SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Ela Gračan

PREPOZNAVANJE UZORAKA POMOĆU NEURONSKIH MREŽA

Diplomski rad

Voditelj rada:
Doc. dr. sc. Nela Bosner

Zagreb, Veljača, 2020.

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 majci koja mi je, osim razumijevanja i podrške, pružila pomoć bez koje bi moj studij zasigurno još duže trajao. Zahvaljujem i svojoj mentorici na prihvaćanju mojih ideja, pomoći pri izradi ovog rada, kao i na svim korisnim savjetima koje mi je dala.

Sadržaj

Uvod

Čovjekova je sposobnost paralelne obrade informacija i obavljanja raznih funkcija, kao što su primjerice prepoznavanje lica i razumijevanje prirodnog jezika, rezultat složene strukture neurona u ljudskom mozgu. Biološka neuronska mreža sastoji se od otprilike 10^{11} neurona međusobno povezanih sinapsama. Dio neuronske strukture je urođen, dok se ostatak razvija tijekom života. Te se promjene očituju u osnaživanju, odnosno slabljenju nekih sinaptičkih veza. Upravo je biološka građa neurona bila inspiracija za konstrukciju umjetnih neuronskih mreža.

Umjetne neuronske mreže apstrakcija su bioloških i puno su jednostavnije građe. Ostvarene su u najvećoj mjeri u obliku programskog kôda te nemaju sposobnosti ni približne ljudskom mozgu. Unatoč tomu, one u posljednje vrijeme nalaze primjenu u sve više djelatnosti. Neuronske se mreže tako mogu koristiti u zrakoplovstvu, bankarstvu, medicini i robotici te mnogim drugim područjima u kojima pomažu ili samostalno obavljaju razne analize, procjene ili predviđanja.

U ovom ćemo radu dati pregled osnovnih pojmova vezanih za umjetne neuronske mreže. Opisat ćemo matematički model neurona te proučiti različite načine njihova povezivanja u mrežu. Razradit ćemo neke od metoda i algoritama koji se koriste u učenju umjetnih neuronskih mreža. Na kraju ćemo odabrati neku od opisanih struktura i primijeniti ju na problem prepoznavanja uzoraka.

Poglavlje 1

Motivacija i primjena neuronskih mreža

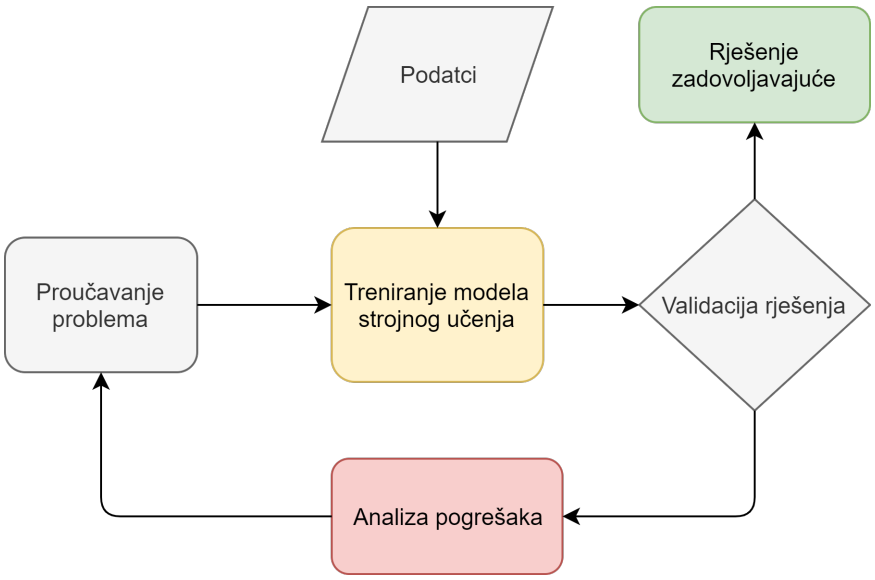
1.1 Neuronske mreže kao grana strojnog učenja

Pojam strojnog učenja i umjetne inteligencije često budi ljudsku maštu te svi odmah pomisle na robote koji imaju vlastitu volju ili na slične filmske interpretacije. **Umjetnom inteligencijom** smatramo svaki nebiološki sustav koji ima sposobnost rješavanja dosad neviđenog problema [14]. **Strojno učenje** grana je umjetne inteligencije koja se bavi razvojem metoda i algoritama pomoću kojih računalo nizom iteracija (učenjem) poboljšava obavljanje nekog zadatka. Svim je područjima strojnog učenja zajednička velika količina podataka i nepostojanje eksplicitnog algoritma za rješavanje problema nad tim podatcima. Jedan od prvih sustava strojnog učenja koji je ušao u svakodnevnu upotrebu i svakodnevno ga koriste milijuni ljudi je *spam* filter, odnosno sustav koji na temelju sadržaja poruke elektroničke pošte istu razvrstava u neželjenu, odnosno željenu poštu (engl. *spam or ham*) [5]. Područje strojnog učenja koje se oslanja na neuronske mreže naziva se **duboko učenje** te su ta dva pojma usko povezana i često se koriste naizmjenično.

Slika 1.1: Neuronske mreže grana su strojnog učenja

Uobičajeno je da se neki projekt u strojnom učenju odvija u sljedećim koracima [5]:

- analiza podataka
- izbor modela za učenje
- učenje na podatcima
- testiranje i validacija
- primjena modela na nepoznatim podatcima.



Slika 1.2: Pristup problemu strojnog učenja

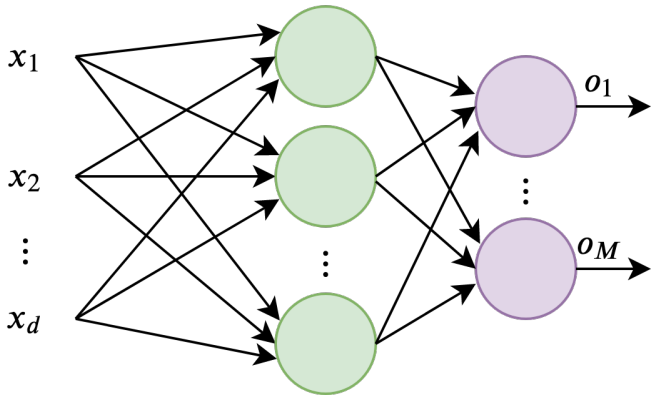
Najvažniji je od gornjih koraka učenje na podatcima, te je to korak koji zahtjeva najviše vremena i računalne snage. U toj fazi učenja očekujemo samostalnu prilagodbu parametara neuronske mreže s ciljem da mreža što točnije predvidi neku dosad neviđenu instancu danog problema.

1.2 Osnovni pojmovi

Osnovni građevni element neuronske mreže, prema njezinom biološkom uzoru, nazivamo **neuron**. U literaturi se često koristi i naziv jedinica (engl. *unit*) [12]. Neurone spajaju veze koje imaju svoje **težine**, što poistovjećujemo sa sinapsama i njihovom snagom. Ako je ulazni signal neurona dovoljno jak, neuron će proslijediti signal sljedećim neuronima. Vrijednost koju neuron treba dostići da bi proslijedio signal nazivamo **prag** (engl. *bias*).

Upravo su težine veza među neuronima i prag oni parametri mreže koji se prilagođavaju tijekom faze učenja. Iz ove osnovne ideje razvili su se razni modeli neurona koji se uglavnom razlikuju prema tome na koji način prosljeđuju signale. Neurone grupiramo u **slojeve**, a svi slojevi čine neuronsku mrežu. Ulazni se signali ponekad nazivaju **ulaznim slojem**, međutim on se ne sastoji od neurona. Osim ulaznog sloja, mreža mora imati i **izlazni sloj**. Reći ćemo da se radi o jednoslojnoj neuronskoj mreži ako se ona sastoji samo od ulaznog sloja (ulaznih podataka) i izlaznog sloja neurona. Kada mreža između ulaznog i izlaznog sloja ima jedan ili više slojeva neurona, te slojeve nazivamo **skrivenim slojevima**. Neuroni u istom sloju nisu međusobno povezani, već je svaki neuron sloja povezan s nekim neuronima u sljedećem sloju. Slika 1.3 prikazuje način povezivanja neurona u mrežu. Gore opisana mreža naziva se **aciklička**, najjednostavnije je građe te je osnova za sve druge arhitekture.

Uočimo da je broj neurona u izlaznom sloju, kao i broj ulaza mreže, određen prirodom problema. Primjerice, ako želimo ulazne parametre podijeliti u 10 klasa, tada će izlazni sloj takve mreže sadržavati 10 neurona. Broj neurona u skrivenim slojevima teže je odrediti te se on razlikuje od problema do problema.



Slika 1.3: Neuroni povezani u slojeve

Jedan se ulazni podatak može sastojati od više signala koji se nazivaju i **značajke** (engl. *feature*), dok ciljni podatak nazivamo **oznaka** (engl. *label*). Uobičajeno je da se ulazni podatci dijele na **skup za učenje**, **skup za validaciju** i **skup za testiranje** kako bi se osigurao ispravan rad mreže na dosad neviđenim podatcima. Ako ne osiguramo pravilnu podjelu ovih podataka, može doći do problema **pretreniranosti**. To je pojava u kojoj se mreža u potpunosti prilagodi podatcima za treniranje i gubi mogućnost prilagodbe novim instancama problema (engl. *overfitting*). Skup podataka za testiranje koristi se samo za evaluaciju mreže, te se ti podaci ne smiju koristiti u algoritmima za učenje.

Pri konstrukciji neuronske mreže postoje razni **hiperparametri** koje možemo prilago-

đavati. Hiperparametri su vrijednosti koje se kroz samo učenje ne mijenjaju, ali utječu na efikasnost učenja mreže, poput primjerice broja slojeva. Usredotočimo li se na jedan hiperparametar, prirodan bi način bio da pokušamo trenirati mrežu nekoliko puta, svaki puta s različitom vrijednošću odabranog hiperparametra i odlučimo se za onaj koji daje najbolje rezultate na skupu za testiranje. Međutim, kada neuronsku mrežu stavimo u uporabu, može se ispostaviti da su joj performanse značajno lošije od onih na skupu za testiranje. Razlog tome je što smo ponovljenim treniranjem i testiranjem mrežu implicitno prilagodili i skupu podataka za testiranje, pa mreža u stvarnosti nema sposobnost prilagođavanja novim podatcima.

Rješenje ovog problema je dodatni skup podataka koji nazivamo skup za validaciju. Nakon što faza učenja završi i kada mreža postiže zadovoljavajuće rezultate na skupu podataka za validaciju, tada na podacima za testiranje možemo provjeriti je li spremna za stvarnu uporabu.

1.3 Primjena

U posljednjih je nekoliko godina interes za umjetne neuronske mreže ponovno porastao, prvenstveno zbog velikog razvoja računalne moći, što je dovelo do porasta kapaciteta neuronskih mreža i značajnog napretka u njihovu razvoju.

Neuronske mreže koriste se u raznim područjima kojima je zajednička velika količina podataka. Neki od najslikovitijih primjera dolaze iz područja kao što su obrada slike i prepoznavanje govora te raznih grana medicine i ekonomije.

Primjerice, neuronske se mreže koriste u detekciji kartičnih prijevara. Naime, u doba internetske kupovine gdje kupac nije fizički prisutan, nemoguće je provjeriti svaku transakciju te je takav sustav podložan raznim ilegalnim manipulacijama. Neuronske su mreže u mogućnosti uočiti neke sličnosti u nelegalnim transakcijama te ih označiti ili spriječiti [8].

Nadalje, neuronske mreže ugrađene u sustave s optičkim čitačima koriste se za prepoznavanje slova, znamenki i drugih znakova te se koriste za pretvaranje rukom pisanih, skeniranih ili tiskanih dokumenata u tekst u ASCII ili Unicode formatu [13].

U medicini se neuronske mreže mogu koristi primjerice za detekciju ili potvrdu srčanog udara iz elektrokardiograma ili za otkrivanje demencije iz elektroencefalograma pacijenata.

Neuronske se mreže koriste i u obradi govora gdje mogu prepoznati jezik kojim govorimo pa čak i detektirati emociju kao što je ljutnja ili humor.

Poglavlje 2

Model neurona i arhitekture mreže

2.1 Matematički model neurona

Građa umjetnog neurona

Ulazni podatci neurona mogu biti vektori, matrice ili čak tenzori, međutim radi jednostavnosti pretpostavimo da su vektori iz $\mathbb{R}^d$. Točnije, neka je $d \in \mathbb{N}$ dimenzija pojedinog ulaznog podatka neurona te $N \in \mathbb{N}$ ukupan broj ulaznih podataka. Označimo ulazne podatke s $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ pri čemu je

$$\mathbf{x}_k = [x_{k1}, x_{k2}, \ldots, x_{kd}], \qquad k = 1, 2, \ldots, N. \tag{2.1}$$

Sve ulazne podatke možemo promatrati kao matricu:

$$X = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1d} \\ x_{21} & x_{22} & \ldots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \ldots & x_{Nd} \end{bmatrix}. \tag{2.2}$$

Neka je sada $\mathbf{x} = [x_1, x_2, \ldots, x_d]$ jedan ulazni podatak, odnosno jedan redak matrice X koja se često naziva **matrica dizajna** [4]. Matematički model neurona ima d ulaznih signala te d odgovarajućih težina tih ulaznih signala $w_1, w_2, \ldots, w_d$. Ulaznim signalima neurona smatramo komponente vektora $\mathbf{x}$ koje smo nazvali značajke. Dakle, svakom od ulaznih signala neurona $x_i \in \mathbb{R}$ pridružena je težinska vrijednost $w_i \in \mathbb{R}, i = 1, 2, \ldots, d$. Neuron može imati i prag $b \in \mathbb{R}$ koji u jednostavnom modelu predstavlja kriterij za slanje signala. Osim ulaza, težina i praga, neuron ima ugrađenu i **funkciju prijelaza** odnosno

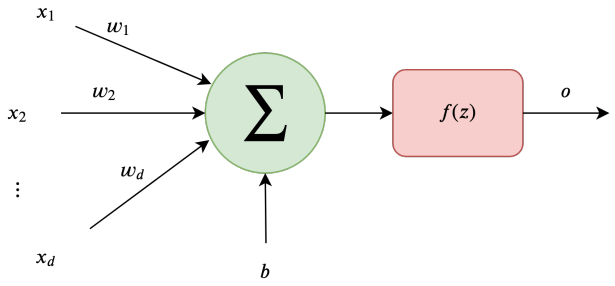
aktivacijsku funkciju. Kao ulazni parametar, funkcija prijelaza prima težinsku sumu z koja se često naziva **aktivacija neurona**.

$$z = \sum_{i=1}^{d} x_i w_i + b \tag{2.3}$$

Isti izraz zapisan u matričnom obliku glasi $z = \mathbf{x}W + b$, pri čemu je

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}. \tag{2.4}$$

Neuron s ulazom $\mathbf{x}$, težinama W i funkcijom prijelaza f grafički je prikazan na slici 2.1



Slika 2.1: Model neurona

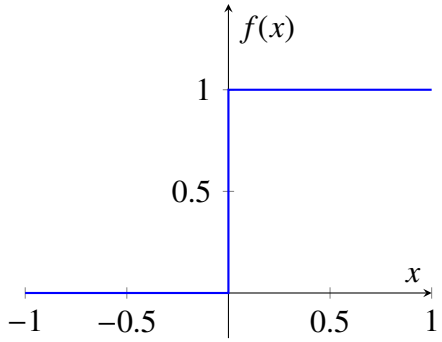
Izlaz neurona je skalar o kojeg računamo kao $o = f(z) = f(\mathbf{x}W + b)$, gdje je f neka određena funkcija prijelaza [6].

Funkcije prijelaza

Najjednostavnija funkcija prijelaza je identiteta $f(x) = x$, odnosno izlaz neurona dan je s $o = \mathbf{x}W + b$. Model neurona koji koristi ovu funkciju naziva se ADALINE (engl. *ADAptive LINear Element*).

Sljedeća funkcija prijelaza je step funkcija (engl. *hard limit transfer function*) čiji je izlaz 0 ili 1 te time dobivamo model neurona koji nazivamo **TLU** (engl. *Threshold Logic Unit*). Taj se model naziva i **perceptron** te se može koristiti za jednostavne zadatke klasifikacije gdje su podatci linearno odvojivi. Perceptron je jedan od najranije modeliranih neurona i iz njega su nastale razne druge poboljšane inačice [3].

$$f(x) = \begin{cases} 1 & \text{za } x \geq 0, \\ 0 & \text{inače.} \end{cases} \tag{2.5}$$



Slika 2.2: Step funkcija

Funkcija koja ima sposobnost nelinearne klasifikacije je pozitivna linearna funkcija.

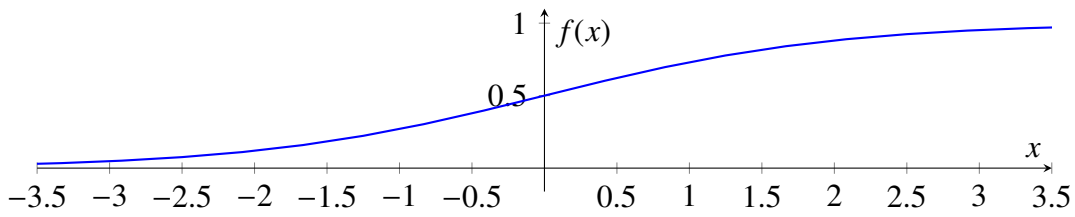
$$f(x) = max(0, x) \tag{2.6}$$

Neuron koji koristi ovu funkciju kao aktivacijsku funkciju naziva se **ReLU** (engl. *Rectified Linear Unit*). Često se koristi i njezina derivabilna verzija $f(x) = \log(1 + e^x)$.

Nadalje, kao aktivacijska funkcija može se koristiti sigmoidalna odnosno **logistička** funkcija (engl. *logistic sigmoid function*):

$$f(x) = \frac{1}{1 + e^{-cx}}\,. \tag{2.7}$$

Ona je svojstvena po tome što za svaki $x \in \mathbb{R}$ vrijedi $f(x) \in [0, 1]$ te je diferencijabilna pa je pogodna za optimizacijske metode koje koriste gradijent.



Slika 2.3: Logistička funkcija

Gornje funkcije u obzir uzimaju samo aktivaciju neurona na kojem djeluju. Navedimo i najvažniju aktivacijsku funkciju **softmax** koja pri izračunu izlaza neurona uzima u obzir aktivaciju svih neurona istog sloja. Neka je $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, gdje x_i predstavlja aktivaciju i-tog neurona u nekom sloju od n neurona. Tada je *softmax* funkcija dana s:

$$f(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}. \tag{2.8}$$

Upravo je ova funkcija najčešće korištena u problemima klasifikacije jer bilo koji vektor pretvara u distribuciju vjerojatnosti. Zbog toga se najčešće koristi u izlaznom sloju neurona gdje izlazne vrijednosti neurona označavaju vjerojatnost pripadnosti klase.

Postoje i razne druge funkcije koje se mogu koristiti kao prijelazne funkcije u modelu neurona. Funkcije se prijelaza primjenjuju na aktivaciju neurona z (2.3). Tablica 2.1 daje prikaz najčešćih funkcija prijelaza.

Naziv funkcije	$f(x)$
identiteta	x
step	$\begin{cases} 1 & \text{za } x \geq 0 \\ 0 & \text{inače} \end{cases}$
logistička	$\frac{1}{1 + e^{-cx}}$
tangens hiperbolni	$\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
pozitivna linearna	$max(0, x)$

Tablica 2.1: Najčešće funkcije prijelaza

2.2 Arhitektura neuronskih mreža

Neuronska mreža najjednostavnije arhitekture je **jednoslojna acikličk mreža** (engl. *one layer feedforward network*). Ona sadrži samo jedan sloj neurona, dakle ulazni signali povezani su sa slojem neurona koji je ujedno i izlazni sloj mreže. Nema povratnih veza između neurona, odnosno sve su veze usmjerene od ulaznih signala prema izlaznom sloju. Pritom neuroni istog sloja nisu međusobno povezani [3]. Najistaknutiji su primjeri ove arhitekture mreže ADALINE (funkcija prijelaza je identiteta) i Perceptron (funkcija prijelaza je step funkcija).

Višeslojne (duboke) acikličke mreže promatramo kao niz slojeva neurona. Svaki je neuron povezan usmjerenom vezom s neuronima u sljedećem sloju. Kažemo da je mreža **potpuno povezana** kada je svaki od neurona u sloju povezan sa svakim neuronom u sljedećem sloju te takav sloj nazivamo **gustim slojem** (engl. *dense layer*). U suprotnom kažemo da je mreža **djelomično povezana** [7].

Treća skupina arhitektura su **mreže s povratnim vezama** (engl. *recurrent networks*). To su mreže u kojima postoji barem jedan **ciklus**, odnosno postoji barem jedan neuron koji je povezan s neuronom u prethodnom sloju. Neke od poznatih mreža ove arhitekture su Hammingova mreža koja ima jedan sloj neurona spojen acikličkim vezama i jedan sloj s povratnim vezama, te Hoppfieldova mreža koja se sastoji samo od sloja s povratnim vezama [6]. Promotrimo razliku u mogućnostima između jednoslojne acikličke mreže (perceptrona) i višeslojnih acikličkih mreža.

Perceptron

Perceptronom nazivamo jednoslojnu mrežu neurona koji kao aktivacijsku funkciju koriste step funkciju, dakle pojedini neuron kao svoj izlaz daje samo 0 ili 1 (TLU neuroni). Perceptron je jedna od najjednostavnijih neuronskih mreža koju je 1957. godine osmislio Frank Rosenblatt [6]. Za ilustraciju konstruirajmo perceptron koji predstavlja logički sklop *ILI*. Kako se radi o jednostavnom perceptronu, ulazni podatci mogu biti $(0,0), (0,1), (1,0), (1,1)$. Nije teško pogoditi da je dovoljno da je matrica $W = [1,1]^T$, te da je $b = 0$.

Ulazni podatak	Aktivacija	Izlaz
(0,0)	0	0
(0,1)	1	1
(1,0)	1	1
(1,1)	2	1

Tablica 2.2: Tablica vrijednosti aktivacija i izlaza za sklop *ILI*

Ako želimo dobiti sklop I, dovoljno je promijeniti $b = -1$ da bismo dobili ispravno rješenje.

Ulazni podatak	Aktivacija	Izlaz
(0,0)	-1	0
(0,1)	0	0
(1,0)	0	0
(1,1)	1	1

Tablica 2.3: Tablica vrijednosti aktivacija i izlaza za sklop I

Međutim, perceptronom ne možemo konstruirati sklop isključivo *ILI* (*XOR*). Već kod takvog jednostavnog zadatka jednoslojna mreža nailazi na probleme te je jasno da su nam potrebni kompleksniji alati.

Višeslojne neuronske mreže

Ove su mreže osnovni gradivni element za mnoge koncepte u dubokom učenju te ćemo se u sljedećem poglavlju najviše usredotočiti na algoritme vezane uz njih. Neka je S ukupan broj slojeva neurona u mreži koji se naziva i **dubina modela**, od kuda potječe pojam dubokog učenja.

Promotrimo način računanja u pojedinom sloju neurona $l = 1, 2, \ldots, S$. Neka je $r_l \in \mathbb{N}$ broj neurona u sloju, te neka je $r_{l-1} \in \mathbb{N}$ broj neurona iz prethodnog sloja, odnosno dimenzija ulaznog podatka d, ako promatramo prvi sloj. Sa o^{l-1} označit ćemo izlaz prethodnog sloja neurona.

$$o^{l-1} = [o_1^{l-1}, o_2^{l-1}, \ldots, o_{r_{l-1}}^{l-1}] \tag{2.9}$$

U slučaju kada je $l = 1$ imamo $o^0 = \mathbf{x} = [x_1, x_2, \ldots, x_d]$. Matrica težina l-tog sloja, $W_l \in M_{r_{l-1} r_l}(\mathbb{R})$ dana je s

$$W_l = \begin{bmatrix} w_{11}^l & w_{12}^l & \ldots & w_{1r_l}^l \\ w_{21}^l & w_{22}^l & \ldots & w_{2r_l}^l \\ \vdots & \vdots & \ddots & \vdots \\ w_{r_{l-1}1}^l & w_{r_{l-1}2}^l & \ldots & w_{r_{l-1}r_l}^l \end{bmatrix} \tag{2.10}$$

Vektor praga b^l dan je s

$$b^l = [b_1^l, b_2^l, \ldots, b_{r_l}^l] \tag{2.11}$$

Sa z^l označavat ćemo vektor težinskih suma l-tog sloja (aktivaciju l-tog sloja) $z^l = [z_1^l, z_2^l, \ldots, z_{r_l}^l]$, pri čemu je

$$z_j^l = \sum_{i=1}^{r_{l-1}} w_{ij}^l \, o_i^{l-1} + b_j^l. \tag{2.12}$$

Matrično, vektor b^l možemo dodati kao zadnji redak matrice W_l te izlaz prethodnog sloja proširiti jedinicom:

$$W_l' = \begin{bmatrix} W_l \\ b^l \end{bmatrix}, \qquad \tilde{o}^{\,l-1} = \begin{bmatrix} o^{l-1} & 1 \end{bmatrix}. \tag{2.13}$$

Tada je ekvivalentno

$$z^l = \tilde{o}^{\,l-1} W_l'. \tag{2.14}$$

Za zadanu aktivacijsku funkciju l-tog sloja f_l, izlaz sloja o^l dan je s

$$o^l = [o_1^l, o_2^l, \ldots, o_{r_l}^l] = [\, f_l(z_1^l), f_l(z_2^l), \ldots, f_l(z_{r_l}^l) \,]. \tag{2.15}$$

Konstruirajmo sada mrežu s jednim skrivenim slojem koja računa isključivo *ILI*. Matrica dizajna dana je s

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{2.16}$$

Ciljni je izlaz y dan s

$$y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \tag{2.17}$$

Skriveni sloj imat će dva neurona $r_1 = 2$ te će koristiti pozitivnu linearnu aktivacijsku funkciju $f_1(x) = max(0, x)$. Izlazni sloj imat će jedan neuron te će njegov izlaz biti jednak aktivaciji, odnosno aktivacijska funkcija bit će identiteta.

Neka je

$$W_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad b_1 = \begin{bmatrix} 0 & -1 \end{bmatrix}. \tag{2.18}$$

Aktivacija i izlaz prvog sloja $z_1 = XW_1 + b_1$ i $o_1 = f_1(z_1)$ dani su s

$$z_1 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}, \qquad o_1 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \tag{2.19}$$

Neka je sada

$$W_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \qquad b_2 = 0. \tag{2.20}$$

Dobivamo izlaz $o = o_2 = z_2 = o_1 W_2 + b_2$

$$o = y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \tag{2.21}$$

Ulazni podatak	Aktivacija prvog sloja	Izlaz prvog sloja	Aktivacija drugog sloja	Izlaz drugog sloja
(0,0)	(0, -1)	(0,0)	0	0
(0,1)	(1, 0)	(1,0)	1	1
(1,0)	(1, 0)	(1,0)	1	1
(1,1)	(2, 1)	(2,1)	0	0

Tablica 2.4: Tablica vrijednosti aktivacija i izlaza za sklop *isključivo ILI*

Konvolucijske neuronske mreže

Konvolucijske neuronske mreže nadogradnja su višeslojnih neuronskih mreža nastale u nastojanju da se poprave performanse kod obrade složenijih ulaznih signala velike dimenzionalnosti kao što su slike i zvuk.

Višeslojnu neuronsku mrežu nazivamo konvolucijskom ako uz određen broj potpuno povezanih slojeva sadrži barem jedan **konvolucijski sloj**. Tipično se konvolucijski sloj koristi kako bismo iz ulaza, primjerice slike, dobili bolji prikaz skrivenih reprezentacija koji se naziva i **mapa značajki**. Htjeli bismo da je naš model neuronske mreže invarijantan na translacije objekata na slici, kao i da ima mogućnost uočavanja lokalnih povezanosti na slici.

Zamislimo da ulazni podatak više nije vektor, već crno-bijela slika, odnosno matrica $x \in M^{mn}(\mathbb{R})$. Umjesto matrice težina konvolucijski sloj koristi matricu $\omega \in M^{pr}(\mathbb{R})$ koju nazivamo **filter** ili **jezgra** (engl. *kernel*).

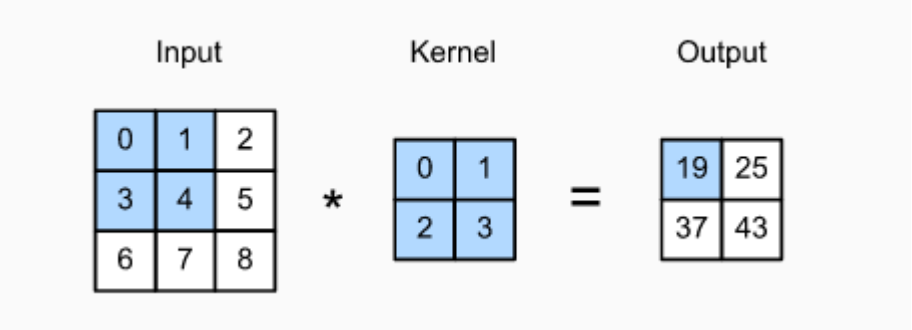
Dvodimenzionalna mapa značajki h računa se na sljedeći način:

$$h = \omega * x, \tag{2.22}$$

pri čemu $*$ označava operaciju konvolucije te vrijedi:

$$h(i, j) = (\omega * x)(i, j) = \sum_{k=0}^{p-1} \sum_{l=0}^{r-1} x(i + k, j + l)\, \omega(k, l). \tag{2.23}$$

Formula koja se zapravo koristi naziva se unakrsna korelacija, međutim zbog sličnosti s formulom konvolucije, mreža nosi takav naziv [15]. Na slici 2.4 vidimo primjer računanja jednog elementa mape značajki.



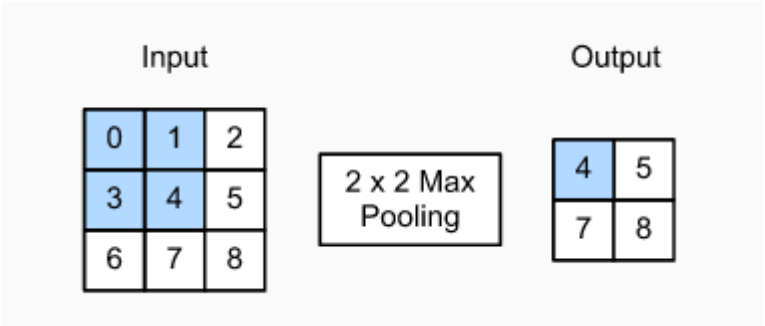
Slika 2.4: Primjer izračuna konvolucije [15]

Prednost konvolucijskog sloja u odnosu na potpuno povezan sloj očituje se u tome što su veze između neurona puno rjeđe zbog naglaska na lokalnu povezanost pa je samim time smanjen broj težina veza koje se tijekom učenja prilagođavaju. Ukoliko je potrebno na dobivenu se mapu značajki također može primijeniti aktivacijska funkcija.

Važan pojam povezan uz konvolucijski sloj je **pomak** (engl. *stride*). Kada primjenjujemo formulu 2.23 na matricu x počinjemo od gornjeg lijevog elementa matrice x te se

pomičemo desno i dolje dok ne prođemo sve dijelove matrice. Pomak određuje za koliko elemenata matrice se želimo pomicati. Može biti različit za horizontalni i vertikalni smjer.

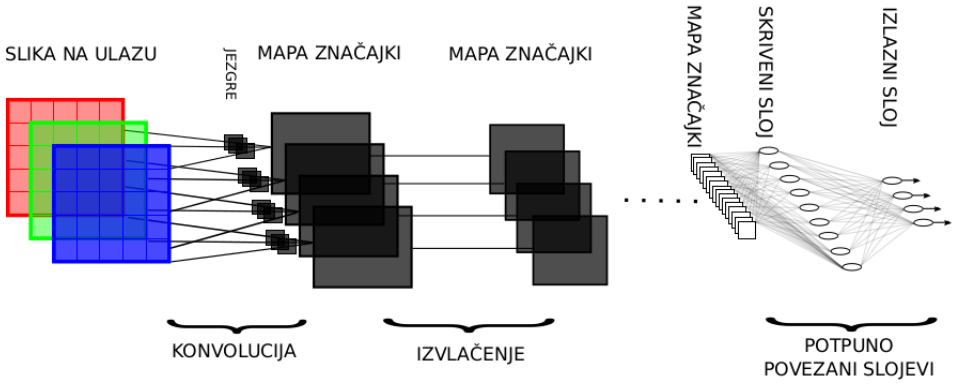
Uz konvolucijski sloj često se nalazi **sloj sažimanja**. Taj sloj služi za smanjenje rezolucije mapa značajki dobivenih u konvolucijskom sloju korištenjem **funkcije sažimanja**. S obzirom na funkciju sažimanja, razlikujemo sažimanje maksimalnom vrijednošću (engl. *max-pooling*) te sažimanje srednjom vrijednošću (engl. *average pooling*) [16]. Funkcija sažimanja djeluje na bliske vrijednosti mape značajki i pretvara ih u jednu vrijednost.



Slika 2.5: Primjer pretvorbe mape značajki sažimanjem maksimalnom vrijednošću [15]

Kao i u konvolucijskom sloju, za sloj sažimanja također definiramo pomak. Uz to potrebno je definirati i **okvir sažimanja**, odnosno odrediti dimenziju podmatrice na koju se primjenjuje funkcija sažimanja. Na slici 2.5 vidimo primjer sažimanja maksimalnom vrijednošću s pomakom 1 te okvirom sažimanja 2×2.

Slika 2.6 prikazuje uobičajeni način konstrukcije konvolucijske neuronske mreže.



Slika 2.6: Uobičajena struktura konvolucijske neuronske mreže [16]

2.3 Učenje mreže

Do sada smo razmotrili razne načine računanja izlaznih parametara pod uvjetom da su nam poznate sve težine veza i pragovi između slojeva neurona. Sada ćemo promotriti pravila i algoritme za njihovo prilagođavanje što se često naziva treniranje, odnosno **učenje mreže**. Razlikujemo tri načina učenja:

- **Učenje pod nadzorom**
  Algoritmu za učenje predaju se parovi (*ulaz*, *izlaz*) koji se nazivaju skup podataka za učenje. Učenje se odvija tako da se trenutni izlaz mreže za dani ulaz uspoređuje s ciljanom vrijednošću izlaza te se na temelju toga parametri mreže prilagođavaju. Najčešći problemi koji se rješavaju učenjem pod nadzorom su predviđanje (regresija) i klasifikacija.

- **Učenje bez nadzora**
  Težine i pragovi prilagođavaju se samo pomoću ulaznih podataka. Ova se vrsta učenja uglavnom koristi za grupiranje podataka u konačan broj grupa.

- **Učenje uz poticaj**
  Algoritam umjesto izlaznih podataka dobiva ocjenu za uspjeh te prilagođava težine i pragove s ciljem popravljanja dobivene ocjene.

Jedno od prvih pravila učenja naziva se **Hebbovo pravilo** koje je 1949. godine predstavio neuroznanstvenik Donald Hebb i predstavlja jedan od mogućih mehanizama za prilagodbu sinaptičkih veza između neurona koji se od tada koristi za prilagodbu parametara mreže. Hebbovo pravilo proizlazi iz Hebbovog postulata koji glasi: ”Kada akson jednog neurona neprestano podražuje drugi neuron, tada se među njima događa promjena tako da se učinkovitost prvog neurona povećava”. Drugim riječima, veze među neuronima osnažuju se ako jedan neuron često podražava drugi ili obratno [6].

Neuronske mreže najveću primjenu nalaze u području učenja pod nadzorom, stoga ćemo se time baviti u nastavku. Opisat ćemo najpoznatiji algoritam za učenje koji se naziva **metoda gradijentnog spusta** te **algoritam unazadne propagacije** koji služi za izračun gradijenta u višeslojnim mrežama. Sve ove koncepte objasnit ćemo imajući na umu višeslojnu acikličku arhitekturu mreže.

Poglavlje 3

Duboke acikličke neuronske mreže

U ovom ćemo poglavlju obraditi glavne koncepte koji su potrebni za učenje dubokih acikličkih neuronskih mreža. Navest ćemo najvažnije funkcije kojima mjerimo učinak mreže i razraditi metode koje koristimo za optimizaciju učinka.

3.1 Mjerenje učinka mreže

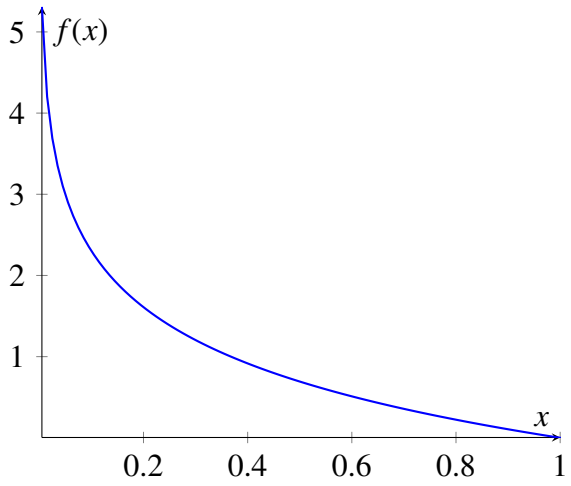
Kako bismo ocijenili rješava li neki model neuronske mreže dani problem, potrebno je definirati mjeru učinka modela. Kod problema klasifikacije, često mjerimo **točnost** modela, odnosno za dani testni skup podataka računamo omjer ispravno klasificiranih ulaznih podataka u odnosu na kardinalnost danog skupa. Takva mjera uspješnosti ne bi nam odgovarala ako rješavamo neki drugi problem, primjerice problem koji ne pripada području učenja pod nadzorom. Umjesto točnosti, analogno možemo mjeriti i **gubitak** modela. Za dani skup podataka za učenje oblika $(\mathbf{x}_i, \mathbf{y}_i), i = 1, 2, \ldots, N$ promotrimo dvije najčešće korištene **funkcije gubitka**.

Unakrsna entropija (engl. *Cross-Entropy*) je funkcija koja se koristi u klasifikacijskim mrežama čiji izlazni sloj ima ugrađenu softmax aktivacijsku funkciju, odnosno izlazne vrijednosti $o_1, o_2, \ldots, o_M$ mreže čine distribuciju vjerojatnosti. Također pretpostavlja se da su ciljne vrijednosti skupa za učenje oblika $[0, \ldots, 0, 1, 0, \ldots, 0] \in \mathbb{R}^M$ pri čemu jedinica na j-tom mjestu označava da ulazni podatak pripada klasi j (engl. *One-Hot Encoding*) [11].

Ako izlazni sloj sadrži jedan neuron, odnosno ciljne vrijednosti u skupu podataka za učenje su 0 ili 1, tada je funkcija unakrsne entropije dana s

$$E(\hat{y}, y) = -(y \ln(\hat{y}) + (1 - y) \ln(1 - \hat{y})), \tag{3.1}$$

gdje je $\hat{y}$ trenutni izlaz mreže, a y ciljni izlaz za pojedini ulazni podatak $\mathbf{x}$. Slika 3.1 prikazuje funkciju unakrsne entropije ako je ciljna vrijednost $y = 1$.



Slika 3.1: Funkcija unakrsne entropije za $y = 1$

U višeklasnoj klasifikaciji, odnosno kada je broj klasa veći od 2, funkciju unakrsne entropije računamo na sljedeći način:

$$E(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{j=1}^{M} y_j \ln(\hat{y}_j), \tag{3.2}$$

pri čemu je M broj klasa u klasifikacijskom modelu, te $\hat{\mathbf{y}} = (\hat{y_1}, \hat{y_2}, \ldots, \hat{y_M})$ vektor izlaza mreže za pojedini ulazni podatak i $\mathbf{y} = (y_1, y_2, \ldots, y_M)$ ciljni izlaz mreže u vektorskom obliku (*one-hot encoding*). Ako u obzir uzmemo cijeli skup podataka, dobivamo globalnu pogrešku koja je definirana s

$$E(\hat{Y}, Y) = \frac{1}{N}\sum_{i=1}^{N} E(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \ln(\hat{y}_{ij}), \tag{3.3}$$

pri čemu su

$$\hat{Y} = \begin{bmatrix} \hat{\mathbf{y}}_1 \\ \hat{\mathbf{y}}_2 \\ \vdots \\ \hat{\mathbf{y}}_N \end{bmatrix}, \qquad Y = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} \tag{3.4}$$

matrica predviđenih vrijednosti i ciljnih izlaza za svaki od N podataka.

Srednje kvadratna pogreška (engl. *Mean Squared Error*) ili standardna devijacija funkcija je koja se često koristi za opisivanje greške, ne samo vezano uz neuronske mreže i strojno učenje:

$$E(\hat{Y}, Y) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} (y_{ij} - \hat{y}_{ij})^2. \tag{3.5}$$

Mreže koje koriste unakrsnu entropiju općenito imaju bolje rezultate, međutim srednje kvadratna pogreška široko je zastupljena u literaturi zbog brojnih algoritama koji se bave njenom minimizacijom pa se često koristi i u dubokom učenju. Postoji više funkcija gubitka i njihov izbor ovisi o problemu, no unakrsna entropija i srednje kvadratna pogreška su najznačajnije. Kad govorimo o učenju mreže zapravo mislimo na minimizaciju određene funkcije gubitka. Budući da izlazni podatci $\hat{Y}$ ovise o ulaznim podatcima X i parametrima mreže θ, funkciju pogreške možemo promatrati i kao funkciju koja ovisi o tim veličinama te koristiti se oznakom $E(X, Y, \theta)$. Ponekad ćemo radi kraćeg zapisa pisati samo $E(\theta)$.

Funkcija koju želimo minimizirati često je dana kao zbroj jedne od navedenih funkcija gubitka i **regularizacijskog izraza** te ćemo taj zbroj nazivati **funkcijom cilja**. Regularizacija je metoda koja se koristi kako bi se spriječio preveliki rast parametara (težina i pragova) mreže. Prevelike težine i pragovi mreže znak su da je mreža pretrenirana te je pogreška na skupu podataka za testiranje velika. Korištenjem regularizacije osiguravaju se bolje performanse mreže na testnim podatcima. Neke od metoda regularizacije objasnit ćemo u sekciji 3.4.

3.2 Gradijentni spust

Gradijentni spust ili gradijentna metoda iterativni je algoritam za pronalazak lokalnog minimuma realne funkcije više varijabli. Jedan je od najpopularnijih algoritama za optimizaciju neuronskih mreža i postoje razne verzije čije se gotove implementacije nalaze u boljim programskim paketima za strojno učenje.

Označimo sa $E(\theta)$ funkciju cilja koju želimo minimizirati te pretpostavimo da je ona analitička. Tražimo način na koji ćemo prilagoditi parametar θ tako da za svaku od n iteracija algoritma vrijedi

$$E(\theta_{t+1}) < E(\theta_t), \qquad t = 0, 1, \ldots, n-1. \tag{3.6}$$

Cilj je pronaći vrijednost $\Delta\theta_t$ takvu da za $\theta_{t+1} = \theta_t + \Delta\theta_t$ vrijedi formula 3.6. Neka je sada $\Delta\theta_t = \alpha u_t$, pri čemu ćemo $\alpha > 0$ nazivati **stopom učenja**, dok vektor u_t predstavlja

vektor smjera u kojem prilagođavamo parametar iz koraka t. Kako bismo odredili vektor u_t koristimo Taylorov razvoj funkcije E u točki θ_t:

$$E(\theta_{t+1}) = E(\theta_t + \alpha u_t) \approx E(\theta_t) + \alpha \nabla_\theta E(\theta_t)^T u_t. \tag{3.7}$$

Zbog $\alpha > 0$ tražimo u_t takav da je

$$\nabla E(\theta_t)^T u_t < 0. \tag{3.8}$$

Htjeli bismo naći u_t u kojem E najbrže pada. Pronađimo

$$\min_{\|u_t\|=1} \nabla E(\theta_t)^T u_t. \tag{3.9}$$

Označimo s ϕ kut između vektora $\nabla E(\theta_t)$ i u_t. Kada je $\|u_t\| = 1$ vrijedi

$$\nabla E(\theta_t)^T u_t = \|\nabla E(\theta_t)\| \cos(\phi). \tag{3.10}$$

Minimum kosinus funkcije je -1 i postiže se za $\phi = \pi$, stoga dobivamo da je

$$u_t = -\frac{\nabla E(\theta_t)}{\|\nabla E(\theta_t)\|}, \tag{3.11}$$

međutim zbog skalara α koji množi vektor u_t u formuli 3.7 dovoljno je uzeti

$$u_t = -\nabla E(\theta_t). \tag{3.12}$$

Dobivena formula po kojoj prilagođavamo parametre mreže glasi [2]:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta E(\theta_t). \tag{3.13}$$

Klasična metoda gradijentnog spusta pretpostavlja da se funkcija pogreške, odnosno njezin gradijent računa u odnosu na sve podatke za učenje u svakom koraku algoritma. Primjerice, ako je dan skup podataka X koji sadrži N ulaznih podataka, tada je računanje gradijenta složenosti $O(N)$, što je u slučaju kada je N izrazito velik jako skupa operacija. Zato je osmišljena modifikacija pod nazivom **stohastički gradijentni spust** (engl. *stohastic gradient descent*) ili kraće **SGD**.

Naime, funkcija je gubitka na čitavom skupu podataka dana kao aritmetička sredina funkcija gubitka za pojedini podatak.

$$\nabla_\theta E(X, Y, \theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta E(\mathbf{x}_i, \mathbf{y}_i, \theta). \tag{3.14}$$

Iz tog je razloga dovoljno uzeti samo dio podataka iz originalnog skupa podataka, pod uvjetom da su uzeti podatci uniformno distribuirani unutar cijelog skupa [4].

Neka je sada $N' << N$, te sa X' označimo dio podataka iz skupa podataka X izabran na gore opisan način. Procijenjeni gradijent dan je s

$$\nabla_\theta E(X', Y', \theta) = \frac{1}{N'} \sum_{i=1}^{N'} \nabla_\theta E(\mathbf{x}_i, \mathbf{y}_i, \theta). \tag{3.15}$$

Broj $N' \in \mathbb{N}$ najčešće je fiksna vrijednost te ne ovisi o veličini skupa podataka N. U tom smislu možemo reći da je SGD algoritam $O(1)$ vremenske složenosti u odnosu na N.

Gradijentni spust dugo se smatrao sporom i nepouzdanom metodom u optimizaciji nekonveksnih funkcija. Danas znamo da učenje neuronskih mreža pomoću ove metode i njenih varijanti postiže dovoljno dobre rezultate te se u primjeni ona pokazala izrazito korisnom.

Metoda momenata

Iako stohastički gradijentni spust nudi znatno poboljšanje u odnosu na klasični gradijentni spust, učenje korištenjem ove metode također može biti izrazito sporo. Metoda momenata nastala je s ciljem da ubrza učenje, osobito u slučajevima velike zakrivljenosti funkcije cilja. Naziv metode potječe iz fizike, a ideja metode je da akumulira prethodno izračunate gradijente funkcije cilja te radi korak u njihovom smjeru.

Uvodimo novi vektor v, **vektor momenata** koji se u svakom koraku mijenja prema sljedećem pravilu:

$$v_{t+1} = \varphi v_t + \alpha \nabla_\theta E(\theta_t), \tag{3.16}$$

pri čemu je $\varphi \in [0, 1\rangle$, a α označava stopu učenja iz klasičnog gradijentnog spusta. Modificirani stohastički gradijentni spust dan je s

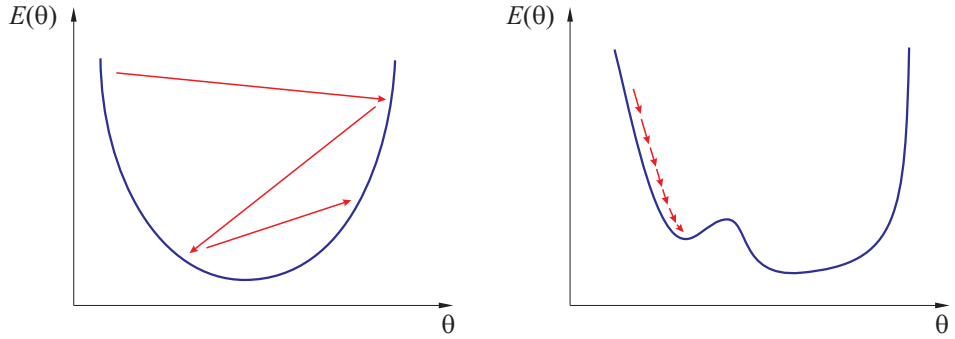
$$\theta_{t+1} = \theta_t - v_{t+1}. \tag{3.17}$$

Što je parametar φ veći, to veći utjecaj prethodne vrijednosti gradijenta funkcije cilja imaju na trenutni smjer kretanja [9]. Rezultat ove modifikacije je da će za komponente

vektora težina čiji je smjer gradijenata konstantan, odgovarajuća komponenta vektora momenta biti veća, dok će za one s čestom oscilacijom smjera biti manja. Time se smanjuju nepotrebni koraci algoritma i oscilacije u gradijentnom spustu te je konvergencija brža i stabilnija. Česte vrijednosti za φ su 0.5, 0.9 ili 0.99 te se ona može s vremenom povećavati, međutim to ima manji utjecaj na učenje od odabira ispravne stope učenja.

Stopa učenja

Razmotrimo sada odabir stope učenja α. Stopa učenja utječe na veličinu koraka u metodi gradijentnog spusta. Ako je stopa učenja premala, put prema minimumu funkcije gubitka je sporiji, dok prevelika stopa učenja znači da u svakom koraku metoda može preskočiti minimum, te završiti u suboptimalnoj točki ili učiniti učenje nestabilnim.



Slika 3.2: Prevelika (lijevo) i premala (desno) stopa učenja [1]

Stopa učenja podesivi je **hiperparametar** mreže za koji obično vrijedi $\alpha \in [0, 1\rangle$. Hiperparametrima nazivamo sve one vrijednosti o kojima ovisi učenje mreže, ali koje se kroz samo učenje ne mijenjaju. Taj naziv koristi se kako bi se napravila razlika između samih parametara mreže. Pod hiperparametre također ubrajamo broj neurona u pojedinim slojevima, kao i sâm broj slojeva.

Najjednostavniji pristup je postaviti stopu učenja na mali broj, primjerice 0.01 ili testirati performanse mreže za nekoliko različitih stopa učenja u intervalu $[0, 1\rangle$. Pokazalo se da fiksna vrijednost za stopu učenja nije najbolje rješenje. Još jedna opcija je u svakom koraku pronaći minimum funkcije $f(\alpha) = E(\theta_t - \alpha\nabla_\theta E(\theta_t))$, primjerice Newtonovom metodom [2]:

$$\alpha_t = \arg\min_{\alpha>0} E(\theta_t - \alpha\nabla_\theta E(\theta_t)). \tag{3.18}$$

U nastavku ćemo navesti neke od modifikacija gradijentnog spusta kojima je zajedničko da prilagođavaju stopu učenja u svakom koraku.

Adagrad algoritam

Do sada smo stopu učenja α u općenitoj formuli gradijentnog spusta 3.13 smatrali pozitivnim skalarom koji je isti za svaki korak algoritma. Međutim, Adagrad aloritam prilagođava stopu učenja za svaki pojedinačni parametar mreže θ_i, odnosno za svaku komponentu vektora θ računa različitu stopu učenja. Stopa učenja za pojedinu komponentu obrnuto je proporcionalna korijenu sume kvadrata svih prijašnjih vrijednosti gradijenta. Točnije, neka je G_t vektor koji na mjestu i ima sumu kvadrata parcijalnih derivacija funkcije cilja po θ_i iz svakog od dosadašnjih t koraka.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} \cdot \frac{\partial E}{\partial \theta_{t,i}}, \quad \alpha \in [0, 1], \tag{3.19}$$

pri čemu $\theta_{t,i}$ označava i-tu komponentu vektora θ u koraku t, dok ϵ služi kao dodatni parametar koji sprječava dijeljenje nulom. Stopa učenja α je konstantna, međutim dodatni ju faktor prilagođava pojedinoj komponenti vektora parametara θ. Na taj se način za komponente s manjim gradijentom čini veći korak, dok se za one s većim gradijentom sporije kreće prema minimumu funkcije gubitka.

Adagrad algoritam u primjeni nema najbolje rezultate, ali posjeduje svojstva poželjna za teoretsko razmatranje [4]. Naime, zbog gomilanja pozitivnih vrijednosti u nazivniku u svakom koraku, stopa učenja vrlo brzo postane mala te samim time dolazi do preranog prestanka učenja.

RMSprop

RMSprop je algoritam koji je predstavio Geoff Hinton u svojim predavanjima na *Coursera*-i. Nastao je kao pokušaj poboljšanja Adagrad algoritma. Osnovna je ideja ovog algoritma da umjesto akumulacije gradijenata u svakom koraku, pamti samo određen broj prethodnih gradijenata. Kako se te vrijednosti ne bi morale pamtiti, algoritam u koraku $t + 1$ računa povijest gradijenta H_t na sljedeći način:

$$H_t = \gamma\, H_{t-1} + (1 - \gamma)\nabla_\theta E(\theta_t) \odot \nabla_\theta E(\theta_t), \tag{3.20}$$

pri čemu s $\odot$ označavamo Hadamardov umnožak, odnosno umnožak po elementima.

Formulu 3.20 koristimo u formuli 3.19, ali umjesto $G_{t,i}$ koristimo i-tu komponentu vektora H_t te dobivamo korak algoritma koji glasi

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{H_{t,i} + \epsilon}} \cdot \frac{\partial E}{\partial \theta_{t,i}}, \quad \alpha \in [0, 1]. \tag{3.21}$$

Za parametar γ u formuli 3.20 u većini slučajeva se uzima vrijednost 0.9. RMSprop pokazao se efikasnim i praktičnim algoritmom za duboko učenje stoga se ova modifikacija gradijentnog spusta često koristi u primjeni.

Adam algoritam

Adam (engl. *Adaptive Moment Estimation*) još je jedan algoritam za računanje stope učenja za svaki parametar mreže. Ovaj je algoritam nastao kao kombinacija metode momenata i RMSprop algoritma. Koristimo dodatne hiperparametre β_1 i β_2 te u svakom koraku t računamo

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta E(\theta_t) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)\nabla_\theta E(\theta_t) \odot \nabla_\theta E(\theta_t) \end{aligned} \tag{3.22}$$

pri čemu m_t nazivamo **prvi moment**, a v_t **drugi moment**. Zatim dobivene vektore normaliziramo:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}. \end{aligned} \tag{3.23}$$

Faktor $\dfrac{1}{1-\beta^t}$ opravdan je sljedećom formulom:

$$\sum_{i=0}^{t} \beta^i = \frac{1-\beta^t}{1-\beta}.$$

Dobivamo algoritam za učenje sljedećeg oblika:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{\hat{v}_{t,i}} + \epsilon}\hat{m}_{t,i}, \quad \alpha \in [0,1]. \tag{3.24}$$

Primijetimo da se u formuli 3.24 ϵ nalazi izvan znaka korijena. Pokazalo se da takva formula radi bolje u praksi, što je razlog nekonzistentnosti u odnosu na formule 3.19 i 3.21 [15].

Autori ovog algoritma predlažu da se koriste sljedeće vrijednosti $\beta_1 = 0.9$, $\beta_2 = 0.999$ te $\epsilon = 10^{-8}$. Empirički je pokazano da ove vrijednosti daju dobre rezultate u velikom broju slučajeva [15].

3.3 Algoritam unazadne propagacije

Algoritam unazadne propagacije postupak je iterativnog prilagođavanja težina i pragova višeslojne neuronske mreže koji koristi metodu gradijentnog spusta. Sâm algoritam osmišljen je 1970-ih godina kao općeniti algoritam za optimizaciju računanja gradijenta funkcija koje su nastale višestrukim komponiranjem, no tek je 1986. godine objavom rada autora Rumelharta, Hintona i Williamsa pod naslovom *Learning Representations by Back-Propagating Errors*, algoritam postao cijenjen u području strojnog učenja. Do tog je trenutka prilagođavanje težina i pragova unutarnjih slojeva mreže bio dugotrajan posao te je zahtijevao stručnjake u području za koje se neuronska mreža koristila. Ovaj je algoritam omogućio primjenu neuronskih mreža na veći broj područja i uvelike raširio njihovu upotrebu.

Za danu funkciju gubitka (pogreške) $E(X, Y, \theta)$, gdje je $X \in M_{Nd}(\mathbb{R})$ matrica dizajna, Y kao u formuli 3.4, a θ vektor parametara mreže, metoda računa njen gradijent u odnosu na parametar θ te ga prilagođava prema formuli gradijentnog spusta

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta E(X, Y, \theta_t). \tag{3.25}$$

Račun se odvija od zadnjeg, izlaznog sloja mreže, prema prvome te se zbog toga algoritam naziva algoritam unazadne propagacije. Funkcija je gubitka na svim podacima odnosno čitavoj matrici dizajna dana kao srednja vrijednost gubitka na pojedinom retku matrice dizajna:

$$E(X, Y, \theta) = \frac{1}{N} \sum_{i=1}^{N} E(\mathbf{x}_i, \mathbf{y}_i, \theta), \tag{3.26}$$

stoga i za gradijent vrijedi

$$\nabla_\theta E(X, Y, \theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta E(\mathbf{x}_i, \mathbf{y}_i, \theta), \tag{3.27}$$

gdje je $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]$ i-ti redak matrice dizajna.

Zato ćemo u nastavku razraditi metodu za jedan ulazni vektor $\mathbf{x}$. Radi jednostavnosti zapisa, koristit ćemo formulu 2.13, odnosno smatrat ćemo da je vektor b^l uključen u matricu težina l-tog sloja W_l kao njen zadnji redak.

Vektor θ sastoji se od svih parametara mreže. Uzmimo jedan element vektora θ i označimo ga s w_{ij}^l, što predstavlja težinu veze koja s j-tim neuronom u l-tom sloju spaja izlaz i-tog neurona prethodnog sloja. Ovdje i prag pojedinog neurona b_j^l radi jednostavnosti

nazivamo težinom. Promotrimo derivaciju funkcije gubitka E za pojedini parametar mreže w_{ij}^l:

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{ij}^l}. \tag{3.28}$$

Iz formule 2.12 slijedi da je

$$\frac{\partial z_j^l}{\partial w_{ij}^l} = o_i^{l-1}, \tag{3.29}$$

dok ćemo prvi dio označavati sa δ_j^l:

$$\delta_j^l = \frac{\partial E}{\partial z_j^l}. \tag{3.30}$$

Izlazni sloj

Promotrimo najprije račun za δ_j^s, gdje je s ukupan broj slojeva mreže. Koristeći oblik formule za funkciju pogreške 3.3 vrijedi:

$$E(\, o_j^s, y_j\,) = E(\, f_s(\, z_j^s\,), y_j\,), \tag{3.31}$$

pa je

$$\delta_j^s = \frac{\partial E}{\partial z_j^s} = \frac{\partial E}{\partial o_j^s}\, f_s'(\, z_j^s\,). \tag{3.32}$$

Sada iz formula 3.29 i 3.32 slijedi

$$\frac{\partial E}{\partial w_{ij}^s} = \delta_j^s o_i^{s-1} = \frac{\partial E}{\partial o_j^s}\, f_s'(\, z_j^s\,)\, o_i^{s-1} \tag{3.33}$$

Skriveni slojevi

Neka je sada $1 \leq l < s$. Tada vrijedi

$$z_k^{l+1} = \sum_{j=1}^{r^l} w_{jk}^{l+1}\, f_l\,(\, z_j^l\,), \tag{3.34}$$

stoga je

$$\delta_j^l = \sum_{k=1}^{r^{l+1}} \frac{\partial E}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_{k=1}^{r^{l+1}} \delta_k^{l+1}\, w_{jk}^{l+1}\, f_l'(\, z_j^l\,). \tag{3.35}$$

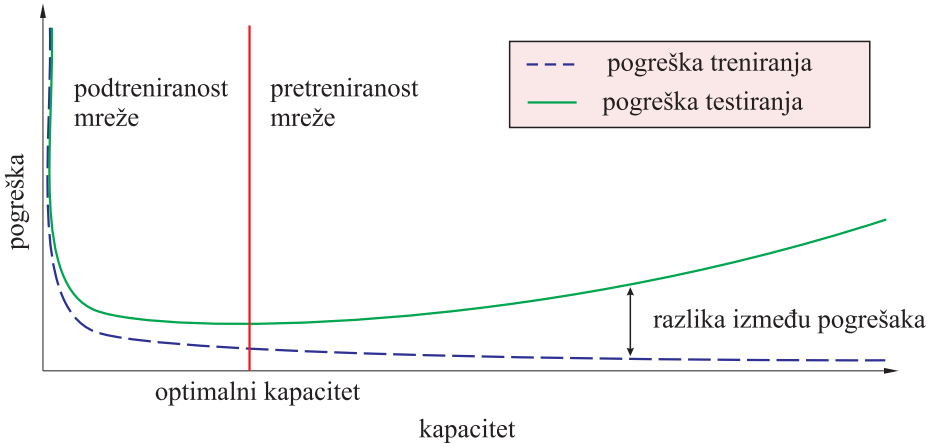
Iz gornje formule dobivamo da se derivacija funkcije gubitka po težinama u skrivenim slojevima računa na sljedeći način:

$$\frac{\partial E}{\partial w_{ij}^{l}} = \delta_j^l o_i^{l-1} = f_l'(z_j^l)\, o_i^{l-1} \sum_{k=1}^{r^{l+1}} \delta_k^{l+1}\, w_{jk}^{l+1} \tag{3.36}$$

Iz posljednje formule vidljivo je po čemu je ovaj algoritam dobio naziv. Naime, komponente gradijenta funkcije gubitka potrebno je računati od zadnjeg sloja prema prvom jer svaki novi sloj koristi vrijednosti δ^{l+1} koje su izračunate za prethodni sloj.

3.4 Regularizacija

Problemi s kojima se neuronska mreža susreće prilikom učenja su problemi **podtreniranosti i pretreniranosti mreže** (engl. *underfitting and overfitting*). Podtreniranost mreže znači da mreža nije u stanju dovoljno minimizirati funkciju gubitka na skupu podataka za učenje. Suprotan pojam je pretreniranost mreže. To je pojava u kojoj je mreža suviše dobro prilagodila svoje parametre skupu podataka za učenje te više ne može postići zadovoljavajuću točnost na skupu podataka za testiranje. To se događa zato što algoritmi za učenje rade na principu smanjivanja pogreške učenja te pri tome nemaju uvid u testne podatke, a cilj je bilo kojeg modela strojnog učenja, pa tako i neuronskih mreža, da pogreška na neviđenim (testnim) podatcima također bude mala. Ta se pogreška naziva **pogreška generalizacije** ili **pogreška testiranja**.



Slika 3.3: Pogreške treniranja i testiranja u odnosu na kapacitet mreže [4]

Pretreniranost mreže očituje se u prevelikoj razlici između male pogreške treniranja i velike pogreške testiranja. Sposobnost mreže da se prilagodi podatcima naziva se **kapa-**

citet mreže te je on proporcionalan broju parametara mreže te veličini skupa podataka za učenje. Dakle kompleksnija mreža koja ima više slojeva neurona te veći broj neurona u pojedinom sloju ima sklonost da bude pretrenirana. Slika 3.3 prikazuje greške treniranja i testiranja u odnosu na kapacitet mreže.

Postoje brojne metode kojima se pokušavaju popraviti performanse mreže na testnim podatcima, odnosno smanjiti pogreška testiranja (generalizacije), ponekad time povećavajući pogrešku treniranja (učenja). Te se metode zajednički nazivaju **regularizacijom**.

L_1 i L_2 regularizacija

Ova vrsta regularizacije odvija se dodavanjem norme vektora parametara mreže funkciji gubitka. U literaturi se ova metoda naziva i **kažnjavanje normom parametara** (engl. *parameter norm penalties*). Dakle, za funkciju gubitka $E(\theta)$ i $\lambda \in \mathbb{R}^+$ koji nazivamo **stopom regularizacije** definiramo **regulariziranu funkciju cilja** $\tilde{E}$ kao

$$\tilde{E}(\theta) = E(\theta) + \lambda\, \Omega(\theta) \tag{3.37}$$

Treba naglasiti da se kod ove vrste regularizacije funkcije cilja često izostavljaju pragovi neurona iz parametara funkcije Ω te se u obzir uzimaju isključivo težine [4]. Vektor parametara mreže iz kojeg su isključeni pragovi neurona označavat ćemo s ω.

L_2 regularizacija

U ovom je slučaju funkcija Ω dana s

$$\Omega(\omega) = \frac{1}{2}\|\omega\|_2, \tag{3.38}$$

pa funkcija cilja poprima sljedeći oblik:

$$\tilde{E}(\omega) = E(\omega) + \lambda\, \frac{1}{2}\|\omega\|_2. \tag{3.39}$$

Njen je gradijent dan s

$$\nabla_\omega\, \tilde{E}(\omega) = \nabla_\omega E(\omega) + \lambda\, \omega, \tag{3.40}$$

stoga iz 3.13 jedan korak u algoritmu gradijentnog spusta glasi:

$$\omega_{t+1} = \omega_t - \alpha \nabla_\omega E(\omega_t) - \alpha\lambda\, \omega_t = (1 - \alpha\lambda)\, \omega_t - \alpha \nabla_\omega E(\omega_t) \tag{3.41}$$

Iz posljednje se jednadžbe vidi modifikacija koraka algoritma koja osigurava smanjivanje težina u svakom koraku algoritma.

L_1 regularizacija

Iako se u praksi češće koristi L_2 regularizacija, ponekad kao funkciju Ω možemo uzeti i L_1 normu vektora težina

$$\Omega(\omega) = \|\omega\|_1 = \sum_i |\omega_i|. \tag{3.42}$$

Sada je funkcija cilja dana s

$$\tilde{E}(\omega) = E(\omega) + \lambda \|\omega\|_1, \tag{3.43}$$

pa je gradijent dan s

$$\nabla_\omega \tilde{E}(\omega) = \nabla_\omega E(\omega) + \lambda \ \text{sign}(\omega), \tag{3.44}$$

Uočavamo da ovakva regularizacija u algoritmu gradijentnog spusta uzima u obzir samo predznak pojedine težine, stoga u svakom koraku algoritma težinama uz gradijent funkcije gubitka dodajemo ili oduzimamo konstantnu vrijednost. Zbog toga je ova vrsta regularizacije slabija od ranije navedene L_2 regularizacije.

Paramatar λ spada u hiperparametre mreže i vrijednost je koja ovisi o konkretnom problemu. Za $\lambda = 0$ funkcije cilja jednaka je originalnoj, dakle nema regularizacije. Što je parametar λ veći, regularizacija je značajnija.

Generiranje podataka

Ova vrsta regularizacije različita je od dosad navedenih metoda te počiva na činjenici da veći skup podataka za učenje znači bolju sposobnost generalizacije, odnosno prilagodbu novim podacima. Međutim, u većini slučajeva ne možemo pribaviti po volji mnogo podataka za učenje. Stoga možemo iz postojećih podataka generirati nove podatke koje ćemo dodati skupu za učenje i time smanjiti pogrešku generalizacije.

Primjerice, u klasifikaciji gdje se skup podataka za učenje sastoji od parova slika i njihovih oznaka, želimo da mreža jednako klasificira određenu sliku, kao i istu sliku zarotiranu za 90°. Istim principom možemo raznim transformacijama kao što su rotacija, translatacija ili skaliranje, povećati skup podataka za učenje te time popraviti performanse mreže. Pri tome treba biti oprezan da transformacijom ne dobijemo podatak koji pripada

drugoj klasi. Naime, rotacijom slike broja 9 za 180° dobivamo sliku broja 6 te više nema smisla da modificirana slika zadrži istu oznaku [4].

Opisana se metoda ne može iskoristiti u svakoj situaciji te izravno ovisi o problemu kojeg rješavamo te o skupu podataka koji nam je dostupan. Najveću primjenu nalazi u zadatcima kao što je prepoznavanje objekata na slikama.

Postoje brojne dodatne metode regularizacije koje nećemo obraditi u ovom radu. Regularizacija je uz optimizaciju učenja centralna tema u području neuronskih mreža.

3.5 Kriterij zaustavljanja

Kako bismo odredili kada algoritam učenja staje obično koristimo određen broj iteracija. Broj iteracija kod gradijentnog spusta određen je brojem podataka za koje računamo gradijent i brojem izvođenja algoritma za cijeli skup podataka za treniranje.

Broj podataka koji koristimo za izračun gradijenta funkcije gubitka nazivamo **skupina** ili **serija** (engl. *batch*).

Jedno izvođenje algoritma za učenje nazivamo **epoha**. Za vrijeme učenja kažemo da je prošla jedna epoha kada je svaki podatak iz skupa podataka za učenje korišten za računanje gradijenta funkcije gubitka točno jednom.

Primjerice, ako naš skup podataka za učenje sadržava 50 000 podataka, te odredimo da je veličina serije 500 tada će se u jednoj epohi odraditi 100 iteracija algoritma. Zatim, ako odredimo da učenje tranje 5 epoha, tada će ukupan broj iteracija biti jednak 500.

Ne postoji optimalan broj iteracija već je potrebno pronaći odgovarajuću veličinu serije i broj epoha. Prevelik broj epoha dovodi do pretreniranosti, dok prevelika serija dovodi do sporog računanja gradijenta funkcije cilja.

Poglavlje 4

Prepoznavanje uzoraka

U ovom ćemo poglavlju primijeniti neke od navedenih metoda rješavanjem problema prepoznavanja rukom pisanih znamenki. Za implementaciju koristit ćemo JavaScript biblioteku Tensorflow JS [10].

Tensorflow JS je biblioteka za strojno učenje u jeziku JavaScript. Pogodna je za jednostavnu i intuitivnu konstrukciju neuronskih mreža te uporabu optimizacijskih algoritama za učenje. Također nudi brojne metode vizualizacije učenja i dobivenih rezultata. Biblioteka sadržava funkcije koje omogućuju implementaciju neuronske mreže na vrlo niskoj razini, kao i gotove optimizacijske funkcije, neke od kojih smo naveli u prethodnom poglavlju.

Naša implementacija nudi mogućnost kreiranja više različitih modela neuronskih mreža odabirom spomenutih hiperparametara i metoda učenja na interaktivnoj mrežnoj stranici `http://web.studenti.math.pmf.unizg.hr/~egracan/diplomski/index.html`. Stranica se sastoji od četiri dijela koji se moraju koristiti jedan za drugim.

U prvom dijelu korisnik može odrediti koliki će dio podataka biti u skupu za učenje, a koliki u skupu za testiranje. Preporučuje se da skup za učenje sadrži otprilike 80% podataka, a skup za testiranje 20% [15]. U tom se dijelu stranice odabire i arhitektura mreže, odnosno algoritam za učenje, stopa učenja, kao i vrsta regularizacije te stopa regularizacije. Osim toga, moguće je izabrati i određen broj primjera te ih prikazati na ekranu. Primjeri se izvlače nasumce iz skupa podataka za učenje.

Drugi je dio namijenjen za vizualizaciju tijeka učenja pomoću grafova koji prikazuju napredak algoritma za učenje. Nakon toga slijedi treći dio, odnosno testiranje modela gdje možemo vidjeti koliko je kreirani model uspješan za odabrani dio skupa podataka za testiranje. Posljednji dio prikazuje plohu za crtanje na kojoj korisnik može sâm nacrtati neku znamenku kako bi ju mreža pokušala prepoznati.

4.1 Skup podataka

Za učenje i analizu kreiranog modela neuronske mreže koristit ćemo **MNIST** skup podataka (engl. *Modified National Institute of Standards and Technology database*).

Ovaj je skup podataka pogodan za usporedbu i analizu arhitektura mreže jer je vrlo česta tema u strojnom učenju. Koriste ga početnici kao i napredni znanstvenici pri testiranju novih modela i algoritama za klasifikaciju. MNIST skup podataka sastoji se od 65 000 crno-bijelih slika rukom pisanih znamenaka. Svaka je slika dimenzije 28×28 piksela, međutim predstavljena je u obliku vektora dimenzije 784. Značajke pojedine ulazne slike u ovom su slučaju jačine piksela, odnosno vrijednosti od 0 do 255.

Na slici 4.1 vidimo kako izgleda dio web stranice na kojem možemo izabrati veličinu skupova podataka za učenje i testiranje.

Slika 4.1: Izgled stranice s izbornikom za podatke

Za učitavanje podataka koristimo JavaScript klasu *MnistData* koja je preuzeta s mrežnih stranica Tensorflowa [10] i prilagođena za daljnje korištenje. Klasa sadrži metode *load*, *nextTrainBatch* i *nextTestBatch*.

Metoda *load* služi za učitavanje slika i njihovih oznaka te ih raspoređuje na skup podataka za testiranje i skup podataka za učenje u zadanom omjeru. Preostale dvije metode služe za uzimanje zadanog broja nasumičnih ulaznih podataka (engl. *batch size*) i njihovih oznaka iz skupa za učenje, odnosno skupa za testiranje. U našem ćemo primjeru mreže koristiti 55 000 podataka za učenje te 10 000 podataka za testiranje.

4.2 Arhitektura mreže

Na dijelu stranice s arhitekturom mreže omogućen je izbor aktivacijske funkcije u skrivenim slojevima mreže. U zadnjem se sloju od 10 neurona koristi *softmax* aktivacijska funkcija kako bismo dobili vjerojatnosti pripadnosti ulaznog podatka nekoj od 10 klasa. Zatim, moguće je izabrati između jednog od četiriju optimizacijskih algoritama gradijentnog spusta: stohastički gradijentni spust (3.15), Adagrad algoritam (3.19), RMSProp (3.21) ili Adam (3.24) te odabrati stopu učenja koja će se u tom algoritmu koristiti. Konačno, možemo odlučiti hoće li naš model mreže koristiti regularizaciju te odabrati stopu regularizacije. Kao primjer mreže koja postiže dobre rezultate uzet ćemo sljedeće parametre:

- aktivacijska funkcija: ReLu (2.6)
- optimizator gradijentnog spusta: Adam (3.24)
- stopa učenja α: 0.01
- bez regularizacije

Implementacija mreže

Mreža sadržava 5 slojeva, dva para konvolucijskog sloja i sloja sažimanja te posljednji potpuno povezan sloj. U nastavku slijedi dio kôda koji se nalazi u funkciji *createNetwork*. Dijelovi koji su promjenjivi kroz sučelje mrežne stranice su aktivacijska funkcija u konvolucijskim slojevima, kao i regularizacija u konvolucijskim slojevima i posljednjem sloju te algoritam za učenje, odnosno neka od optimizacija gradijentnog spusta.

```
1   model = tf.sequential(); // feedforward mreza
2
3   model.add(tf.layers.conv2d({
4     inputShape: [data.IMAGE_WIDTH, data.IMAGE_HEIGHT,
          data.IMAGE_CHANNELS],
5     kernelSize: 5,
6     filters: 8,
7     strides: 1,
8     activation: activationf,
9     kernelInitializer: 'varianceScaling',
10    kernelRegularizer: regulizer
11  }));
12
13  // smanjenje dimenzionalnosti koristenjem maksimuma umjesto
       prosjeka
14  model.add(tf.layers.maxPooling2d({
15    poolSize: [2, 2], strides: [2, 2]
```

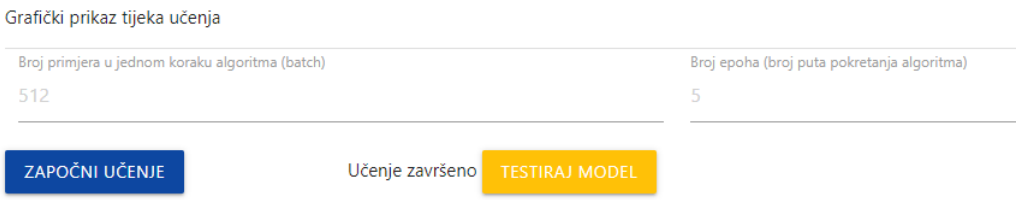
```
16    }));
17
18    model.add(tf.layers.conv2d({
19      kernelSize: 5,
20      filters: 16,
21      strides: 1,
22      activation: activationf,
23      kernelInitializer: 'varianceScaling',
24      kernelRegularizer: regulizer
25    }));
26    model.add(tf.layers.maxPooling2d({
27      poolSize: [2, 2], strides: [2, 2]
28    }));
29
30    // dvodimenzionalne ulazne podatke stavljamo u jednodimenzionalne
31    model.add(tf.layers.flatten());
32
33    // gusti sloj sa 10 neurona za svaku izlaznu klasu
34    // (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
35    model.add(tf.layers.dense({
36      units: data.NUM_CLASSES,
37      kernelInitializer: 'varianceScaling',
38      activation: 'softmax',
39      kernelRegularizer: regulizer
40    }));
41
42    model.compile({
43      optimizer: optimizer,
44      loss: 'categoricalCrossentropy',
45      metrics: ['accuracy'],
46    });
```

Ispis 4.1: Dio kôda zadužen za konstrukciju slojeva mreže

Dio implementacije mreže koji nije promjenjiv je dimenzija jezgre konvolucijskih slojeva koja je u našem slučaju 5×5, kao i pomak koji je jednak jedan. Tako iz slike dimenzije 28×28 dobivamo mapu značajki dimenzije 24×24. U prvom konvolucijskom sloju definiramo osam jezgri, odnosno filtera, što znači da očekujemo osam izlaznih mapi značajki, te da je broj parametara koje mreža prilagođava jednak $5 \cdot 5 \cdot 8 = 200$. Drugi konvolucijski sloj ima jezgru iste dimenzije, međutim povećavamo njihov broj na 16. Nakon svakog konvolucijskog sloja dolazi sloj sažimanja maksimalnom vrijednošću čiji je prozor dimenzije 2×2. U tim slojevima pomak je jednak 2 u vertikalnom i 2 u horizontalnom smjeru. Model kao funkciju gubitka koristi unakrsnu entropiju (3.3). Funkcija *compile* nudi mogućnost odabira dodatnih metrika koje služe za praćenje učenja. Koristimo funkciju **točnosti**, odnosno postotak ispravno klasificiranih podataka.

4.3 Učenje mreže

Na dijelu stranice za učenje odabiremo količinu podataka za izračun gradijenta u jednoj iteraciji algoritma, odnosno veličinu serije (engl. *batch size*) te broj epoha, odnosno ukupan broj izvođenja algoritma na cijelom skupu podataka za treniranje. Za gore navedenu arhitekturu mreže odabiremo broj epoha jednak 5 i veličinu serije jednaku 500 podataka.



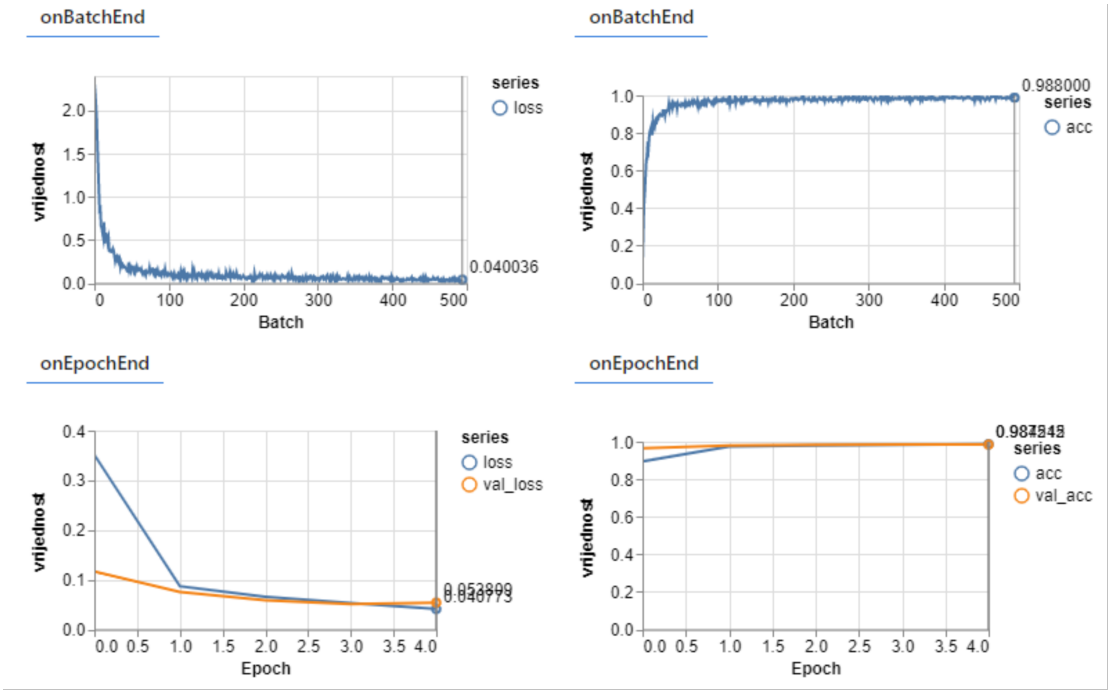
Slika 4.2: Dio stranice s izbornikom za učenje

Promotrimo dio koda koji je zadužen za učenje. Funkcija *fit* koristi prethodno kreiran model i primjenjuje zadani algoritam učenja:

```
1   await model.fit(trainXs, trainYs, {
2     batchSize: BATCH_SIZE,
3     validationSplit: 0.1, // 10 posto podataka ide na validaciju
4     epochs: epochs,
5     shuffle: true,
6     callbacks: cb // grafovi za prikaz ucenja
7   });
```

ValidationSplit označava koliki će se dio podataka za treniranje koristiti za validaciju. Na tom dijelu podataka neće se računati gradijent, odnosno služi isključivo za kontrolu rada algoritma. Gornja implementacija uzima 10% podataka iz skupa za treniranje i koristi ih za validaciju. Zbog toga u stvarnosti imamo 49 500 podataka za učenje, što čini 99 skupina po 500 podatka u jednoj epohi. Time je kroz 5 epoha ukupan broj iteracija algoritma jednak 495.

Argument *callbacks* sadrži definiciju funkcija koje se pozivaju za vrijeme treninga. Te se funkcije pozivaju na kraju svake serije te na kraju svake epohe algoritma i vraćaju vrijednosti funkcije gubitka i točnosti na skupu za treniranje i validacijskom skupu. Te vrijednosti služe za prikazivanje četiriju grafova na dijelu stranice za učenje mreže. Na slici 4.3 vidimo prikaz tijeka učenja za mrežu iz primjera. Za tu je mrežu vrijednost funkcije gubitka na kraju pete epohe učenja jednaka 0.041 na skupu podataka za učenje, dok je na validacijskom skupu ona jednaka 0.054.



Slika 4.3: Prikaz tijeka učenja za svih 495 iteracija algoritma

4.4 Testiranje

Kako bismo što efikasnije mogli vizualizirati uspješnost kreiranog i naučenog modela, mrežna stranica nudi mogućnost grafičkog prikaza postotka točnosti za svaku klasu posebno. Primjer ispisa za mrežu naučenu u prethodnoj sekciji prikazan je na slici 4.4.

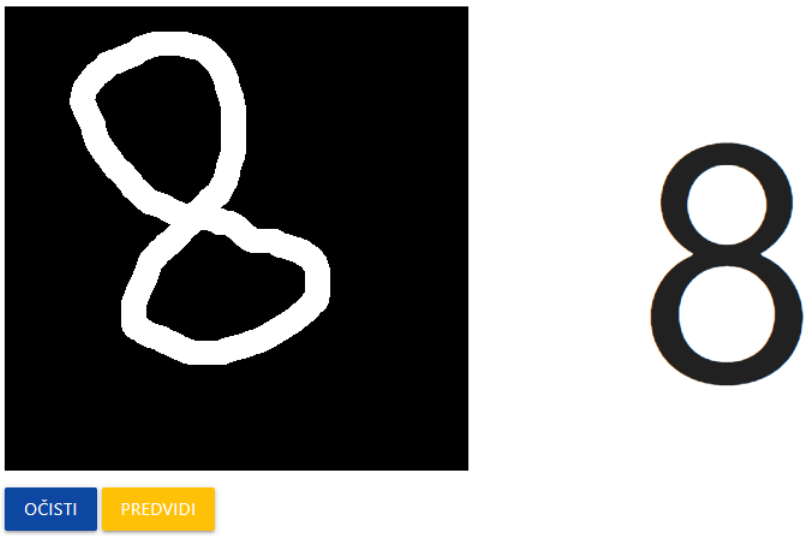
Class	Accuracy	# Samples
Nula	0.9959	980
Jedan	0.9903	1,135
Dva	0.9913	1,032
Tri	0.9871	1,010
Četiri	0.9807	982
Pet	0.9832	892
Šest	0.977	958
Sedam	0.9844	1,028
Osam	0.9887	974
Devet	0.9851	1,009

Slika 4.4: Postotak točnosti za pojedinu klasu

Za dodatnu provjeru, na istom dijelu stranice možemo ispisati **matricu zabune** na čijem se (i, j)-tom mjestu nalazi koliko je puta broj i klasificiran kao broj j. Ukupna je pogreška na čitavom skupu podataka za testiranje jednaka 0.042.

Slika 4.5: Matrica zabune

Sljedeći dio stranice također pripada testiranju, ali dozvoljava nam da kreirano novi podatak na plohi za crtanje. Učitani podatak predaje se naučenoj mreži koja predviđa o kojoj se znamenci radi. Opisani postupak možemo vidjeti na slici 4.6



Slika 4.6: Testiranje mreže na novim podacima

Bibliografija

[1] S. Bhattarai, *Learning Rate Gradient Descent*, 2018., https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/learning-rate-gradient-descent/.

[2] N. Bosner, *Znanstveno računanje 2, Nelinearni sustavi jednadžbi*, 2019., https://web.math.pmf.unizg.hr/~nela/zr2vjezbe/zr2_newton.pdf.

[3] B. Dalbelo Bašić, M. Čupić i J. Šnajder, *Umjetne neuronske mreže*, 2008., https://www.fer.hr/_download/repository/UmjetneNeuronskeMreze.pdf.

[4] I. Goodfellow, Y. Bengio i A. Courville, *Deep Learning*, MIT Press, 2016, http://www.deeplearningbook.org.

[5] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2019., https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/.

[6] M. T. Hagan, H. B. Demuth, M. H. Beale i O. De Jesús, *Neural Network Design, 2nd Edition*, 2015., http://hagan.okstate.edu/NNDesign.pdf.

[7] S. Lončarić, *Neuronske mreže, predavanja @FER*, 2019., https://www.fer.unizg.hr/predmet/neumre_b/predavanja.

[8] R. Patidar i L. Sharma, *Credit Card Fraud Detection Using Neural Network*, 2011., https://pdfs.semanticscholar.org/0419/c275f05841d87ab9a4c9767a4f997b61a50e.pdf.

[9] S. Ruder, *An overview of gradient descent optimization algorithms*, 2016., http://ruder.io/optimizing-gradient-descent/.

[10] Tensorflow, *Tensorflow JS API Reference*, 2019., https://js.tensorflow.org/api/latest/.

[11] Read the Docs, *Machine Learning Cheatsheet*, 2018., https://ml-cheatsheet.readthedocs.io/en/latest/.

[12] Udacity i Google, *Intro to TensorFlow for Deep Learning*, 2018., https://classroom.udacity.com/courses/ud187.

[13] Wikipedia, *Optical Character Recognition*, 2013., https://en.wikipedia.org/wiki/Optical_character_recognition.

[14] _____, *Umjetna Inteligencija*, 2019., https://hr.wikipedia.org/wiki/Umjetna_inteligencija.

[15] Aston Zhang, Zachary C. Lipton, Mu Li i Alexander J. Smola, *Dive into Deep Learning*, 2020, https://d2l.ai.

[16] S. Šegvić, *Duboko učenje, predavanja @FER*, 2019., http://www.zemris.fer.hr/~ssegvic/du/.

Sažetak

Ovaj diplomski rad pruža pregled osnovnih pojmova o području strojnog učenja koje nazivamo duboko učenje, odnosno o umjetnim neuronskim mrežama i njihovoj konstrukciji.

U prvom dijelu objašnjavaju se glavne ideje vezane uz umjetne neuronske mreže kao što su težine i pragovi te navode neki od istaknutijih primjera iz primjene.

Drugi dio sadržava detaljan opis građe umjetnog neurona, odnosno način izračuna izlaznih podataka pojedinog neurona. Navedene su neke od često korištenih funkcija prijelaza u neuronu i njihova svojstva. U istom se djelu nalazi detaljan opis i matematička interpretacija povezivanja neurona u slojeve. Također, spominju se najvažnija svojstva konvolucijskog sloja koji je važan za obradu slika i zvuka.

Treći dio usredotočen je na metode popravljanja performansi mreže, odnosno učenje. Navodimo funkcije koje se koriste za mjerenje učinka mreže. Zatim promatramo metodu gradijentnog sputa, kao i druge metode nastale u nastojanju poboljšanja gradijentnog spusta. Detaljno raspisujemo algoritam unazadne propagacije koju služi bržem izračunu gradijenta u višeslojnim neuronskim mrežama.

Konačno, u zadnjem dijelu opisujemo implementaciju mrežne stranice te komentiramo rezultate jedne neuronske mreže kreirane putem te stranice.

Summary

This thesis gives an overview of artificial neural networks and their construction. The first chapter explains the basic concepts concerning neural networks such as weights and biases, and mentions some interesting applications.

The second chapter contains a detailed description of the artificial neuron structure and explains calculation of output values performed by a single neuron. This chapter also lists the transfer functions used in neurons, as well as describes the way neurons are connected into layers, and gives a mathematical interpretation of this concept. Finally, it explains the basic principles of convolutional neural networks which are important in the fields of image or sound processing.

In the third chapter the focus is on learning methods for neural networks. Firstly, we introduce the two most important loss functions used for measuring the success of a neural network. Then we explain the basic method behind minimizing the loss function - gradient descent, and mention some of the most successful optimization algorithms based on gradient descent.

The fourth and final chapter describes implementation of an interactive website that uses the TensorflowJS library for creating different convolutional network models.

Životopis

Rođena sam 02.08.1993. u Zagrebu. Pohađala opći smjer X. gimnazije "Ivan Supek", također u Zagrebu. Upisala sam Prirodoslovno-matematički fakultet Sveučilišta u Zagrebu 2012. godine. Preddiplomski smjer Matematika završila sam 2017. godine, kada sam upisala Diplomski studij Primijenjena matematika. Na tom sam studiju bila jednu akademsku godinu, nakon čega sam upisala Diplomski studij Računarstvo i matematika.

Za vrijeme studija u trajanju od jedne godine radila sam kao pomoćnica uredniku časopisa CIT. Journal of Computing and Information Technology na Fakultetu elektrotehnike i računarstva gdje sam obavljala poslove korespondencije s autorima te objavljivanja digitalnog izdanja časopisa. 2018. godine zaposlila sam se u Zagrebačkoj banci kao student u Aplikativnom razvoju kreditne aplikacije gdje sam početkom 2020. godine započela stalni radni odnos kao specijalistica za dizajn i razvoj aplikativnih rješenja.

Osim toga, bavim se veslanjem od dvanaeste godine. Sudjelovala sam na brojnim domaćim i međunarodnim natjecanjima. 2015. godine bila sam državna prvakinja u lakom samcu, a u posljednje sam vrijeme nastupala za studentsku reprezentaciju Sveučilišta u Zagrebu. Najznačajniji rezultat ostvarila sam 2018. godine na Europskim sveučilišnim igrama u Coimbri, u Portugalu, gdje sam u posadi ženskog osmerca osvojila brončanu medalju.