{x}^1, \ldots, \boldsymbol{x}^{(n-a+1)(m-b+1)-1})^T \in \mathbb{R}^{(n-a+1)(m-b+1)\times ab},$$

gdje je

$$\boldsymbol{x}^i = \operatorname{vec}(R^{a\times n}_{\operatorname{mod}(i,\ n-a+1)} M S^{m\times b}_{\operatorname{div}(i,\ n-a+1)}).$$

Ovakva definicija funkcije φ ekvivalentna je funkciji `im2row`, transponiranoj inačici funkcije `im2col`, definiranoj u [29] za programski alat `MATLAB`. Primijetimo da je $\varphi(M)^T$ zapravo niz (vektoriziranih) podmatrica od M. Radi se baš o podmatricama prikazanim na slici 2.4, odnosno, podmatricama na kojima izvršavamo konvoluciju filterom istih dimenzija. Za sliku M i filter f vrijedi

$$\varphi(M)\,\mathrm{vec}(f) = \mathrm{vec}(M * f).$$

Znajući dimenzije od $M * f$, iz vektora $\varphi(M)\,\mathrm{vec}(f)$ lako izvodimo $M * f$. Slijedi da je s $\varphi(M)\,\mathrm{vec}(f)$ dana definicija operatora konvolucije slike M i filtera f. Također, izravno iz takve definicije slijedi da je konvolucija matrica linearan operator.

Preostaje još proširiti definiciju konvolucije na tenzore reda 3. Neka je $\boldsymbol{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l}$ ulazni tenzor, tj. slika. Umjesto matrice, za filter $f \in \mathbb{R}^{H \times W \times D^l}$ koristimo tenzor reda 3. Za razliku od definicije konvolucije matrica pomoću φ, gdje se radi o podmatricama, ovdje je riječ o (pod)tenzorima reda 3, koje vektoriziramo u redove. Primijetimo da je posljednja dimenzija filtera f i ulaza $\boldsymbol{x}^l$ jednaka pa se pomak opet događa samo po visini i širini, dok u svakom koraku, umjesto podmatrice, uzimamo D^l podmatrica po dubini te njih slažemo u vektor. Preciznije, imamo

$$\boldsymbol{x}^l = (a^1, a^2, \ldots, a^{D^l}) \in \mathbb{R}^{H^l \times W^l \times D^l}$$

i

$$f = (b^1, b^2, \ldots, b^{D^l}) \in \mathbb{R}^{H \times W \times D^l}.$$

Potom definiramo operator ϕ tako da, uz oznake

$$\varphi(a^i) = \left(c_i^1, \ldots, c_i^{(H^l-H+1)(W^l-W+1)}\right)^T,$$

vrijedi

$$\phi(x^l) = \left(\bigoplus_{i=1}^{D^l} c_i^1, \ldots, \bigoplus_{i=1}^{D^l} c_i^{(H^l-H+1)(W^l-W+1)}\right)^T.$$

Primijetimo da je $c_i^j \in \mathbb{R}^{HW}$, pa onda i $\varphi(a_i) \in \mathbb{R}^{(H^l-H+1)(W^l-W+1)\times HW}$. Iz toga slijedi $\phi(\boldsymbol{x}^l) \in \mathbb{R}^{(H^l-H+1)(W^l-W+1)\times HWD^l}$. Sada opet, kao i za φ, vrijedi tvrdnja

$$\phi(\boldsymbol{x}^l)\,\mathrm{vec}(f) = \mathrm{vec}(\boldsymbol{x}^l * f),$$

odakle, jer znamo dimenzije $\boldsymbol{x}^l * f$, možemo jednostavno dobiti konvoluciju. Za potpunu definiciju konvolucijskog neurona preostaje nam još dodati vrijednost praga. Za filter f i $b \in \mathbb{R}$ imamo

$$\phi(\boldsymbol{x}^l)\,\mathrm{vec}(f) + b, \tag{2.1}$$

gdje je zbrajanje između vektora i realnog broja definirano kao pribrajanje tog realnog broja svakom elementu vektora. Napomenimo još da je (2.1) zapravo vektor, no vrlo jednostavno se pretvara u matricu, odnosno, oblik

$$\boldsymbol{x}^l * f + b,$$

što je finalna definicija konvolucijskog neurona.

Analogno potpuno povezanom sloju i ovdje će se konvolucijski neuroni slagati u konvolucijski sloj. Neka je $\boldsymbol{f} = (f^1, \ldots, f^D)$ niz filtera iz $\mathbb{R}^{H \times W \times D^l}$, odnosno, tenzor reda 4. Tada $\boldsymbol{f}$ možemo zapisati u obliku matrice kao

$$F = (\mathrm{vec}(f^1), \ldots, \mathrm{vec}(f^D)) \in \mathbb{R}^{HWD^l \times D}.$$

Primijetimo da se sada konvolucija D filtera na ulaz x^l može primijeniti koristeći izraz

$$\phi(\boldsymbol{x}^l)F + \boldsymbol{b},$$

gdje je $\boldsymbol{b} = (b^1, \ldots, b^D) \in \mathbb{R}^D$, a zbrajanje matrice i vektora definiramo sa

$$\begin{bmatrix} a_{11} & \ldots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \ldots & a_{nm} \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_{11} + b_1 & \ldots & a_{1m} + b_n \\ \vdots & \ddots & \vdots \\ a_{n1} + b_1 & \ldots & a_{nm} + b_n \end{bmatrix}.$$

Ovime je definiran prvi element uređene trojke iz definicije sloja.

Napomena 2.3.6. *Ovako definirana zbrajanja između vektora i realnih brojeva te matrica i vektora koriste se kako bi se naglasilo da se radi o dijeljenim vrijednostima. Naime, umjesto zbrajanja vektora s realnim brojem $(a_1, \ldots, a_n) + b$, može se definirati $(a_1, \ldots, a_n) + (b, \ldots, b)$, no iz toga nije izravno jasno da svaki sloj ima samo jednu realnu vrijednost kao prag, odnosno, da se nakon koraka treniranja ne smije dogoditi da se jedna komponenta početnog vektora $(b, \ldots, b)$ razlikuje od druge.*

Napomena 2.3.7. *Jedno od pojednostavljenja koje ćemo koristiti u nastavku je zanemarivanje praga, odnosno, koristimo definiciju konvolucijskog sloja takvu da vrijedi*

$$\mathrm{vec}(\boldsymbol{y}) = \mathrm{vec}(\boldsymbol{x}^{l+1}) = \mathrm{vec}(\phi(\boldsymbol{x}^l)F). \tag{2.2}$$

Primijetimo još da je za ulaznu sliku dimenzija $H^l \times W^l \times D^l$ i filter $H \times W \times D^l$, izlaz tenzor dimenzija $(H^l - H + 1) \times (W^l - W + 1)$, odnosno, tenzor manjih dimenzija nego ulazni. U praksi je često poželjno da ulazni i izlazni vektor budu istih dimenzija, do na dubinu. Za taj efekt koristi se nadopunjavanje nulom (eng. zero padding), odnosno, dodavanjem redova iznad i ispod te stupaca lijevo i desno, kako bi ulazni

tenzor zadobio dimenzije $(H^l + H - 1) \times (W^l + W - 1) \times D^l$. *Time su dimenzije izlazne matrice upravo* $H^l \times W^l$. *U sklopu ovog rada nećemo koristiti nadopunjavanje nulom.*

Još je jedan važan dio u praksi kod operatora konvolucije – pomak (eng. stride). Radi se o pomaku kod izgradnje operatora ϕ. *Na slici 2.4 prikazana je konvolucija s pomakom* $s = 1$. *Pomak predstavlja promjenu broja elemenata pri izgradnji operatora konvolucije. Na primjer, za sliku dimenzija* 4×4, *filter dimenzija* 2×2 *te pomak* 2, *izlazna matrica konvolucije ima dimenzije* 2×2, *odnosno, izgrađena je od* 4 *podmatrice slike, dimenzija* 2×2. *Analogan primjer vrijedi i za konvoluciju tenzora. Kao i prethodne dvije stavke, i ovaj ćemo slučaj pojednostaviti te uvijek pretpostaviti vrijednost pomaka* $s = 1$.

Zašto konvolucija?

Slika 2.5 prikazuje rezultat primjene konvolucije za filtere K i K^T, gdje je

$$K = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

Primijetimo da za filter K vrijedi da je rezultat konvolucije veći, ako je razlika između gornjeg i donjeg reda piksela velika, odnosno, on detektira horizontalne rubove. Isto tako slijedi da K^T detektira vertikalne rubove. Filteri K i K^T zovu se **Sobelovi operatori** i tek su jedni od mnoštva korisnih konvolucijskih filtera. Konvolucijske neuronske mreže potaknute su upravo idejom o učenju, odnosno, pronalasku novih, apstraktnih filtera za detekciju složenijih svojstava.

Slika 2.5: Originalna slika te rezultati primjene konvolucije filtera K i K^T

Kroneckerov umnožak

Jedan od alata potreban za definiciju gradijenata je Kroneckerov umnožak matrica. Neka su $A \in \mathbb{R}^{n\times m}$ i $B \in \mathbb{R}^{p\times q}$ matrice. Kroneckerov umnožak matrica A i B, s oznakom $A \otimes B \in \mathbb{R}^{np\times mq}$, definiran je blok matricom

$$A \otimes B = \left[\begin{array}{c|c|c} a_{11}B & \cdots & a_{1m}B \\ \hline \vdots & \ddots & \vdots \\ \hline a_{n1}B & \cdots & a_{nm}B \end{array}\right].$$

Kroneckerov umnožak ima sljedeća svojstva

$$(A \otimes B)^T = A^T \otimes B^T, \tag{2.3}$$

$$\operatorname{vec}(AXB) = (B^T \otimes A)\operatorname{vec}(X). \tag{2.4}$$

Koristeći Kroneckerov umnožak matrica, definiciju konvolucijskog sloja (2.2) možemo zapisati u obliku

$$\operatorname{vec}(\boldsymbol{y}) = \operatorname{vec}(\phi(\boldsymbol{x}^l)FI) = (I \otimes \phi(\boldsymbol{x}^l))\operatorname{vec}(F), \tag{2.5}$$

$$\operatorname{vec}(\boldsymbol{y}) = \operatorname{vec}(I\phi(\boldsymbol{x}^l)F) = (F^T \otimes I)\operatorname{vec}(\phi(\boldsymbol{x}^l)), \tag{2.6}$$

gdje je I jedinična matrica odgovarajućih dimenzija.

Ažuriranje parametara

U nastavku je s $Y \in \mathbb{R}^{(H^{l+1}W^{l+1})D}$ označena matrica dobivena promjenom dimenzija izravno iz $\boldsymbol{y}$, odnosno, $\boldsymbol{x}^{l+1}$. U ovom radu ćemo koristiti sva tri oblika, bez posebne napomene o promjeni između istih. Neka je z funkcija gubitka. Iz pravila o derivaciji kompozicije znamo da vrijedi

$$\frac{\partial z}{\partial \operatorname{vec}(F)^T} = \frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})^T}\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial \operatorname{vec}(F)^T}, \tag{2.7}$$

odnosno, moramo izračunati samo $\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial \operatorname{vec}(F)^T}$. Izravno iz (2.5) slijedi

$$\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial \operatorname{vec}(F)^T} = \frac{\partial (I \otimes \phi(\boldsymbol{x}^l))\operatorname{vec}(F)}{\partial \operatorname{vec}(F)^T} = I \otimes \phi(\boldsymbol{x}^l).$$

Iz formule (2.7) transponiranjem dobivamo

$$\begin{aligned}\frac{\partial \operatorname{vec}(z)}{\partial \operatorname{vec}(F)} &= \left(I \otimes \phi(\boldsymbol{x}^l)\right)^T \frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})} \\ &= \left(I \otimes \phi(\boldsymbol{x}^l)^T\right) \operatorname{vec}\left(\frac{\partial z}{\partial Y}\right) \\ &= \operatorname{vec}\left(\phi(\boldsymbol{x}^l)^T \frac{\partial z}{\partial Y} I\right) \\ &= \operatorname{vec}\left(\phi(\boldsymbol{x}^l)^T \frac{\partial z}{\partial Y}\right).\end{aligned}$$

Naposlijetku zaključujemo da vrijedi

$$\frac{\partial z}{\partial F} = \phi(\boldsymbol{x}^{\boldsymbol{l}})^T \frac{\partial z}{\partial Y},$$

što je upravo drugi element uređene trojke iz definicije sloja.

Alternativni zapis operatora ϕ

Matricom $M \in \mathbb{R}^{(H^{l+1}W^{l+1}HWD^l)\times(H^lW^lD^l)}$ zapisat ćemo sve potrebne informacije o preslikavanju $\phi(\boldsymbol{x}^l)$. Ideja je spremiti informaciju o tome odakle je određeni element matrice $\phi(\boldsymbol{x}^l)$ originalno došao, odnosno, za indekse (p, q) odrediti indekse (i^l, j^l, d^l) takve da vrijedi $(\phi(\boldsymbol{x}^l))_{p,q} = \boldsymbol{x}^l_{i^l,j^l,d^l}$. Iz načina izgradnje operatora ϕ slijedi da takvo preslikavanje postoji. Primijetimo još da, zbog preklapanja blokova kod izgradnje $\phi(\boldsymbol{x}^l)$, preslikavanje iz (i^l, j^l, d^l) u (p, q) nije jedinstveno!

Neka je m funkcija koja preslikava (p, q) u (i^l, j^l, d^l) na gore navedeni način. Svaki red u matrici M označava jedan element matrice $\phi(\boldsymbol{x}^l)$, odnosno, jedan uređen par (p, q). Slično, svaki stupac matrice M označava jedan element od $\boldsymbol{x}^l$, odnosno, uređenu trojku (i^l, j^l, d^l). Primijetimo da se, zbog toga, za red x i stupac y mogu jedinstveno izračunati indeksi (p, q) i (i^l, j^l, d^l), respektivno. Elemente od M definiramo na sljedeći način

$$M_{x,y} = \begin{cases} 1, & \text{za } m(p,q) = (i^l, j^l, d^l), \\ 0, & \text{inače.} \end{cases}$$

Primijetimo da se, zapravo, radi o zapisu relacije m u obliku matrice. Matrica M definirana je tako da vrijedi

$$\operatorname{vec}(\phi(\boldsymbol{x}^l)) = M \operatorname{vec}(\boldsymbol{x}^l). \tag{2.8}$$

Propagiranje unazad

Sada s $X \in \mathbb{R}^{(H^l W^l) \times D^l}$ označavamo matricu dobivenu promjenom dimenzija iz $\boldsymbol{x}^l$ i obje oznake koristimo naizmjenično, po potrebi. Iz pravila o derivaciji kompozicije funkcija opet imamo

$$\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)^T} = \frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})^T} \frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial \operatorname{vec}(\boldsymbol{x}^l)^T}, \tag{2.9}$$

gdje preostaje izračunati $\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial \operatorname{vec}(\boldsymbol{x}^l)^T}$. Iz (2.6) i (2.8) slijedi

$$\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial \operatorname{vec}(\boldsymbol{x}^l)^T} = \frac{\partial (F^T \otimes I) \operatorname{vec}(\phi(\boldsymbol{x}^l))}{\partial \operatorname{vec}(\boldsymbol{x}^l)^T} = (F^T \otimes I) M,$$

odnosno, iz (2.9) imamo

$$\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)^T} = \frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})^T} (F^T \otimes I) M.$$

Odavde, zbog (2.3) i (2.4), vrijedi

$$\begin{aligned}
\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})^T}(F^T \otimes I) &= \left((F \otimes I) \frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})} \right)^T \\
&= \left((F \otimes I) \operatorname{vec}\left(\frac{\partial z}{\partial Y} \right) \right)^T \\
&= \operatorname{vec}\left(I \frac{\partial z}{\partial Y} F^T \right)^T \\
&= \operatorname{vec}\left(\frac{\partial z}{\partial Y} F^T \right)^T,
\end{aligned}$$

odakle slijedi

$$\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)^T} = \operatorname{vec}\left(\frac{\partial z}{\partial Y} F^T \right)^T M,$$

odnosno,

$$\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)} = M^T \operatorname{vec}\left(\frac{\partial z}{\partial Y} F^T \right). \tag{2.10}$$

Ovime je definiran i posljednji član uređene trojke sloja, čime je upotpunjena definicija konvolucijskog sloja.

Nešto detaljniji izvodi i metoda efikasnijeg izračuna izraza (2.10) mogu se pronaći u [32].

2.4 Sloj isključivanja

Za razliku od prethodnih slojeva, ovo je sloj koji se koristi kao sredstvo regularizacije, odnosno, poboljšanja generalizacije mreže. Prije definicije sloja potrebna je funkcija koja nasumično generira brojeve. Bez precizne definicije na vjerojatnosnom prostoru, neka je `random` *funkcija* koja za ulaznu vrijednost $p \in [0,1]$ generira realan broj x u istom segmentu te vraća 1 u slučaju $p \geq x$, a 0 u suprotnom.

Definicija 2.4.1. *Neka je $p \in [0,1]$.* ***Sloj isključivanja (eng. dropout)*** *definiramo po elementima, formulom*

$$(\texttt{dropout}_p(x))_i = \texttt{random}(p) \cdot x_i.$$

Ova tehnika u svakom koraku treniranja nasumično *isključuje* elemente izlaznog vektora prethodnog sloja i prosljeđuje nove vrijednosti sljedećem sloju u mreži. Nakon treniranja, sloj `dropout` koristi vrijednost $p = 1$, odnosno, ekvivalentan je identiteti.

Sloj nema parametara pa slijedi $\frac{\partial f}{\partial w} = 0$. Za posljednji element uređene trojke sloja imamo $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} y$, gdje vrijedi jednakost $y = \texttt{dropout}_p(x)$. Iz toga vidimo da elementi izbačeni u nekom koraku ostanu izbačeni i za potrebe propagiranja unazad u istom koraku. Drugi način gledanja na ulogu ovog sloja je dodavanje šuma na ulazne vrijednosti sljedećeg sloja u nizu. Više o motivaciji iz genetike, te konkretnim prednostima korištenja, može se pronaći u [27].

2.5 ℓ^2 regularizacija

Za razliku od sloja isključivanja, ovdje se ne radi o sloju, već samo o tehnici za regularizaciju, koja se može primijeniti na sve slojeve. Neka je S sloj i w tenzor njegovih parametara. Vrijednost $\alpha \cdot \|\text{vec}(w)\|_2^2$, za $\alpha > 0$, pribraja se funkciji gubitka. Ova metoda pomaže kod smanjenja prevelike prilagođenosti podacima iz trening skupa (eng. *overfitting*), pa samim time i generalizaciji na podatke izvan tog skupa [23].

Dio II

Primjena

Poglavlje 3

Primjene neuronskih mreža

3.1 Klasifikacija općenito

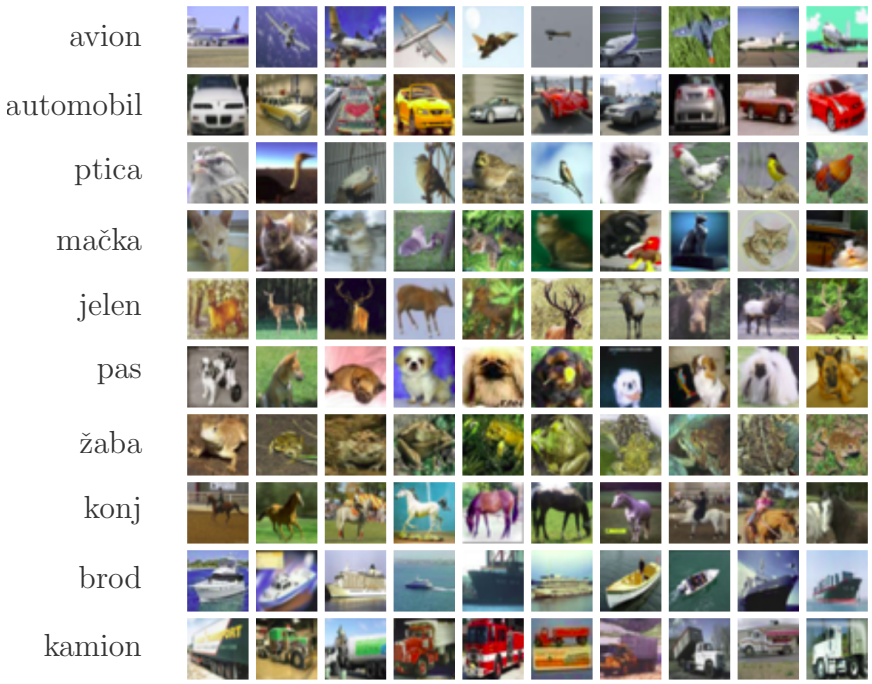
Prije pregleda raznih uporaba neuronskih mreža u obradi slika, promotrimo problem klasifikacije slika. Klasifikacija slika jedna je od najosnovnijih primjena neuronskih mreža u području slika. Problem se sastoji od određivanja kategorije slike, npr. imamo kategorije `pas` i `mačka`, a za ulaznu sliku želimo odrediti radi li se o slici psa ili mačke. Broj kategorija u tom slučaju iznosi 2, no skupovi podataka kao što su `ImageNet` zahtijevaju klasifikaciju na čak 1000 kategorija [5].

Iako se na prvi pogled ovaj problem čini jednostavan, radi se o izrazito teškom problemu. Percepcija težine iskrivljena je, jer ljudi imaju nevjerojatnu sposobnost prepoznavanja uzoraka koju ni ne primjećujemo. Za slučaj psa, taj se problem svodi na problem preciznog opisa slike psa, uzevši u obzir razne pasmine, kuteve slikanja, količinu svjetlosti i mnoge druge realne uvjete. Očito je da se radi o izrazito teškom problemu, no neuronske mreže pokazale su se kao nevjerojatno moćan alat pri njegovom rješavanju.

U sklopu ovoga rada dajemo implementaciju rješenja jednog problema klasifikacije, čiji precizniji opis slijedi u sljedećem poglavlju.

3.2 Skup podataka

`CIFAR-10` (eng. *Canadian Institute for Advanced Research*) [17] predstavlja skup slika iz 10 različitih kategorija. Radi se o skupu sastavljenom od 60000 slika u boji, dimenzija 32×32. Primjeri slika i nazivi kategorija prikazani su na slici 3.1.



Slika 3.1: Primjeri kategorija iz skupa `CIFAR-10`

Obrada podataka

U radu su prikazani rezultati iste arhitekture sa i bez dodatne obrade ulaznih podataka. Zbog toga što konvolucijske neuronske mreže nisu otporne na rotacije i zrcaljenja, takve i slične modifikacije originalnih slika znatno proširuju skup podataka te samim time omogućuju ostvarivanje boljih rezultata.

Za transformacije i modifikacije originalnih podataka korištena je Python biblioteka `imgaug` [14]. Iako se originalan skup podataka koristi kao osnova, na svaku epohu njegova korištenja modifikacije se iznova nasumično primijenjuju. U nastavku tek ugrubo navodimo korištene modifikacije, a više detalja i sama implementacija mogu se pronaći na [14].

Horizontalno zrcaljenje

Svaka ulazna slika zrcali se horizontalno (oko vertikalne osi) s vjerojatnošću 0.5.

Nasumično izrezivanje

Izrezivanje do maksimalno 10% širine/visine od svake slike.

Množenje

Množenjem skalarima iz intervala $[0.8, 1.2]$ po svim kanalima slike, ili u 20% slučajeva po svakom zasebno, dobije se zatamnjenje ili posvjetljenje određenih kanala boja.

Afine transformacije

Primjena skaliranja, translacija i rotacija.

Prijašnji rezultati

Maleno istraživanje Andreja Karpathya iz 2011. godine ukazuje na ljudsku grešku od 6% [15]. Iako se radi o klasifikaciji samo 400 slika iz `CIFAR-10` skupa, ovo daje barem okvirnu ideju o težini zadatka i nečistoći samih slika.

Cilj ovoga rada nije natjecanje s najboljim algoritmima u ovom zadatku, već samo primjer primjene te demonstracija potencijala konvolucijskih neuronskih mreža u klasifikaciji slika. Ipak, u nastavku navodimo najbolje rezultate u trenutku pisanja ovog rada.

Trenutno najbolji rezultat s greškom od samo 3.47% postignut je na Sveučilištu Warwick uvođenjem racionalnog (eng. *fractional*) `max-pool` sloja [10].

Suradnja tima s Kalifornijskog sveučilišta San Diego te Facebook AI istaživačkog tima uspjela je `ResNeXt` arhitekturom postići rezultat od 3.58% [33]. Rezidualni slojevi nisu u opsegu ovoga rada, no predstavljaju izrazito popularnu ideju te se moderne neuronske mreže za obradu slika najčešće sastoje od kombinacije rezidualnih i konvolucijskih slojeva.

3.3 Biblioteke

Implementacijska osnova projekta je programski jezik `python3` te računalna biblioteka `tensorflow` [1]. TensorFlow je Python biblioteka stvorena za implementaciju i izvršavanje algoritama strojnog učenja. Skalabilnost na brojne računalne arhitekture i fleksibilnost pri izražavanju algoritama čine ovu biblioteku izrazito pogodnom za implementaciju neuronskih mreža. Kompletan kôd programa dostupan je u git repozitoriju na adresi

`https://gitlab.com/tlevani/cifar-10`

3.4 Arhitektura modela

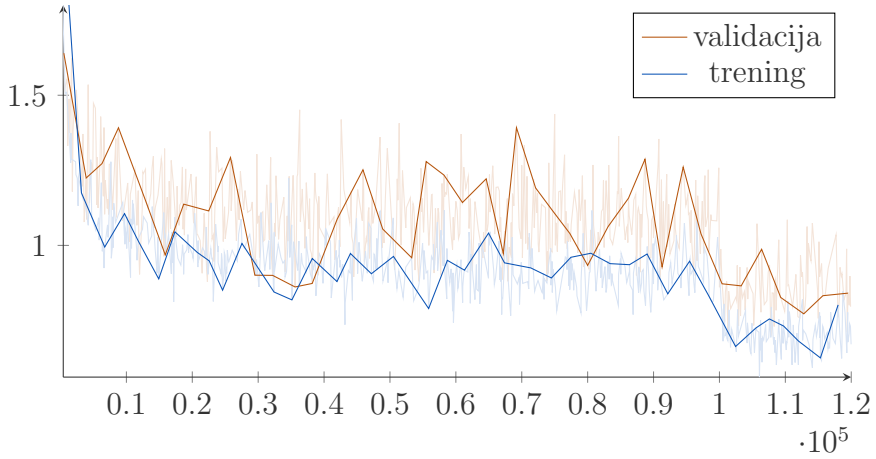
Arhitektura same mreže bazirana je na `All-CNN-C` arhitekturi iz [26]. Umjesto `ReLU`, kao u članku, ovdje za aktivacijske funkcije koristimo `LReLU`. Osim toga, posljednji dio mreže implementiran je potpuno povezanim slojevima, dok se u članku opet radi o konvolucijskim slojevima. Detalji modela dani su u tablici 3.1.

vrsta sloja	svojstva	ulaz	izlaz
ulaz		$32 \times 32 \times 3$	
`dropout`	$p = 0.8$		
konvolucija	$s = 1$; $(3 \times 3 \times 3) \times 96$	$32 \times 32 \times 3$	$32 \times 32 \times 96$
`LReLU`	$\alpha = 0.2$		
konvolucija	$s = 1$; $(3 \times 3 \times 96) \times 96$	$32 \times 32 \times 3$	$32 \times 32 \times 96$
`LReLU`	$\alpha = 0.2$		
konvolucija	$s = 2$; $(3 \times 3 \times 96) \times 96$	$32 \times 32 \times 3$	$16 \times 16 \times 96$
`LReLU`	$\alpha = 0.2$		
`dropout`	$p = 0.9$		
konvolucija	$s = 1$; $(3 \times 3 \times 192) \times 192$	$16 \times 16 \times 96$	$16 \times 16 \times 192$
`LReLU`	$\alpha = 0.2$		
konvolucija	$s = 1$; $(3 \times 3 \times 192) \times 192$	$16 \times 16 \times 192$	$16 \times 16 \times 192$
`LReLU`	$\alpha = 0.2$		
konvolucija	$s = 2$; $(3 \times 3 \times 192) \times 192$	$16 \times 16 \times 192$	$8 \times 8 \times 192$
`LReLU`	$\alpha = 0.2$		
`dropout`	$p = 0.9$		
vektorizacija		$8 \times 8 \times 192$	12288
potpuno povezan		12288	1024
`LReLU`	$\alpha = 0.2$		
`dropout`	$p = 0.5$		
potpuno povezan		1024	1024
`LReLU`	$\alpha = 0.2$		
`dropout`	$p = 0.5$		
potpuno povezan		1024	10
`softmax`			

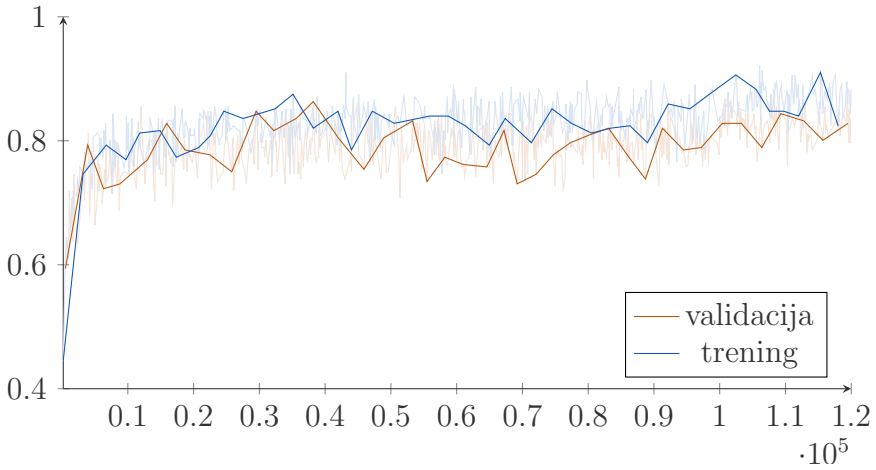
Tablica 3.1: Arhitektura neuronske mreže

3.5 Treniranje i rezultati

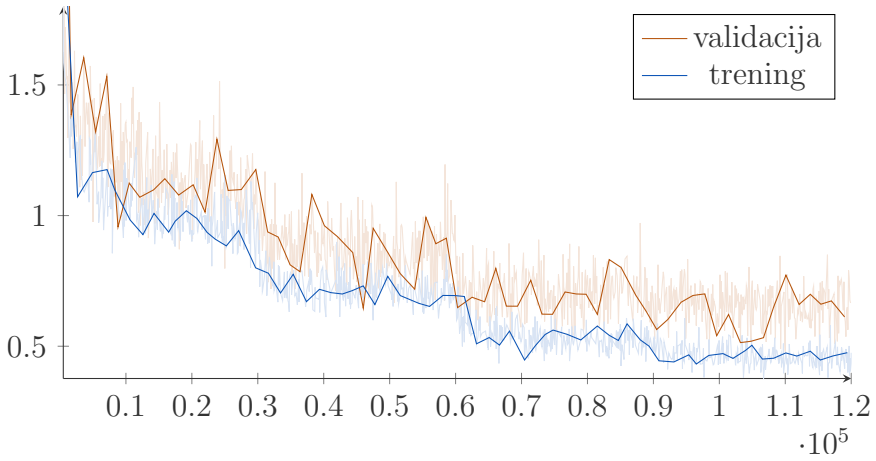
Brzina učenja bila je konstantna po dijelovima, odnosno, smanjivana je s vremenom treniranja. Svaka iteracija koristila je uzorak od 256 slika. Konačni rezultati su greške od 17.42% i 13.09% na testnim skupovima, za pokretanja bez dodatnog procesiranja slika i sa istim, respektivno. Slike 3.2 i 3.3 prikazuju vrijednosti funkcije gubitka i točnost po iteracijama, za treniranje bez dodatne obrade slika. Analogno, slike 3.4 i 3.5 prikazuju iste vrijednosti, za treniranje uz dodatnu obradu slika. Više detalja o implementaciji dostupno je u samom kôdu programa.



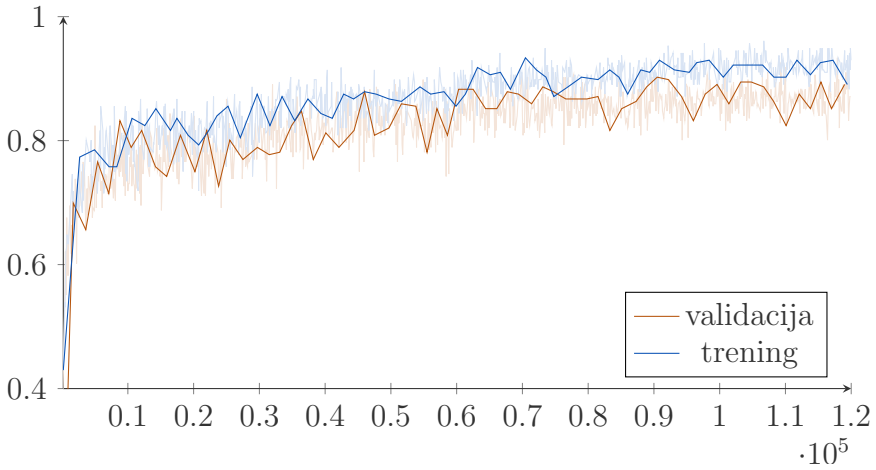
Slika 3.2: Funkcija gubitka bez obrade slika



Slika 3.3: Točnost (eng. *accuracy*) bez obrade slika



Slika 3.4: Funkcija gubitka uz obradu slika



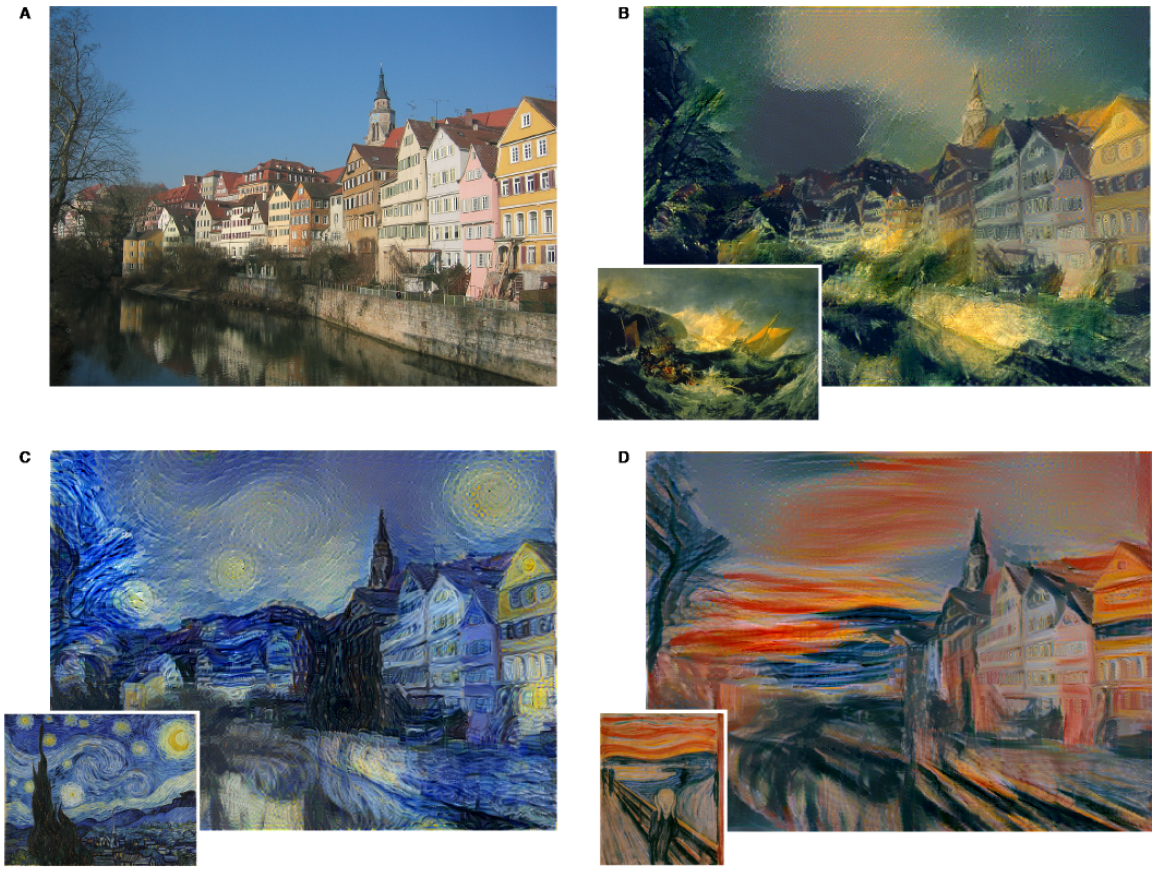
Slika 3.5: Točnost (eng. *accuracy*) uz obradu slika

3.6 Ostale primjene

Prijenos stila

Kao jednu od glavnih razlika između računala i ljudi često se navodi kreativnost, odnosno, nesposobnost računala za bavljenje umjetnošću. Baš zato su rezultati istraživanja u ovom smjeru izrazito interesantni. Radi se o zadatku prijenosa stila, odnosno, za ulaznu fotografiju A i stil B potrebno je generirati verziju fotografije A

u stilu B. Stil B, također, je zadan fotografijom. Primjeri prijenosa stila dani su na slici 3.6.



Slika 3.6: Primjer prijenosa stila. Slika `A` je originalna, a veće slike na primjerima `B`, `C` i `D` su primjeri originalne slike sa stilom manje slike u donjem lijevom kutu.

Rezultati sa slike 3.6 dobiveni su algoritmom iz [7]. Pojednostavljena verzija ideje je uzeti mrežu treniranu na nekoj klasifikaciji te pomoću skrivenih slojeva definirati svojstva ulaznih slika. Zatim, za ulaznu sliku A i stil, odnosno, ulaznu sliku B, usporedbom tih vrijednosti modificirati sliku A tako da se u njoj pojave uzorci iz slike B.

Valja napomenuti kako je ovo tek prvi u nizu rezultata koji pokazuju kako su neuronske mreže u stanju naučiti, i razdvojiti, sadržaj i stil slike. Prijenos stila samo je jedan od mnoštva primjena neuronskih mreža u smjeru klasične obrade fotografija.

Super-rezolucija

Super-rezolucija slike, odnosno, povećanje rezolucije slike, predstavlja zanimljiv problem često prikazivan u kriminalističkim serijama i filmovima. Iako takvi rezultati trenutno nisu mogući, postavlja se pitanje postoji li način generiranja iste slike većih dimenzija, bez očiglednog gubitka kvalitete.

Trenutno se za ovaj problem u praksi najčešće koriste numeričke interpolacije, kao što su bilinearna ili bikubična interpolacija, no ova istraživanja pokazuju kako korištenje neuronskih mreža ima velik potencijal.

Neka od ponuđenih rješenja sa zavidnim rezultatima su `PixelCNN` arhitektura tima *Google Brain* [4] te `SRGAN` mreža zaposlenika tvrtke *Twitter* [18]. Treniranje na specifičnim kategorijama, kao što su fotografije lica, daje vrhunske rezultate pri povećanju slika istih kategorija.

Prepoznavanje lica

Za razliku od obične klasifikacije, problem prepoznavanja lica podrazumijeva dinamičan broj kategorija te ga to čini znatno težim za rješavanje. Ono što današnja rješenja rade je generiranje n-dimenzionalnog opisnog vektora, takvog da je za slike lica istih osoba udaljenost opisnih vektora malena, dok je za različite osobe velika. Takav pristup omogućuje kategorizaciju opisnih vektora bržim algoritmima, kao što su k-sredine (eng. *k-means*) ili `SVM` (eng. *Support Vector Machine*).

Budući da se ne radi o običnoj klasifikaciji, treniranje se obavlja na nešto složeniji način. Jedan primjer je rješenje *Google* tima, koje za dvije različite slike iste osobe te jednu sliku neke druge osobe, funkciju gubitka definira kao razliku udaljenosti opisnog vektora različitih osoba te opisnog vektora istih osoba [24]. Na taj način postižu se željene karakteristike opisnih vektora. Alternativno rješenje, ponuđeno u [31], dano je standardnom klasifikacijom na fiksan broj klasa, gdje predzadnji sloj predstavlja opisni vektor. Osim `softmax` dijela funkcije gubitka dobivenog rezultatima klasifikacije, zadaje se i dodatan uvjet kojim se postiže zahtjev o udaljenostima opisnih vektora istih i različitih osoba. Za svaku od klasa definira se dodatni vektor, koji predstavlja centar svih lica iz te kategorije. U svakoj iteraciji ti centri približavaju se trenutnim procjenama, dok se udaljenost opisnih vektora od tih centara pridodaje funkciji gubitka.

Igrice

Igrice predstavljaju prirodno mjesto uporabe konvolucijskih neuronskih mreža, jer se zapravo radi o mnoštvu slika. S druge strane, trenutno stanje ovisi o vlastitim

akcijama u prethodnom potezu te mnoštvu drugih faktora, pa je ovo idealno područje za učenje podrškom.

Jedna metoda učenja podrškom je takozvano Q učenje (eng. *Q-learning*). Osnovna ideja sastoji se od predviđanja izlaza funkcije Q, čija vrijednost u nekom potezu predstavlja maksimalnu moguću dobit ostvarivu do kraja runde u tijeku. Ovime se ostvaruje odabir najprofitabilnijeg poteza u svakom koraku. Verziju Q učenja, pod nazivom duboko Q učenje, predstavljaju zaposlenici tvrtke *DeepMind* u članku [22]. Ono što je revolucionarno u ovom algoritmu jest činjenica da je isti algoritam savladao nekolicinu igrica igraće konzole *Atari* 2600. To je bio jedan od prvih primjera algoritma umjetne inteligencije koji nije apsolutno specijaliziran za jednu djelatnost.

Nakon što je *DeepBlue* algoritam tvrtke *IBM* još 1997. godine pobijedio tadašnjeg šahovskog svjetskog prvaka i jednog od najboljih, ako ne i najboljeg, igrača svih vremena, na red je došla drevna kineska igra *Go*. Sa znatno većim brojem mogućih stanja, ova igra predstavljala je pravi izazov. Tek 2016. godine, svjetski prvak Lee Sedol izgubio je 4 naprema 1 od `AlphaGo` algoritma tvrtke *DeepMind*. Ta verzija mreže trenirana je na potezima iz igara vrhunskih igrača. Već sljedeće godine, nakon nekoliko poboljšanja, izdana je i verzija `AlphaGo Zero` [25]. Način treniranja ove verzije je nevjerojatan, naime, mreža je igrala sama protiv sebe uz trivijalno predznanje poznavanja pravila igre. Već nakon 3 dana treniranja ova verzija prestigla je `AlphaGo` algoritam.

Još jedan primjer vrhunskih rezultata neuronskih mreža u području igrica je rezultat iz članka [30]. Radi se, opet, o pobjedi algoritma tvrtke *DeepMind* nad svjetskim prvakom, ovoga puta u računalnoj igri *StarCraft II*. Riječ je o izrazito složenoj strateškoj igrici koja, za razliku od prethodno navedenih, u svakom trenutku igre ne daje pristup cjelokupnom trenutnom stanju. Naime, sastavni dio igre jest i otkrivanje teritorija, odnosno, trenutnog stanja na nekom dijelu mape.

Bibliografija

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu i Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015, `https://www.tensorflow.org/`, Software available from tensorflow.org.

[2] Djork-Arné Clevert, Thomas Unterthiner i Sepp Hochreiter, *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*, CoRR **abs/1511.07289** (2015), `http://arxiv.org/abs/1511.07289`.

[3] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of Control, Signals and Systems **2** (1989), br. 4, 303–314, ISSN 1435-568X, `https://doi.org/10.1007/BF02551274`.

[4] R. Dahl, M. Norouzi i J. Shlens, *Pixel Recursive Super Resolution*, ArXiv e-prints (2017).

[5] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li i L. Fei-Fei, *ImageNet: A Large-Scale Hierarchical Image Database*, CVPR09, 2009.

[6] John Duchi, E Hazan i Y Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, The Journal of Machine Learning **12** (2011), 2121–2159.

[7] Leon A. Gatys, Alexander S. Ecker i Matthias Bethge, *A Neural Algorithm of Artistic Style*, CoRR **abs/1508.06576** (2015), `http://arxiv.org/abs/1508.06576`.

[8] Xavier Glorot, Antoine Bordes i Yoshua Bengio, *Deep Sparse Rectifier Neural Networks*, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (Fort Lauderdale, FL, USA) (Geoffrey Gordon, David Dunson i Miroslav Dudík, ur.), Proceedings of Machine Learning Research, sv. 15, PMLR, 11–13 Apr 2011, str. 315–323, http://proceedings.mlr.press/v15/glorot11a.html.

[9] Ian Goodfellow, Yoshua Bengio i Aaron Courville, *Deep Learning*, The MIT Press, 2016, ISBN 0262035618, 9780262035613.

[10] Benjamin Graham, *Fractional Max-Pooling*, CoRR **abs/1412.6071** (2014), http://arxiv.org/abs/1412.6071.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren i Jian Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, CoRR **abs/1502.01852** (2015), http://arxiv.org/abs/1502.01852.

[12] J. J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities*, Proceedings of the National Academy of Sciences **79** (1982), br. 8, 2554–2558, http://www.pnas.org/content/79/8/2554.abstract.

[13] J. J. Hopfield i D. W. Tank, *"Neural" computation of decisions in optimization problems*, Biological Cybernetics **52** (1985), br. 3, 141–152, ISSN 1432-0770, https://doi.org/10.1007/BF00339943.

[14] Alexander Jung, *Image augmentation for machine learning experiments*, srpanj 2015, https://github.com/aleju/imgaug, posjećeno 07. 01. 2018.

[15] Andrej Karpathy, *Lessons learned from manually classifying CIFAR-10*, travanj 2011, http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/, posjećeno 19. 10. 2017.

[16] Diederik P. Kingma i Jimmy Ba, *Adam: A Method for Stochastic Optimization*, CoRR **abs/1412.6980** (2014), http://arxiv.org/abs/1412.6980.

[17] Alex Krizhevsky, Vinod Nair i Geoffrey Hinton, *CIFAR-10 (Canadian Institute for Advanced Research)*, http://www.cs.toronto.edu/~kriz/cifar.html.

[18] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang i W. Shi, *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*, ArXiv e-prints (2016).

[19] Andrew L. Maas, Awni Y. Hannun i Andrew Y. Ng, *Rectifier Nonlinearities Improve Neural Network Acoustic Models*, 2013.

[20] J. E. Marsden i M. J. Hoffman, *Elementary Classical Analysis*, W. H. Freeman, 1993, ISBN 9780716721055, `https://books.google.hr/books?id=mMBY5jdjGfoC`.

[21] Warren S. McCulloch i Walter H. Pitts, *A Logical Calculus of the Ideas Immanent in Nervous Activity*, Bulletin of Mathematical Biophysics **5** (1943), 115–133.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra i Martin A. Riedmiller, *Playing Atari with Deep Reinforcement Learning*, CoRR **abs/1312.5602** (2013), `http://arxiv.org/abs/1312.5602`.

[23] P. Murugan i S. Durairaj, *Regularization and Optimization strategies in Deep Convolutional Neural Network*, ArXiv e-prints (2017).

[24] Florian Schroff, Dmitry Kalenichenko i James Philbin, *FaceNet: A Unified Embedding for Face Recognition and Clustering*, CoRR **abs/1503.03832** (2015), `http://arxiv.org/abs/1503.03832`.

[25] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel i Demis Hassabis, *Mastering the game of Go without human knowledge*, Nature **550** (2017), br. 7676, 354–359, ISSN 0028-0836, `https://doi.org/10.1038/nature24270`.

[26] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox i Martin A. Riedmiller, *Striving for Simplicity: The All Convolutional Net*, CoRR **abs/1412.6806** (2014), `http://arxiv.org/abs/1412.6806`.

[27] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever i Ruslan Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting.*, Journal of Machine Learning Research **15** (2014), br. 1, 1929–1958, `http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf`.

[28] T. Tieleman i G. Hinton, *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural Networks for Machine Learning, 2012.

[29] Andrea Vedaldi i Karel Lenc, *MatConvNet - Convolutional Neural Networks for MATLAB*, CoRR **abs/1412.4564** (2014), `http://arxiv.org/abs/1412.4564`.

[30] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp i Rodney Tsing, *StarCraft II: A New Challenge for Reinforcement Learning*, CoRR **abs/1708.04782** (2017), `http://arxiv.org/abs/1708.04782`.

[31] Yandong Wen, Kaipeng Zhang, Zhifeng Li i Yu Qiao, *A Discriminative Feature Learning Approach for Deep Face Recognition*, str. 499–515, Springer International Publishing, Cham, 2016, ISBN 978-3-319-46478-7, `https://doi.org/10.1007/978-3-319-46478-7_31`.

[32] Jianxin Wu, *Introduction to Convolutional Neural Networks*, svibanj 2017, `https://cs.nju.edu.cn/wujx/paper/CNN.pdf`, posjećeno 4. 10. 2017.

[33] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu i Kaiming He, *Aggregated Residual Transformations for Deep Neural Networks*, CoRR **abs/1611.05431** (2016), `http://arxiv.org/abs/1611.05431`.

[34] Neha Yadav, Anupam Yadav i Manoj Kumar, *History of Neural Networks*, str. 13–15, Springer Netherlands, Dordrecht, 2015, ISBN 978-94-017-9816-7, `https://doi.org/10.1007/978-94-017-9816-7_2`.

Sažetak

Konvolucijske neuronske mreže (`CNN`) predstavljaju popularan model neuronskih mreža, izrazito pogodan za obradu slika. U današnje vrijeme, gotovo sve *state-of-the-art* mreže za obradu slika baziraju se na ovoj arhitekturi.

Prvi dio ovoga rada definira i proučava matematički model konvolucijskih neuronskih mreža. Preciznije, definiraju se slojevi potrebni za konvolucijski model te algoritam propagiranja unazad.

U drugome dijelu daje se primjena te implementacija konvolucijskog modela mreže u obliku programa za klasifikaciju slika. Implementacija je izvedena unutar programskog jezika `Python` te se bazira na modulu `tensorflow`.

Summary

Convolutional neural networks (`CNN`) represent a popular neural network model, extremely suitable for image processing. At present time, almost every *state-of-the-art* neural network for image processing is based on this architecture.

The first part of this thesis defines and researches a mathematical model of convolutional neural networks. More precisely, we define various layers used inside a convolutional model and the *backpropagation* algorithm.

The second part deals with `CNN` implementation and application for image based classification. The programming language `Python` and its module `tensorflow` are used for the implementation.

Životopis

Rođen sam 22. kolovoza 1993. u Zagrebu i od tad živim u Hrastovcu, malom selu kraj Garešnice. Tamo sam pohađao Područnu školu Hrastovac, nakon koje završavam Osnovnu školu Garešnica. U Kutini upisujem Srednju školu Tina Ujevića, smjer matematička gimnazija. Poslije srednje škole upisujem preddiplomski studij Matematika na Matematičkom odsjeku Prirodoslovno–matematičkog fakulteta Sveučilišta u Zagrebu. Zbog toga što moji interesi leže u matematici i računarstvu, 2015. upisujem diplomski studij Računarstvo i matematika na istome fakultetu.