# DIPLOMSKI

Table of Contents

[DIPLOMSKI 1](#_Toc66873596)

[DUBOKO UČENJE I UMJETNE NEURONSKE MREŽE 3](#_Toc66873597)

[ŠTO JE DUBOKO UČENJE 3](#_Toc66873598)

[UVOD U UMJETNE NEURONSKE MREŽE 3](#_Toc66873599)

[SLOJEVI U NEURONSKOJ MREŽI 4](#_Toc66873600)

[VEZE IZMEĐU SLOJEVA 6](#_Toc66873601)

[PROSLJEĐIVANJE PREMA NAPRIJED 8](#_Toc66873602)

[KONVOLUCIJSKE NEURONSKE MREŽE 8](#_Toc66873603)

[STRUKTURA JEDNOSTAVNE KONVOLUCIJSKE NEURONSKE MREŽE 11](#_Toc66873604)

[ULAZNI SLOJ 11](#_Toc66873605)

[KONVOLUCIJSKI SLOJ 11](#_Toc66873606)

[FILTERI 12](#_Toc66873607)

[ZERO PADDING 15](#_Toc66873608)

[POMAK 16](#_Toc66873609)

[AKTIVACIJSKE FUNKCIJE U UMJETNOJ NEURONSKOJ MREŽI 17](#_Toc66873610)

[SIGMOIDNA AKTIVACIJSKA FUNKCIJA 18](#_Toc66873611)

[HIPERBOLIČKO-TANGENTNA AKTIVACIJSKA FUNKCIJA 20](#_Toc66873612)

[ReLU AKTIVACIJSKA FUNKCIJA 21](#_Toc66873613)

[LEAKY ReLU AKTIVACIJSKA FUNKCIJA 23](#_Toc66873614)

[SWISH AKTIVACIJSKA FUNKCIJA 24](#_Toc66873615)

[SOFTMAX AKTIVACIJSKA FUNKCIJA 25](#_Toc66873616)

[SLOJ SAŽIMANJA 26](#_Toc66873617)

[POTPUNO POVEZANI SLOJEVI I IZLAZNI SLOJ 28](#_Toc66873618)

[TRENIRANJE UMJETNIH NEURONSKIH MREŽA 30](#_Toc66873619)

[FUNKCIJA GUBITKA 30](#_Toc66873620)

[KAKO NEURONSKA MREŽA UČI 30](#_Toc66873621)

[GRADIJENT 31](#_Toc66873622)

[STOHASTIČKI GRADIJENTNI SPUST 32](#_Toc66873623)

[SKUPOVI ZA TRENIRANJE, PROVJERU I TESTIRANJE 34](#_Toc66873624)

[REDUCIRANJE PRENAUČENOSTI I PODNAUČENOSTI 35](#_Toc66873625)

[PRIMJER, GRUPA, EPOHA 38](#_Toc66873626)

[PRAG 39](#_Toc66873627)

[POVRATNA PROPAGACIJA 41](#_Toc66873628)

[PREGLED STOHASTIČKOG GRADIJENTNOG SPUSTA (SGD) 41](#_Toc66873629)

[PROPAGACIJA PREMA NAPRIJED 41](#_Toc66873630)

[RAČUNANJE GUBITKA 42](#_Toc66873631)

[INTUICIJA POVRATNE PROPAGACIJE 42](#_Toc66873632)

[MATEMATIČKA NOTACIJA POVRATNE PROPAGACIJE 44](#_Toc66873633)

[MATEMATIČKA PROMATRANJA ZA POVRATNU PROPAGACIJU 45](#_Toc66873634)

[FUNKCIJA GUBITKA 46](#_Toc66873635)

[ULAZNA VRIJEDNOST 46](#_Toc66873636)

[IZLAZ IZ AKTIVACIJSKE FUNKCIJE 47](#_Toc66873637)

[FUNKCIJA GUBITKA KAO KOMPOZICIJA FUNKCIJA 47](#_Toc66873638)

[RAČUNANJE GRADIJENTA 48](#_Toc66873639)

[POVRAT U POVRATNOJ PROPAGACIJI 51](#_Toc66873640)

[SPAJANJE IZRAZA 54](#_Toc66873641)

[PROSJEČNA DERIVACIJA FUNKCIJE GUBITKA 54](#_Toc66873642)

[NESTAJUĆI I EKSPLODIRAJUĆI GRADIJENT 55](#_Toc66873643)

[PARAMETRI I HIPERPARAMETRI U UMJETNIM NEURONSKIM MREŽAMA 56](#_Toc66873644)

[PARAMETRI 56](#_Toc66873645)

[HIPERPARAMETRI 58](#_Toc66873646)

# DUBOKO UČENJE I UMJETNE NEURONSKE MREŽE

U ovom poglavlju će se generalno upoznati s pojmovima koji su bitni za razumijevanje ovog diplomskog rada. Prvo će se objasniti što je to duboko učenje, što je motivacija dubokog učenja i gdje se ono danas primjenjuje. Nakon toga će se opisati što su tradicionalne ili standardne umjetne neuronske mreže kako bi se stvorila podloga za upoznavanje s konvolucijskim neuronskim mrežama koje su glavna tema ovog rada. Potom će se dati kratki, općeniti pregled rada i strukture umjetnih neuronskih mreža kako bi se opisalo kako točno podaci prolaze kroz umjetnu neuronsku mrežu. Detaljan način rada će se opisati u jednom od sljedećih poglavlja. Ovo poglavlje će nakratko usporediti te dvije vrste mreža i dati razlog zašto su se konvolucijske mreže uopće razvile i zašto se koriste.

## ŠTO JE DUBOKO UČENJE

Duboko učenje (eng. *deep learning*) je područje strojnog učenja koje 'uči' računala ono što ljudi rade po svojoj prirodi: uče na primjerima. Ono je jedna od ključnih tehnologija iza autonomnih vozila. Čini ih sposobnima prepoznati znak stop ili razlikovati pješaka od uličnog svjetla. [8]

On uči računalni model kako ispravno klasificirati podatke iz vizualnih, tekstualnih ili audio podataka te u nekim situacijama modeli dubokog učenja pokazuju bolje rezultate od ljudi. Ti modeli su trenirani korištenjem velikog skupa označenih podataka što je značajka nadziranog učenja (eng. *supervised learning*). [8]

Duboko učenje se prvi put spominje tijekom 1980.-ih, ali samo u teoretskom smislu. Dva su razloga zašto je to tako:

1. Duboko učenje zahtijeva velike količine označenih podataka. Na primjer, razvoj autonomnog vozila zahtijeva milijune slika i tisuće sati videa.
2. Duboko učenje zahtijeva znatnu računalnu moć. Grafičke kartice visokih performansi s mogućnošću paralelnog izvođenja pokazale su se efikasnima u dubokom učenju. Kada se one udruže s računarstvom u oblaku (eng. *cloud computing*), znatno se smanjuje vrijeme potrebno za treniranje dubokih neuronskih mreža. Govori se o prvotnom vremenu od nekoliko tjedana do konačnog vremena od nekoliko sati ili manje. [2]

Širok je spektar industrija u kojima se primjenjuje duboko učenje. Neke od tih industrija su:

1. industrija autonomnih vozila
2. zračna industrija
3. medicinska industrija
4. industrijska automatizacija
5. marketinška industrija
6. robotika [2]

## UVOD U UMJETNE NEURONSKE MREŽE

Modeli dubokog učenja nazivaju se 'umjetne neuronske mreže' (eng. *artificial neural networks*). Umjetna neuronska mreža je računalni model inspiriran strukturom i načinom rada biološke neuronske mreže (kao što je ljudski mozak). Umjetne neuronske mreže se sastoje od velikog broja međusobno povezanih jednostavnih jedinica koje se zovu neuroni, čvorovi ili perceptroni (eng. *neurons, nodes, perceptrons*). [1][2]

Svaki neuron može donijeti jednostavnu odluku i proslijediti ju neuronima s kojima je povezan preko veze (eng. *connection*). Neuroni su organizirani u slojeve (eng. *layers*). Više međusobno povezanih slojeva tvori umjetnu neuronsku mrežu. Samo neuroni, koji pripadaju različitim susjednim slojevima, mogu biti međusobno povezani vezama.[1][5]

Umjetna neuronska mreža tako može oponašati skoro bilo koju moždanu funkciju i tako praktički odgovoriti na bilo koje pitanje nakon što se istrenira na dovoljnom broju uzoraka za treniranje. [5]

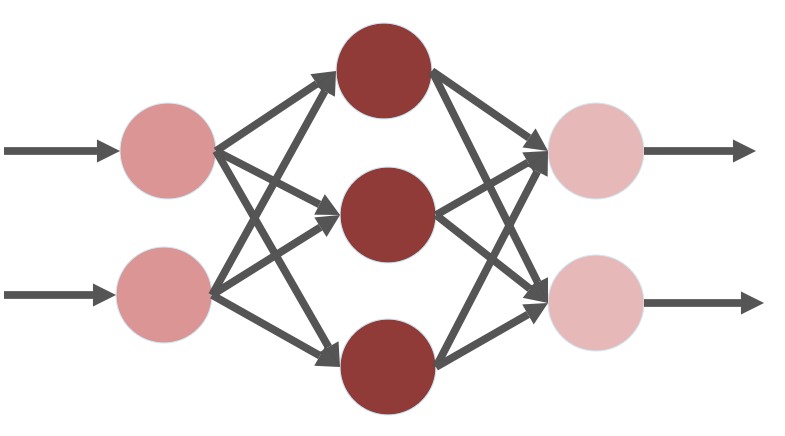
Umjetna neuronska mreža se u literaturi može naći i pod nazivima:

1. mreža (eng. *net*)
2. neuronska mreža (eng. *neural net*)
3. model

### SLOJEVI U NEURONSKOJ MREŽI

'Plitka' umjetna neuronska mreža, kakva je prikazana na slici, sadrži samo tri sloja:

1. ulazni sloj (eng. *input layer*) koji prima neovisne varijable ili ulaze u model
2. jedan skriveni sloj (eng. *hidden layer*) koji se nalazi između ulaznog i izlaznog sloja
3. izlazni sloj (eng. *output layer*) koji prikazuje predviđanja modela. [5]



Slika 1: prikaz plitke, potpuno povezane umjetne neuronske mreže

Različiti slojevi izvode različite vrste transformacija na svojim ulazima korištenjem aktivacijskih funkcija (eng. *activation function*). Primjeri (ili podaci) se prosljeđuju mreži preko ulaznog sloja, obično u obliku nekog višedimenzionalnog vektora. Broj čvorova u ulaznom sloju jednak je broju dimenzija ulaznog primjera. Kada bi se kao primjer uzela gore prikazana slika, broj dimenzija ulaznog primjera bi bio dva. Ulazni sloj te podatke, takvi kakvi jesu prosljeđuje skrivenim slojevima.[1][8]

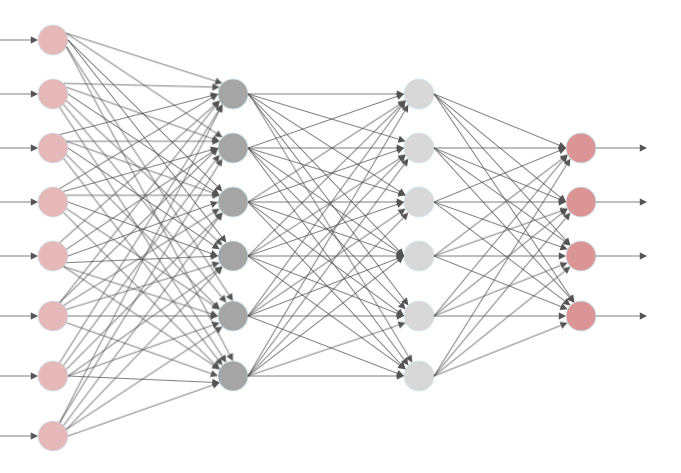
Skriveni slojevi donose određene odluke na temelju podataka koje su dobili na svojim ulazima od prethodnog sloja i na temelju aktivacijske funkcije koju koriste. Te odluke prosljeđuju sljedećem sloju kao vlastite izlaze. Proces se ponavlja sve dok se ne dosegne izlazni sloj. Broj čvorova u skrivenom sloju je proizvoljan u smislu da nije uvjetovan strukturom ulaznih primjera već proizvoljno odabranim brojem mogućih klasa.[8]

Izlazni sloj izvodi konačnu operaciju kako bi prikazao predviđanja modela. Broj čvorova u izlaznom sloju jednak je broju klasa predviđanja (eng. *prediction classes*). Ako se kao primjer uzme gornja slika, broj klasa predviđanja, odnosno izlaznih čvorova, bi bio broj dva. [8]

Neka se za primjer uzme da je mreža trebala donijeti odluku je li određena osoba bolesna na temelju rezultata bolničkih pretraga. Jedan čvor bi prikazivao koliko je neuronska mreža uvjerena da je dana osoba bolesna, dok bi drugi čvor prikazivao koliko je neuronska mreža uvjerena da je dana osoba zdrava.

Duboka neuronska mreža posjeduje sličnu strukturu. Razlika je u tome što duboka neuronska mreža sadrži dva ili više 'skrivenih' slojeva koji obrađuju ulazne podatke.[5]

Primjer takve mreže prikazan je na slici.



Slika 2: prikaz strukture potpuno povezane duboke umjetne neuronske mreže

Iako se i plitke neuronske mreže mogu nositi sa složenim problemima, duboke neuronske mreže daju ispravnija predviđanja i ta ispravnost raste kako im se dodaje više skrivenih slojeva. Kod tradicionalnih dubokih umjetnih neuronskih mreža najčešće je optimalno imati do devet ili deset skrivenih slojeva. Pokazalo se da, dodavanjem više skrivenih slojeva od toga broja, preciznost tradicionalne neuronske mreže počinje opadati. Danas, duboke neuronske mreže imaju najčešće između tri i deset skrivenih slojeva.[5]

Obje slike prikazuju 'potpuno povezanu neuronsku mrežu' (eng. *fully-connected neural network*) s takozvanim 'gustim slojevima' (eng. *dense layers*). Taj naziv se odnosi na to da je neuron u određenom sloju povezan sa svakim neuronom iz susjednog sloja.

Postoje različite vrste slojeva u umjetnim neuronskim mrežama. Neke od vrsta su:

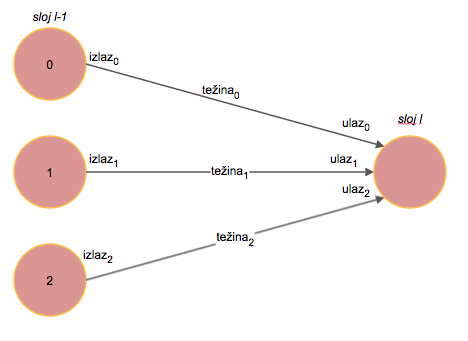
1. gusti (ili potpuno povezani) slojevi
2. konvolucijski slojevi (eng. *convolutional layers*)
3. slojevi sažimanja (eng. *pooling layers*)
4. slojevi povrata (eng. *recurent layers*)
5. slojevi za normalizaciju (eng. *normalization layers*)

Razlog, zbog kojeg postoji više vrsta skrivenih slojeva, je taj što različiti slojevi izvode različite transformacije na svojim ulaznim podacima te su stoga određene vrste slojeva prikladnije za pojedini zadatak u odnosu na druge slojeve. [1]

### VEZE IZMEĐU SLOJEVA

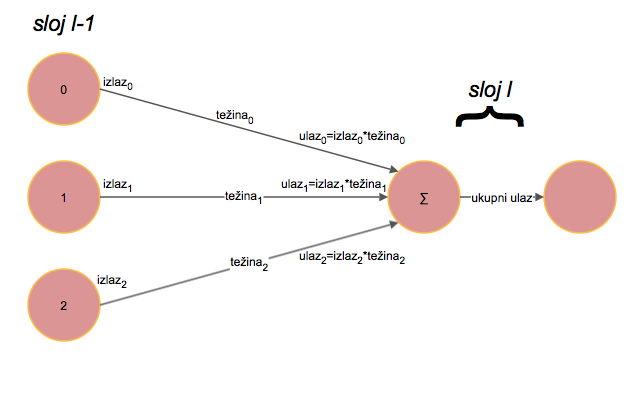
Kao što je već spomenuto, neuroni, koji se nalaze u različitim slojevima, međusobno su povezani vezama. Neuroni, koji pripadaju istom sloju, ne mogu biti međusobno povezani.

Svakoj se vezi između dva neurona pridružuje određena težina (eng. *weight*) u obliku određene brojčane vrijednosti. Svaka težina predstavlja snagu veze između dva određena čvora. Kada mreža na ulazu dobije neku ulaznu vrijednost (eng. *input value*), ona se pridružuje određenom čvoru u ulaznom sloju. Nadalje, kada se ta ulazna vrijednost prosljeđuje određenim neuronima u sljedećem, ovdje skrivenom, sloju, ona će biti pomnožena s težinom koja je pridružena određenoj vezi. Ta nova vrijednost, koja se prosljeđuje neuronu u sljedećem sloju, čini izlaznu vrijednost prethodnog neurona (eng. *output value*).[1]



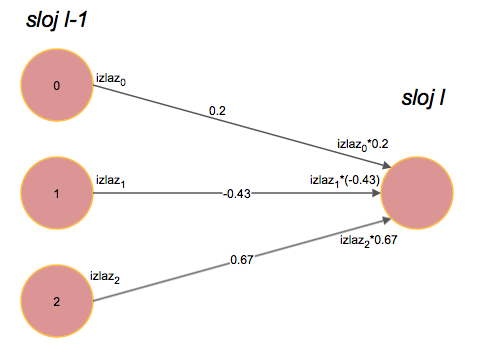
Slika kojom se ilustrira što je izlaz iz neurona i što je ulaz u neuron

Čvor u ovdje skrivenom sloju, na svoj ulaz prima više ulaznih vrijednosti iz više čvorova koji se nalaze u prethodnom sloju. Konačnu će ulaznu vrijednost u određeni čvor činiti ponderirana suma (eng. *weighted sum*)svakog pojedinog ulaza u dani čvor.



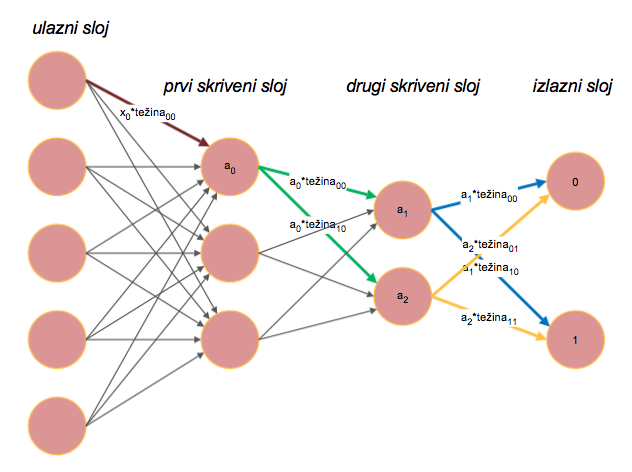
Tako, izlazna vrijednost iz danog čvora može biti izražena na sljedeći način:

Težine veza su inicijalizirane nasumičnim brojčanim vrijednostima i čine jedan od parametara koje mreža treba naučiti tijekom procesa učenja kako bi što ispravnije preslikavala dani ulazni primjer u ispravnu klasu predviđanja.



### PROSLJEĐIVANJE PREMA NAPRIJED

Nastavljajući primjer iz prethodnog poglavlja, jednom kada se dobije izlazna vrijednost za dani čvor, taj dobiveni izlaz se pojedinačno množi sa svakom težinom veze koja povezuje dani čvor sa čvorovima koji se nalaze u sljedećem sloju. Te nove vrijednosti se prenose kao ulazne vrijednosti odgovarajućim čvorovima.



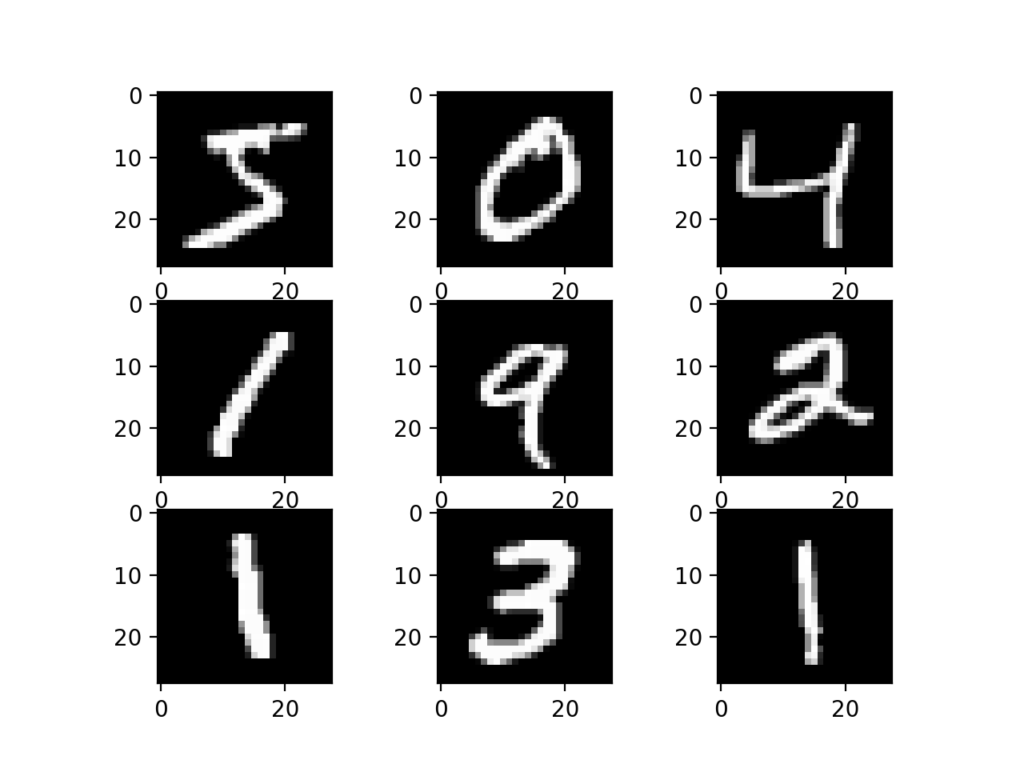
*Slika koja pokazuje kako jedan čvor ima utjecaja na ostale čvorove u mreži*

Taj se opisani proces ponavlja sve dok se ne dosegne izlazni sloj s klasama predviđanja. Ako se za primjer uzme jedna od prije prikazanih slika, može se primijetiti da se proces ponavlja od ulaznog sloja prema izlaznom sloju, odnosno s lijeva na desno. Tako se cijeli proces prolaska podataka kroz umjetnu neuronsku mrežu naziva 'prosljeđivanje prema naprijed'. (eng. *forward pass*).

## KONVOLUCIJSKE NEURONSKE MREŽE

Jedno od najvećih ograničenja tradicionalnih umjetnih mreža jest da se počinju 'mučiti' pod računalnom složenošću koju zahtijeva obrada slika. Uobičajeni skupovi podataka za strojno učenje, kao što je MNIST skup podataka i koji sadrži slike ručno pisanih znamenki dimenzija 28x28, prikladni su za većinu tipova umjetnih mreža. Neuron bi u prvom skrivenom sloju tradicionalne umjetne mreže trebao sadržavati prihvatljiv broj od 784 veze (28x28x1).[8]

*Slika s primjerima iz MNIST-a*



Izvor: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/

Međutim, kada bi tradicionalna neuronska mreža trebala obraditi RGB sliku dimenzija 64x64, neuron u prvom skrivenom sloju bi imao 12,288 veza. Tradicionalna mreža u tom slučaju ne bi raspolagala dovoljnom računalnom snagom i bila bi sklona prenaučenosti (eng. *overfitting*).[8]

Drugim riječima, struktura potpuno povezane neuronske mreže pokazala se neefikasnom tamo gdje je potrebno analizirati slike (ili podatke) velikih dimenzija. Konvolucijska neuronska mreža (eng. *convolutional neural network*) koristi strukture kod kojih neuroni u jednom sloju ne komuniciraju nužno sa svim neuronima u sljedećem sloju. Umjesto toga, svaki skup neurona analizira malo područje ili značajku slike. [4]

Pristup korištenja konvolucijskih mreža temelji se na pretpostavci da model može ispravno funkcionirati na temelju lokalnog razumijevanja slike. Konvolucijska mreža koristi manje parametara u odnosu na potpuno povezane mreže tako što ponovno iskorištava parametre i po nekoliko puta (eng. *parameter sharing*). Dok tradicionalna potpuno povezana neuronska mreža generira težinu veze za svaki piksel na slici, konvolucijska neuronska mreža generira tek toliko težina kako bi mogla skenirati malo područje na slici u danom trenutku.[4]

Ovaj je pristup koristan tijekom procesa učenja – što mreža posjeduje manje parametara, to će pokazivati bolju izvedbu i brže će učiti.[4]

Konvolucijske neuronske mreže primarno su bile korištene kako bi se riješili problemi raspoznavanja uzoraka ili struktura na slikama. One uče kako detektirati različite strukture (eng. *features*) na ulaznim primjerima korištenjem na desetke ili čak stotine skrivenih slojeva. Svaki sloj u konvolucijskoj mreži povećava kompleksnost strukture koju model uči raspoznavati. Na primjer, skriveni sloj na početku konvolucijske mreže uči detektirati rubove i svjetlinu, dok posljednji sloj uči detektirati strukture koje su specifične za objekt kojeg model treba moći znati prepoznati (kao na primjer raspoznavanje lica). [2][5][8]

Konvolucijske mreže su po strukturi analogne standardnim umjetnim neuronskim mrežama na način da se sastoje od neurona, veza i slojeva. I dalje će neuron primiti neki ulaz, izvesti određenu operaciju i dobivenu vrijednost proslijediti dalje kroz mrežu.[8]

Ulazi u konvolucijski sloj se još mogu nazivati ulaznim kanalima (eng. *input channels*), a izlazi se mogu nazivati izlaznim kanalima (eng. *output channels*). [1]

Međutim, skriveni slojevi konvolucijske neuronske mreže izvode operacije koje transformiraju ulazne podatke s namjerom da model nauči strukture koje su specifične za pojedine ulazne primjere. Tri najčešće korištena sloja su:

1. Konvolucijski sloj: obrađuje ulaznu sliku koristeći skup konvolucijskih filtera. Svaki od filtera aktiviran je od strane određene strukture na slici.
2. Aktivacijski sloj: aktivira značajke koje se prosljeđuju sljedećem sloju u mreži.
3. Sloj sažimanja (eng. *pooling layer*): pojednostavljuje izlaz tako što izvodi ne-linearno uzorkovanje i tako reducira broj parametara koje mreža treba naučiti, ali u isto vrijeme čuva najbitnije informacije.

Ove operacije se ponavljaju kroz desetke ili stotine slojeva. Svaki sloj uči detektirati različite strukture ili značajke.[5]

Konvolucijske neuronske mreže su popularne zbog tri glavna razloga:

1. eliminiraju potrebu manualnog izvlačenja struktura i značajki – značajke se uče direktno od strane konvolucijske mreže,
2. daju visoku kvalitetu raspoznavanja objekata,
3. mogu se ponovno istrenirati za nove zadatke prepoznavanja i tako omogućuju razvoj nove mreže na temelju već postojeće.[5]

Konvolucijske mreže su osobito korisne za pronalaženje struktura na slikama s ciljem prepoznavanja objekata, lica i krajolika. Također su izrazito djelotvorne i kod klasificiranja ne-vizualnih podataka kao što su audio podaci, podaci s vremenskim serijama ili signalima. [4][5]

# STRUKTURA JEDNOSTAVNE KONVOLUCIJSKE NEURONSKE MREŽE

Konvolucijska neuronska mreža je specijalizirana vrsta neuronskih mreža koja je dizajnirana za rad s 2D podacima (slike i fotografije), iako ih se može koristiti i kod 1D ili 3D podataka. Središte konvolucijskih neuronskih mreža čine konvolucijski slojevi.[7]

## ULAZNI SLOJ

Općenito, ulazni sloj prima neovisne ulazne primjere. Ulazni primjeri su podaci koji se prosljeđuju neuronskoj mreži i na temelju kojih ona donosi određenu odluku ili predviđanje. Svaka ulazna vrijednost (vrijednosti koje sačinjavaju ulazni primjer) prosljeđuje se jednom neuronu u ulaznom sloju neuronske mreže. Za razliku od ostalih slojeva u bilo kojoj neuronskoj mreži, ulazni sloj ne izvodi nikakve operacije nad ulaznim primjerima već on služi samo zato da primi ulazne podatke takvima kakvi oni jesu i da ih proslijedi mreži. [5]

Broj neurona u ulaznom sloju uvjetovan je samom strukturom primjera koji se prosljeđuju neuronskoj mreži. On odgovara broju jedinstvenih značajki, ili broju piksela, ulaznih podataka.

Jedna od glavnih razlika između tradicionalne neuronske mreže i konvolucijske neuronske mreže je ta da su neuroni u konvolucijskoj mreži organizirani u tri dimenzije (visina, širina i dubina). U konvolucijskom neuronskim mrežama, neuroni ulaznog sloja sadrže vrijednosti piksela ulazne slike.[8]

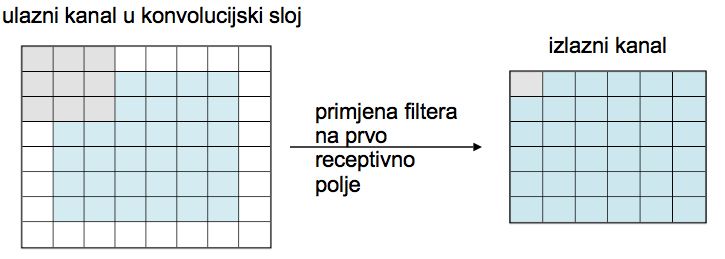
Zato je potrebno, prije nego se bilo koja slika proslijedi konvolucijskoj neuronskoj mreži, ulazne podatke prvo pred-procesirati kako bi se osiguralo da će mreža uvijek na svoj ulaz primati podatke istih dimenzija.

Kao što je već spomenuto, ulazni sloj konvolucijske neuronske mreže najčešće prima slikovne podatke. Slikovni podaci su, kao što je maloprije spomenuto, prikazani 3D matricom. Tu matricu je, prije nego se proslijedi mreži, potrebno transformirati u vektor, odnosno u matricu s jednim stupcem i više redaka. Neka ulazna slika ima dimenzije 28x28=784 i neka je crno-bijela ('dubina' joj je 1). Tu sliku je potrebno transformirati u vektor 784x1. To znači da će ulazni sloj u konvolucijsku neuronsku mrežu imati onoliko piksela koliko ih ima ulazna slika u sve svoje tri dimenzije.[16][17]

## KONVOLUCIJSKI SLOJ

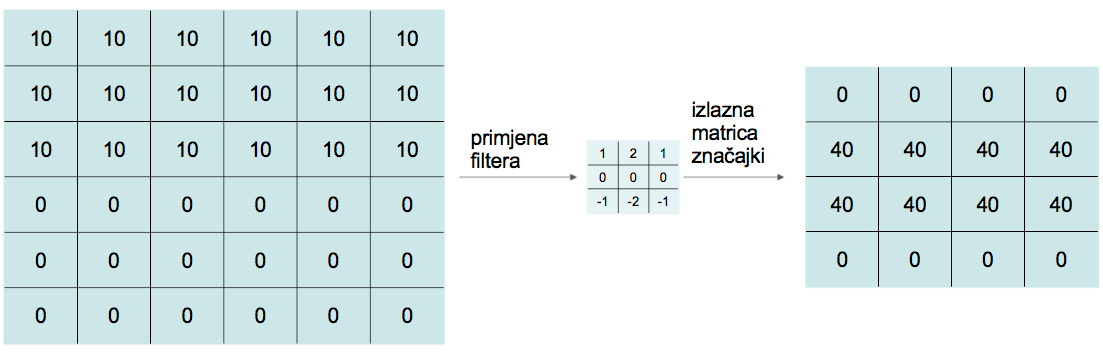
Računalo 'čita' slike kao matrice s vrijednostima piksela koje imaju dimenzije NxNx3 (visina, širina, dubina). Slike se najčešće sastoje od 3 kanala (RGB). U kontekstu konvolucijskih neuronskih mreža, konvolucija je linearna operacija koja uključuje množenje skupa težina (filtera) sa skupom odgovarajućih ulaza (matrica s vrijednostima piksela).[5][10]

Ulaz u konvolucijski sloj se još naziva ulaznim kanalom (eng. *input channel*). Izlaz iz konvolucijskog sloja se naziva izlaznim kanalom (eng. *output channel*), matricom značajki (eng. *feature map*) ili matricom aktivacija (eng. *activation map*). [1][7]



### FILTERI

Svaki konvolucijski sloj u konvolucijskoj neuronskoj mreži posjeduje definirani broj filtera. Ti filteri se sustavno primjenjuju na ulazni kanal i kreiraju matricu značajki. [1][7][9]



Ulazni kanal je obično 3D slika (broj redaka, stupaca i kanala). Tako i filteri moraju imati 3 dimenzije s jednakim brojem kanala, ali manjim brojem stupaca i redaka u odnosu na ulaznu sliku. Filter sadrži težine (u tradicionalnim neuronskim mrežama bi to bile uobičajene težine veza) čije optimalne vrijednosti mreža treba naučiti tijekom procesa učenja. Težine filtera definiraju strukturu ili obrazac (eng. *feature*) koji će filter detektirati na svom ulaznom kanalu. [9]

Slika se matematički može izraziti kao matrica sljedećih dimenzija:

Gdje je:

* : visina slike
* : širina slike
* : broj kanala

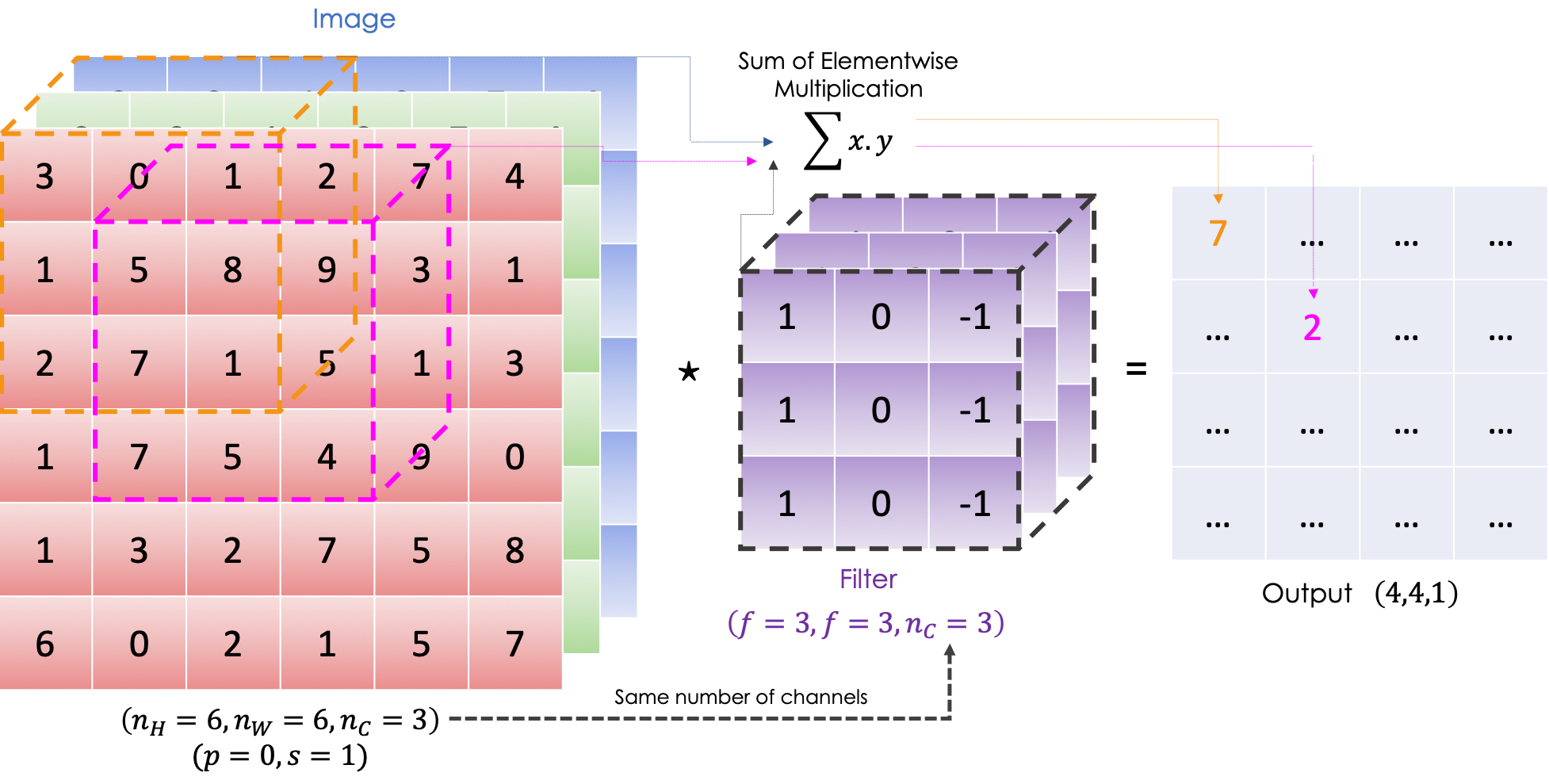
Za RGB sliku, broj kanala je jednak 3 s obzirom da RGB slika posjeduje crveni, zeleni i plavi kanal. [11]

Filter, koji se označava s *K*, najčešće ima jednake dimenzije visine i širine (na primjer 3x3, 5x5 ili 7x7) koje se označavaju s *f*. Takav filter, s neparnim brojem za dimenzije, omogućava da se svaki piksel na slici u jednom trenutku nalazi u sredini filtera. Filter mora imati jednak broj kanala kao i ulazna slika.[11]

Dimenzije filtera se tako označavaju na sljedeći način:

Konvolucijski produkt slike i filtera je 2D matrica značajki. Svaki element na toj matrici je rezultat sume množenja po odgovarajućim elementima (eng. *elementwise multiplication*). Filteri sadrže težine koje su inicijalizirane nasumičnim brojčanim vrijednostima. Filter u danom trenutku računa točkasti produkt (eng. *dot product*) na 'kocki' (jer posjeduje visinu, širinu i dubinu) ulazne matrice, odnosno slike. Kocka ima iste dimenzije kao i filter. Ta 'kocka' slike koju filter promatra u danom trenutku naziva se '*patch'* ili 'receptivno polje' (eng. *receptive field*).[11][12]

Primjer jednog konvolucijskog produkta prikazan je na slici:



Konvolucijska operacija se matematički opisuje na sljedeći način:

Gdje je:

* *I*: ulazna slika
* *K*: filter
* *x*: redak u matrici značajki gdje će se spremiti produkt
* *y*: stupac u matrici značajki gdje će se spremiti produkt
* *i*: redni broj retka ulazne matrice
* *j*: redni broj stupca ulazne matrice
* *k*: redni broj kanala ulazne matrice

Dimenzije matrice značajki se dobiju na sljedeći način:

Gdje je pod (eng. *floor*) funkcija od *x*.

Postoji nekoliko posebnih vrsti konvolucije:

1. *Valid* konvolucija:
2. *Same* konvolucija: veličina izlaza je jednaka veličini ulaza:
3. 1x1 konvolucija: ; ovo je korisno u nekim slučajevima kako bi se smanjio broj kanala bez da se smanje dimenzije ().

Konvolucijski sloj izvodi konvolucijske operacije na svojim ulazima tako što istovremeno primjenjuje više filtera. Izlazi iz konvolucijskog sloja se prosljeđuju aktivacijskoj funkciji koja se u ovom primjeru označava s .

Ovdje je potrebno napomenuti da literatura nije usuglašena pripadaju li aktivacijske funkcije konvolucijskom sloju ili čine zasebni, aktivacijski sloj. U ovom poglavlju će se opisati na koji se način aktivacijske funkcije primjenjuju na rezultate konvolucije dok će aktivacijske funkcije biti obrađene u zasebnom poglavlju.

U *l*-tom sloju neuronske mreže vrijede sljedeće anotacije:

* ulaz u sloj: dimenzija gdje je slika koja ulazi u neuronsku mrežu.
* *padding*: i pomak:
* broj filtera: (broj kanala u tom sloju) gdje svaki (oznaka za filter) ima dimenzije
* prag *n*-te konvolucije:
* aktivacijska funkcija:
* izlaz iz sloja: s dimenzijama

Sada vrijedi:

:

I nadalje:

S tim da:

Parametri u *l*-tom sloju koji se mogu naučiti su:

* filteri s parametara
* prag s parametara

### ZERO PADDING

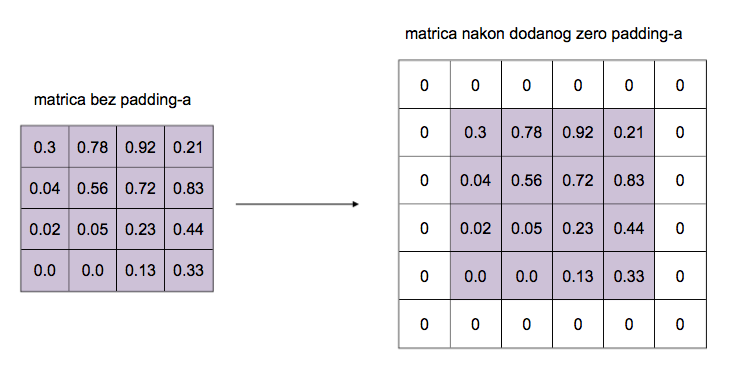
Svaki konvolucijski sloj posjeduje definirani broj filtera određenih dimenzija. Sustavnom primjenom filtera, dimenzije matrice značajki su reducirane u odnosu na dimenzije ulaznog kanala. [7]

Kao što je matematički opisano u prethodnom poglavlju, filter u konvolucijskom sloju se počinje primjenjivati na ulazni kanal tako što se gornji lijevi kut filtera poklapa s gornjim lijevim kutom ulaznog kanala. Nakon što izračuna točkasti produkt, filter se pomiče za jedno mjesto (ili u slučaju slike, piksel) u desno (jer se uzima u obzir da vrijedi osnovna situacija kada je pomak postavljen na jedan) te ponovno računa točkasti produkt. Cijeli proces se ponavlja sve dok se gornji desni kut filtera ne poklopi s gornji desnim kutom slike. Znači, filter se primjenjuje samo unutar slike, ne izvan nje i zbog toga dolazi do redukcije dimenzija.

Redukcija dimenzija matrice značajki u odnosu na ulazni kanal naziva se 'efekt ruba' (eng. *border effect*). Taj efekt ne predstavlja veliki problem kod velikih slika i malih filtera, iako se i dalje gube potencijalno važne informacije koje se mogu nalaziti na rubovima slike. Problem je puno veći kada se filter primjenjuje na slike malih dimenzija, kada je više konvolucijskih slojeva serijski povezano ili kod dubokih mreža s desetak ili stotinama skrivenih slojeva. U jednom trenutku mreža može ostati bez podataka u matrici značajki nad kojom bi mogla izvoditi operacije ili donositi odluke. [7][11][15]

Pikseli, koji se nalaze na rubovima ulaznog kanala (ili slike), bit će izloženi samo rubovima filtera i rjeđe se koriste od piksela koji se nalaze po sredini slike. Kada bi se filter primjenjivao tako da se njegov rub nalazi izvan okvira slike, pikseli, koji se nalaze na rubovima ulaznog kanala, imaju veću mogućnost utjecati na rezultat konvolucije. Drugim riječima, ostvaruje se više prilika za detekciju određenih struktura. Još jedna posljedica je da matrica značajki ima iste dimenzije kao i ulazni kanal.[7][11]

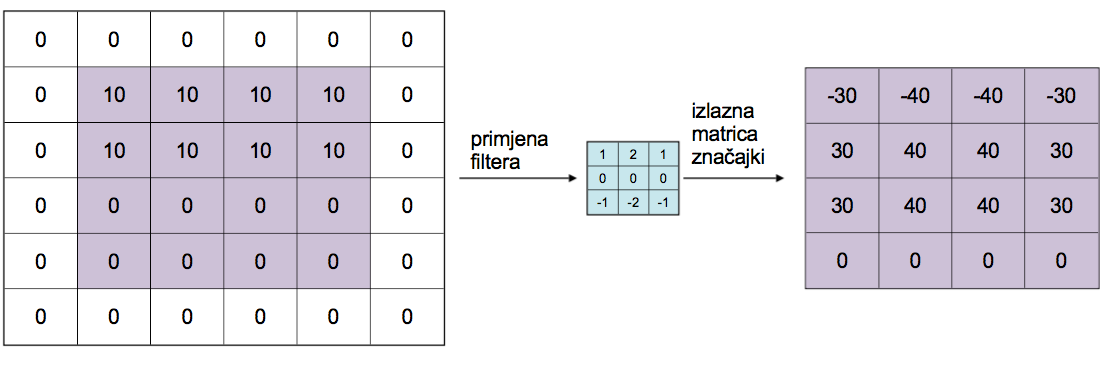
Kako bi se riješio taj problem 'odbacivanja' piksela na rubovima ulaznog kanala, oko rubova ulaznog kanala se dodaje neka vrsta popune (eng. *padding*). Dodaju se dodatni pikseli oko cijelog okvira ulaznog kanala. Prema konvenciji se taj *padding* puni nulama. Zato se naziva *zero padding*. *Padding* se označava s *p* i označava 'debljinu' *paddinga*-a koji je dodan na sve četiri strane ulaznog kanala. [11][15]



S obzirom na to koristi li se *padding* ili ne, postoje dvije vrste konvolucije: *same* konvolucija i *valid* konvolucija. *Valid* konvolucija znači da se konvoluira po originalnom ulaznom kanalu bez dodanog *padding*-a. *Same* konvolucija znači da se oko okvira slike stavlja dodatni *padding* s nulama tako da matrica značajki ima iste dimenzije kao i ulazni kanal. U drugom slučaju, 'debljina' *padding*-a se računa na sljedeći način:

Gdje su:

* *p*: 'debljina' obruba
* *f*: visina ili širina filtera. [1][11]



### POMAK

Kao što je već opisano, filter se po ulaznom kanalu pomiče s lijeva na desno, od gore prema dolje po jedan piksel horizontalno pa vertikalno. Broj piksela koji definira koliko se filter pomiče može varirati, ali je uvijek simetričan u horizontalnom i vertikalnom smjeru. Taj broj se naziva 'pomak' (eng. *stride*).[7][12]

Osnovni pomak je (1, 1) za pomicanje filtera po širini i visini ulaznog kanala i u većini slučajeva daje zadovoljavajuće rezultate. Ako se ne koristi pomak (1,1), onda se najčešće koristi pomak (2, 2). Na taj način se filter pomiče dva stupca u desno i dva reda dolje. Posljedica većeg pomaka je da će matrica značajki imati puno manje dimenzije od ulaznog kanala. Na primjer, ako se 3x3 filter s pomakom (1, 1) primjenjuje na 6x6 sliku, matrica značajki će imati dimenzije 3x3.[7]

## AKTIVACIJSKE FUNKCIJE U UMJETNOJ NEURONSKOJ MREŽI

Aktivnost u biološkoj neuronskoj mreži, gdje su različiti neuroni aktivirani različitim podražajima, bila je inspiracija za korištenje aktivacijskih funkcija u umjetnim neuronskim mrežama. U biološkoj neuronskoj mreži, neki neuroni su u danom trenutku ili aktivirani ili ne. Primjenom aktivacijske funkcije u umjetnoj neuronskoj mreži, aktivnost neurona može biti prikazana bilo kojim brojem između nula (ili minus jedan) i jedan (ili proizvoljno odabranoj gornjoj granici). Što je vrijednost neurona bliža nuli, to je neuron manje aktiviran. S druge strane, što je vrijednost neurona bliža jedinici, to je neuron više aktiviran. [1][5][9][19]

Aktivacijska funkcija u umjetnoj neuronskoj mreži jest funkcija koja preslikava ulaznu vrijednost, ponderiranu sumu svih ulaza određenog čvora, u odgovarajuću izlaznu vrijednost.

Aktivacijska funkcija na neki unaprijed određeni način transformira ulaznu vrijednost u broj koji se nalazi unutar unaprijed određene donje i gornje granice. Obično, umjetne neuronske mreže koriste ne-linearne aktivacijske funkcije jer su vrste preslikavanja kod dubokih neuronskih mreža složenije od običnih linearnih preslikavanja i zbog toga što ne-linearne aktivacijske funkcije omogućuju računanje proizvoljno složenih funkcija. Još jedna prednost aktivacijskih funkcija jest ta što aktivacijske funkcije dopuštaju povratnu propagaciju koja je objašnjena u drugom poglavlju. [9][10][19]

Obično su aktivacijske funkcije, koje se koriste u skrivenim slojevima, iste za sve skrivene slojeve u danoj neuronskoj mreži. Nije uobičajeno vidjeti ReLU aktivacijsku funkciju u jednom skrivenom sloju i sigmoidnu aktivacijsku funkciju u drugom skrivenom sloju. Koristi se ili samo ReLU aktivacijska funkcija ili samo sigmoidna aktivacijska funkcija u svim skrivenim slojevima dane umjetne neuronske mreže.

Kako bi se razumjelo zašto se koriste ne-linearne aktivacijske funkcije, i kako bi se poslije dokazalo da je jedna od aktivacijskih funkcija (ReLU) ne-linearna, prvo je potrebno razumjeti što linearne funkcije čini linearnima.

Neka je *f* funkcija na skupu *X*.

Neka su *a* i  *b* elementi skupa *X*.

Neka je *x* neki realan broj.

Funkcija *f* je linearna funkcija ako i samo ako vrijedi:

i ako vrijedi:

.

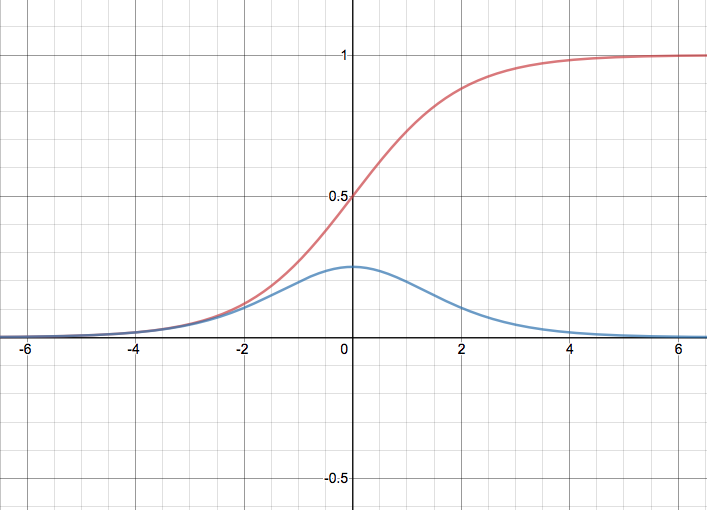
Važna značajka linearnih funkcija jest da kompozicija dviju linearnih funkcija čini novu linearnu funkciju. To znači da će, čak i u dubokim umjetnim neuronskim mrežama, ako se primjenjuju samo linearne funkcije nad ulaznim vrijednostima kod prosljeđivanja prema naprijed (eng. *forward propagation*), preslikavanje ulaza na izlaz uvijek biti linearno. Nadalje, kada bi svaki sloj u neuronskoj mreži koristio samo pragove (eng. *bias*) i težine (eng. *weights*) bez aktivacijskih funkcija, cijela mreža bi bila jednaka linearnoj kombinaciji težina i pragova. Drugim riječima, formula neuronske mreže bi se mogla faktorizirati i svesti na jednostavan linearni regresijski model. Takav model bi mogao rješavati jednostavne linearne ovisnosti, ali ne bi mogao obavljati konkretne zadatke neuronskih mreža kao što su obrada slika i zvukova. [1]

### SIGMOIDNA AKTIVACIJSKA FUNKCIJA

Sve sigmoidne funkcije imaju sposobnost preslikavanja ulaznog niza brojeva u izlazni niz brojeva koji se nalaze u malom opsegu između 0 i 1 ili -1 i 1. Tako se sigmoidna funkcija može koristiti za pretvorbu realnog broja u vrijednost koja se potom tumači kao vjerojatnost. Zbog toga se sigmoidna funkcija, osim u skrivenim slojevima, može koristiti i u izlaznom sloju gdje pretvara izlazni rezultat cjelokupne mreže u prikaze vjerojatnosti s kojima se onda može lakše raditi i koje se mogu lakše protumačiti. Zbog svog 'S' oblika na grafu nosi naziv sigmoidna funkcija. Taj naziv se često odnosi na naziv 'logistička sigmoidna funkcija'. [18][20]

Matematička formula sigmoidne funkcije glasi:

Grafički prikaz sigmoidne funkcije i njene derivacije prikazani su na slici.

**

*sigmoidna aktivacijska funkcija (crveno) i njena derivacija (plavo)*

Sigmoidna funkcija je ne-linearna, kontinuirano diferencirana, monotona funkcija. Glavna prednost joj je to što je jednostavna i dovoljno dobra kao funkcija za klasifikaciju. S druge strane, veliki nedostatak joj je što uzrokuje problem 'nestajućeg gradijenta' (eng. *vanishing gradient problem*) jer joj vrijednost nije centrirana oko nule. Zbog toga ažurirane vrijednosti mogu ići predaleko u različitim smjerovima. Nadalje, korištenjem sigmoidne funkcije, teže je provesti optimizaciju te računanje u skrivenim slojevima oduzima puno vremena. [18][20]

#### KORIŠTENJE SIGMOIDNE AKTIVACIJSKE FUNKCIJE

Glavno područje strojnog učenja, gdje je bitno korištenje sigmoidne funkcije, je područje korištenja logističkog regresijskog modela. Model logističke regresije se koristi kako bi se procijenila vjerojatnost na binarnom nivou, kao što je, na primjer, 'živo' ili 'neživo', 'lažno' ili 'istinito' i slično. Model vraća vrijednost koja se nalazi između 0 i 1. [20]

Razlog zašto se baš (logistička) sigmoidna funkcija koristi u logističkog regresiji je zbog činjenice da funkcija uvijek vraća vrijednost između 0 i 1, a logistička regresija je izvedena iz pretpostavke da su podaci iz obje klase normalno distribuirani.[20]

#### USPOREDBA SIGMOIDNE I ReLU FUNKCIJE

U modernim umjetnim neuronskim mrežama običaj je vidjeti, umjesto sigmoidne funkcije, ReLU aktivacijsku funkciju.

ReLU aktivacijska funkcija donosi nekoliko bitnih prednosti u odnosu na sigmoidnu funkciju. Glavna prednost jest da se ReLU funkcija puno brže izračunava. Nadalje, aktivacijski potencijal se u biološkim neuronskim mrežama ne mijenja za negativne ulaze. Zbog toga se čini da ReLU funkcija bolje oponaša biološke neuronske funkcije. [20]

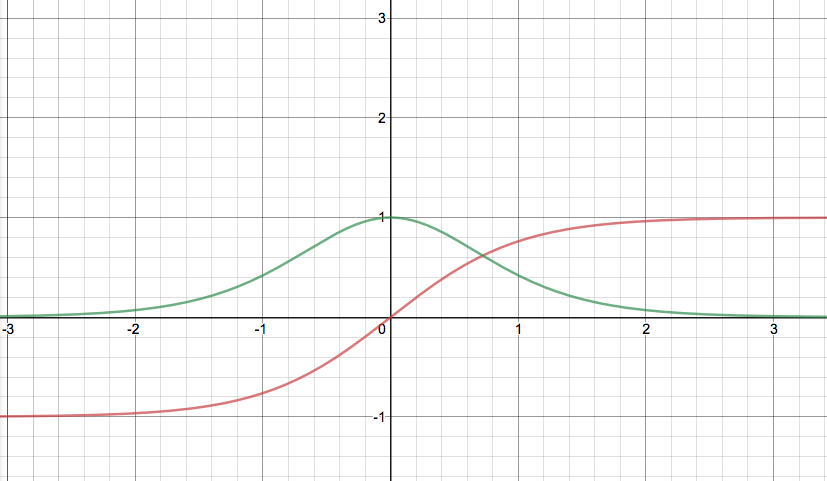
Još jedna u nizu prednosti se odnosi na pozitivne vrijednosti . ReLU funkcija posjeduje konstantni gradijent 1, dok sigmoidna funkcija posjeduje gradijent koji vrlo brzo konvergira prema 0. Ovo svojstvo neuronsku mrežu, koja sadrži sigmoidnu aktivacijsku funkciju, čini sporom za učenje. [20]

### HIPERBOLIČKO-TANGENTNA AKTIVACIJSKA FUNKCIJA

Hiperboličko-tangentna funkcija (tanh) je poboljšana sigmoidna funkcija. Opseg tanh funkcije je od -1 do 1 te, kao i sigmoidna funkcija, ima 'S' oblik. [18][19]

Matematička formula tanh funkcije glasi:

Grafički prikaz tanh funkcije i njene prve derivacije prikazan je na donjoj slici:



grafički prikaz tanh funkcije (crveno) i njene prve derivacije (zeleno)

Prednost tanh nad sigmoidnom funkcijom je što će negativne vrijednosti biti preslikane u izrazito negativne vrijednosti, a vrijednosti, koje imaju približnu vrijednost 0, bit će preslikane u vrijednosti koje su blizu 0 i tako rješava problem sigmoidne funkcije. Nadalje, pokazala se da daje bolje rezultate kod više-slojnih neuronskih mreža. Međutim, kao i sigmoidna funkcija, uzrokuje problem nestajućeg gradijenta, zbog čega se danas preferira korištenje ReLU funkcije.[19][21]

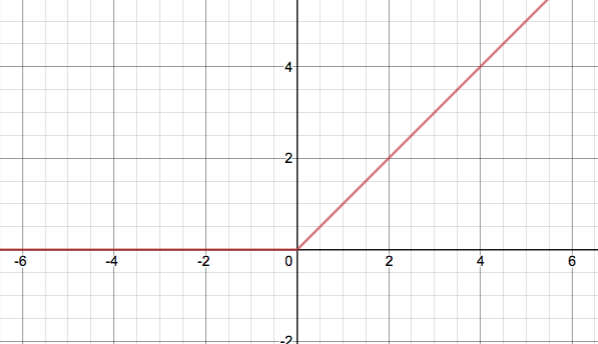
Ova funkcija se najčešće koristi za klasifikaciju podataka u dvije kategorije.

### ReLU AKTIVACIJSKA FUNKCIJA

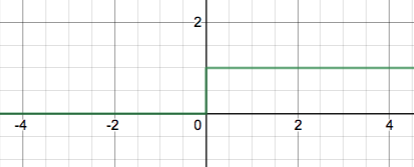
ReLU aktivacijska funkcija je kratica od *Rectified Linear Unit* i koja se izražava matematičkom formulom:

To je jedna od najboljih i najčešće korištenih aktivacijskih funkcija koje se danas koriste (pogotovo u konvolucijskim neuronskim mrežama) u odnosu na sigmoidnu i tanh funkciju jer se uspješno suočava s problemom nestajućeg gradijenta (eng. *vanishing gradient problem*). [22]

Grafički prikaz ReLU aktivacijske funkcije i njene derivacije nalaze se na slikama.



ReLU aktivacijska funkcija



Prva derivacija ReLU aktivacijske funkcije

Kao što se može vidjeti na slici, ReLU aktivacijska funkcija ponaša se kao linearna funkcija za sve pozitivne vrijednosti i vraća nulu za sve negativne vrijednosti. To znači nekoliko stvari:

1. Izlazna vrijednost ReLU funkcije se jednostavno računa. Zato će modelu trebati manje vremena za učenje i za kasnije donošenje odluka.
2. Brže konvergira. Linearnost ReLU funkcije znači da njen nagib nema 'visoravan' kada ulazna vrijednost *x* poprimi veliku vrijednost.
3. Slabo je aktivirana. S obzirom da ReLU poprima vrijednost nula za sve negativne ulazne vrijednosti, postoji vjerojatnost da se neki neuroni u mreži nikada neće aktivirati, što je u većini slučajeva dobro. [22]

U biološkim neuronskim mrežama, gdje postoji na milijarde neurona, nisu svi neuroni aktivni u isto vrijeme jer imaju različite uloge i aktiviraju ih različiti podražaji. Slaba aktiviranost u modelima rezultira bržim i boljim predviđanjima i manjoj prenaučenosti.[22]

#### DOKAZ DA JE ReLU NE-LINEARNA FUNKCIJA

Kako bi se dokazalo da je ReLU aktivacijska funkcija ne-linearna funkcija, dokazat će se da ReLU ne uspijeva biti linearna funkcija. Kao što je već spomenuto, aktivacijske funkcije su ne-linearne funkcije.

Za svaki realan broj *x*, funkcija *f* je definirana tako da je:

Neka se pretpostavi da je *a* realan broj za koji vrijedi .

Koristeći pretpostavku da vrijedi , može se vidjeti da je:

i da je:

Radi ovoga dolazimo do zaključka:

Tako je dokazano da funkcija *f*, odnosno ReLU funkcija, ne uspijeva biti linearna funkcija. [1]

#### UMIRUĆI ReLU

Kao što je već spomenuto, slaba aktiviranost ReLU funkcije je u većini slučajeva poželjna jer to rezultira bržim i boljim rezultatima te manjim rizikom pojave prenaučenosti. Međutim, negativna strana toga što ReLU poprima vrijednost nula za sve negativne vrijednosti je ta što se može pojaviti problem koji se naziva 'umirući ReLU' (eng. *dying ReLU*). [22][18]

ReLU neuron je 'mrtav' ako je ostao zaglavljen na negativnoj strani vrijednosti zbog čega uvijek ima vrijednost nula. Kako je nagib ReLU funkcije za negativne vrijednosti nula, jednom kada neuron poprimi negativnu vrijednost, malo je vjerojatno da će se oporaviti od te vrijednosti. Takvi neuroni ne pridonose preslikavanju ulaza na izlaz te postaju praktički beskorisni. Tijekom vremena se može dogoditi da veliki dio mreže ne radi ništa (u nekim slučajevima čak oko 40%).[22][18][23]

Zbog toga se, osim ReLU funkcije, znaju koristiti njene izvedenice koje nastoje zaobići taj problem. Neke od njih će biti opisane u zasebnim poglavljima. Neke od izvedenica su:

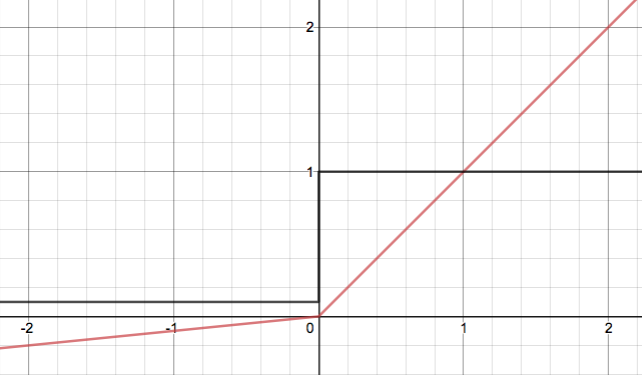
1. Leaky ReLU
2. Parametric ReLU (PReLU)
3. ReLU-6
4. Concatenated ReLU (CReLU). [22]

### LEAKY ReLU AKTIVACIJSKA FUNKCIJA

*Leaky* ReLU aktivacijska funkcija je izvedenica ReLU funkcije. Izvedena je s ciljem da riješi problem 'umirućeg ReLU-a'. Ona, za razliku od ReLU funkcije, ima blagi pozitivni nagib u negativnom dijelu vrijednosti i tako omogućuje izvedbu povratne propagacije i za negativne vrijednosti. To znači da je opseg *leaky* ReLu funkcije . [22][19][18]

Matematička formula *Leaky* ReLU funkcije glasi:

Grafički prikaz *leaky* ReLU funkcije kao i njene prve derivacije prikazan je na donjoj slici:



LeakyReLU (crveno) i prva derivacija (crno), gdje je parametar a postavljen na 0.1

Međutim, *leaky* ReLU ne nudi konzistentna predviđanja za negativne vrijednosti. Tijekom propagacije prema naprijed, ako je stopa učenja postavljena na preveliki broj, uzrokovat će *overshooting* i tako 'ubiti' neuron. [22][19]

Zbog toga se ideja *leaky* ReLU-a može i dalje proširiti. Umjesto da se *x* pomnoži nekim konstantnim koeficijentom, može se pomnožiti hiper-parametrom kojeg mreža može naučiti tijekom procesa učenja. Ova izvedenica *leaky* ReLU-a je poznata kao *parametric* ReLU.

### SWISH AKTIVACIJSKA FUNKCIJA

Aktivacijske funkcije imaju dugu povijest. Prvo se koristila sigmoidna funkcija zbog svoje jednostavne derivacije, opsega između 0 i 1 te zbog svog probabilističkog oblika. Tanh funkcija se smatrala zamjenom za 'običnu' sigmoidnu funkciju jer je ulazne vrijednosti preslikavala u vrijednosti koje se nalaze između -1 i 1. Međutim, te dvije funkcije su u većini slučajeva zamijenjene ReLU aktivacijskom funkcijom. [24]

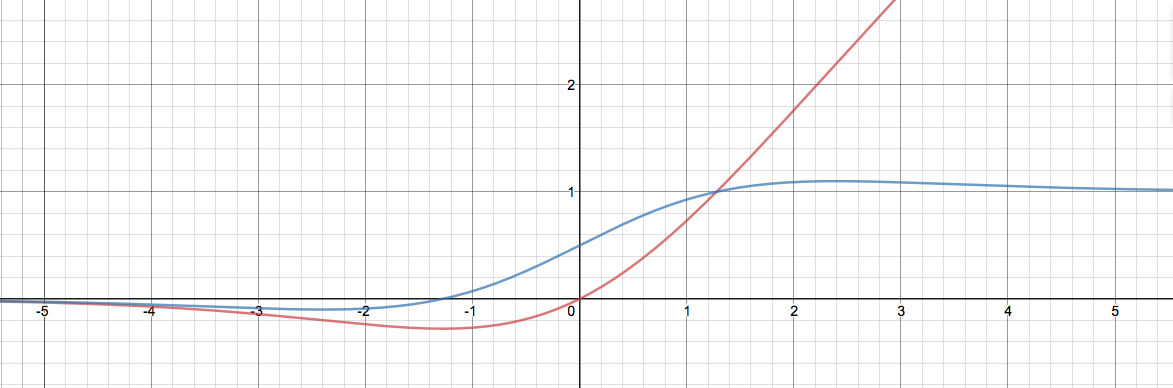
Kao i ReLU, Swish, aktivacijska funkcija razvijena od strane Google razvojnog tima, je omeđena s donje strane. To znači da, kako se vrijednost približava negativnoj vrijednosti, se približava nekoj konstantnoj vrijednosti. Isto tako je, kao i ReLU, neograničena s gornje strane (kako se vrijednost približava pozitivnoj vrijednosti, se približava nekoj beskonačnoj vrijednosti). Ali, za razliku od ReLU funkcije, Swish je 'glatka', odnosno nema iznenadne promjene gibanja. [24]

Matematička formula Swish aktivacijske funkcije glasi:

odnosno:

Derivacija Swish funkcije je onda:

Grafički prikaz Swish funkcije kao i grafički prikaz njene derivacije nalaze se na slikama.

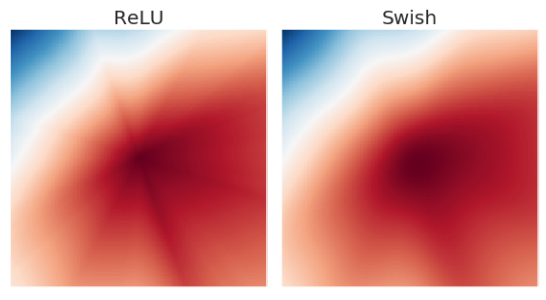


Swish derivacija (crveno) i njena prva derivacija

#### SVOJSTVA SWISH FUNKCIJE

Neomeđenost funkcije je poželjna kod svake aktivacijske funkcije jer se time izbjegava dugo vrijeme učenja s vrijednostima gradijenta koje su blizu 0, kao što su sigmoidna i tanh funkcije koje su ograničene s gornje i donje strane i zbog čega se mreže moraju oprezno inicijalizirati.[24]

Imati funkciju koja je omeđena s donje strane, kao što je ReLU funkcija, daje prednost tijekom procesa učenja jer se odbacuju velike negativne ulazne vrijednosti. Nadalje, uglađenost funkcije pomaže kod optimiziranja i generalizacije neuronske mreže. Na dolje prikazanoj slici uspoređene su ReLU i Swish funkcija gdje se vidi kako Swish funkcija ima bolju uglađenost. [24]



Slika 7: prikaz uglađenosti ReLU i Swish aktivacijske funkcije

Pejzažna glatkoća direktno korelira s pejzažem greški, što je on glađi, to je lakše pronaći minimum. [24]

Eksperimenti, koje je proveo Googleov razvojni tim, pokazali su da Swish funkcija daje bolje rezultate od ReLU funkcije kod dubokih neuronskih mreža. Na primjer, samo zamjenom ReLU funkcije Swish funkcijom, poboljšana je ispravnost klasifikacije ImageNet baze podataka za 0.9% i NASANetA za 0.6%. Nadalje, Swish je pokazala bolje izvođenje od ReLU kod dubokih mreža koje posjeduju između 40 i 50 slojeva, kada optimizacija postaje teška. Što se tiče veličine grupa (eng. *batch*), obje funkcije bilježe pad u kvaliteti izvođenja, ali i tada Swish ipak ima bolju izvedbu od ReLU funkcije. [23]

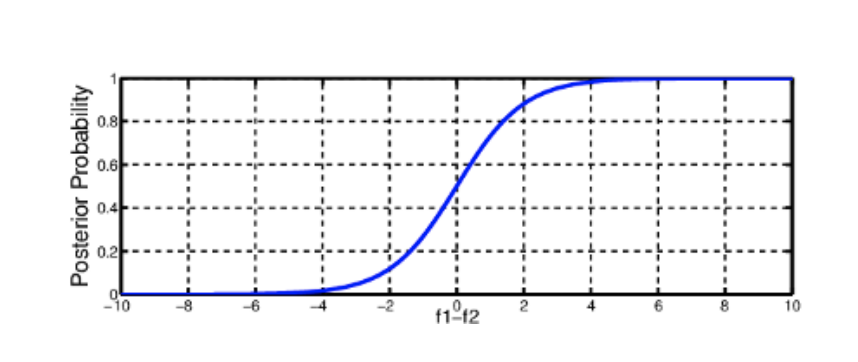
### SOFTMAX AKTIVACIJSKA FUNKCIJA

Još poznata i kao 'softargmax' funkcija ili više-klasna logistička regresija, softmax funkcija je funkcija koja prima vektor K realnih vrijednosti i vraća vektor K realnih vrijednosti čiji zbroj daje vrijednost 1. Ulazne vrijednosti mogu biti pozitivne, negativne, 0, veće od jedan, manje od jedan, ali će ih softmax funkcija uvijek transformirati u vrijednosti koje se nalaze između 0 i 1 tako da mogu biti protumačene kao vjerojatnosti. [25]

Matematička formula softmax funkcije slična je onoj sigmoidne funkcije i glasi:

Sličnost je u tome što softmax funkcija prima vektor, dok sigmoidna funkcija prima skalar. Dapače, sigmoidna funkcija je posebni slučaj softmax funkcije kada je ulazne vrijednosti potrebno klasificirati u dvije izlazne klase. [18]

Grafički prikaz sigmoidne prikazan je na donjoj slici:



Slika 8: grafički prikaz softmax funkcije

Softmax funkcija se najčešće koristi u izlaznom sloju neuronske mreže gdje je potrebno izlazne rezultate prikazati kao vjerojatnosti da ulazni primjer pripada određenom izlazu. [18]

## SLOJ SAŽIMANJA

Sloj sažimanja (eng. *pooling layer*) slijedi nakon jednog ili više konvolucijskih slojeva. Taj sloj, kao što mu i samo ime govori, reducira dimenzije izlaza iz konvolucijskog sloja. Drugim riječima, uzorkuje prostornu dimenzionalnost (visinu i širinu, ali ne i broj kanala) te tako smanjuje broj parametara i kontrolira prenaučenost mreže. Postoji nekoliko razloga zašto konvolucijska neuronska mreža sadrži takav skriveni sloj. [8][9][10][12]

Konvolucijski slojevi sustavno primjenjuju određeni filter (ili više njih) na ulazne kanale kako bi kreirali matrice značajki koje ukazuju na prisutnost odgovarajućih struktura na ulaznom kanalu. Ti slojevi su se pokazali jako efikasnima i slaganje konvolucijskih slojeva u seriju omogućuje da konvolucijska mreža detektira sofisticirane strukture kao što su konkretni oblici i specifični objekti. [1]

Međutim, matrice značajki pohranjuju precizne lokacije detektiranih struktura. To znači da bi mali pomak u lokaciji rezultirao potpuno različitom matricom značajki. Takav mali pomak u lokaciji se događa kod manjih modifikacija na, na primjer, ulaznim slikama. [3]

Taj problem se rješava uzorkovanjem, odnosno sažimanjem. Izlaz iz sloja sažimanja je matrica značajki manjih dimenzija koja i dalje sadrži važne informacije, ali ne sadrži informacije koje vrlo vjerojatno nisu potrebne. Takve matrice značajki su korisne jer će male promjene u lokaciji rezultirati jako sličnom, ako ne i istom, matricom značajki čime je postignuta generalizacija pretpostavljenih struktura. Ova sposobnost sažimanja se zove 'neosjetljivost mreže na lokalnu translaciju' (eng. *net's invariance to local translation*). [3][10][12][13]

Slojevi sažimanja također sadrže filtere, odnosno receptivno polje. Njihova dimenzija je skoro uvijek 2x2. Ti filteri su često puno manji od filtera konvolucijskog sloja i nikada ne sadrže vlastite brojčane vrijednosti. Čak bi ih se moglo nazivati okvirima. Također, pomak filtera u sloju sažimanja je postavljen na brojčanu vrijednost koja namjerno izbjegava preklapanje, a to je najčešće broj dva. [3][12]

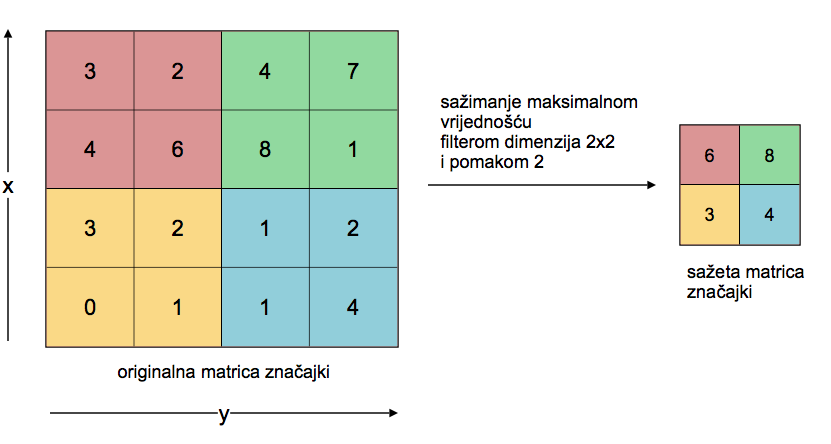
To znači da će sloj sažimanja skoro uvijek reducirati matricu značajki za faktor dva, odnosno svaka će se dimenzija matrice značajki konvolucijskog sloja prepoloviti (visina i širina). Broj elemenata ulazne matrice značajki će se smanjiti na četvrtinu. [3]

Postoje dvije moguće operacije u sloju sažimanja: sažimanje maksimalnom vrijednošću (eng. *max pooling*) i sažimanje prosječnom vrijednošću (eng. *average pooling*). [3][9]

Sažimanje maksimalnom vrijednošću uzima maksimalnu vrijednost iz okvira. To se čini tako što filter klizi po matrici značajki i, sa svakim pomakom, uzima najveću vrijednost iz okvira. [9]

Sažimanje prosječnom vrijednošću uzima prosječnu vrijednost okvira i pohranjuje ju u matricu značajki reduciranih dimenzija. [3]

Na dolje prikazanoj slici je opisano sažimanje maksimalnom vrijednošću:



Na slici je prikazana matrica značajki dimenzija 4x4. Filter sažimanja ima dimenzije 2x2 i pomak je postavljen na vrijednost dva.

Prva regija matrice značajki je obojena u crveno. Maksimalna vrijednost tog područja je 6 i ona se sprema u prvo polje sažete matrice značajki.

Filter se pomiče za dva mjesta u desno. Maksimalna vrijednost tog područja je 8 i ona se sprema u drugo polje.

Kako je filter došao do ruba matrice, vraća se na početak i spušta se za dva polja dolje. Cijeli proces se ponavlja i određene vrijednosti se pohranjuju na odgovarajuća mjesta u matrici.

Isti ovaj proces se odvija i kod sažimanja prosječnom vrijednošću. Jedina razlika je što se umjesto maksimalne vrijednosti uzima prosječna brojčana vrijednost područja koja se potom sprema na odgovarajuće mjesto u matrici.

Matematički se cijeli proces može opisati na sljedeći način:

Neka vrijedi sljedeća anotacija:

* ulaz u sloj: s dimenzijama gdje je slika koja ulazi u neuronsku mrežu.
* *padding*: (rijetko se koristi) i pomak:
* veličina filtera za sažimanje:
* funkcija sažimanja:
* izlaz iz sloja: s dimenzijama

Vrijedi:

S tim da:

Sloj za sažimanje ne posjeduje parametre koje konvolucijska neuronska mreža uči tijekom procesa učenja. [11]

## POTPUNO POVEZANI SLOJEVI I IZLAZNI SLOJ

Potpuno povezani slojevi u konvolucijskoj neuronskoj mreži su standardni skriveni slojevi tradicionalne neuronske mreže. Ovi slojevi također sadrže neke od ne-linearnih aktivacijskih funkcija. Ako je taj sloj izlazni sloj, često će kao aktivacijsku funkciju imati softmax ili sigmoidnu aktivacijsku funkciju. [12]

Potpuno povezani slojevi se koriste pri kraju konvolucijske neuronske mreže. Oni koriste rezultate konvolucijskog sloja ili sloja sažimanja kako bi prikazali odluke mreže. [12][13]

Postoji nekoliko vrsta potpuno povezanih slojeva u konvolucijskoj neuronskoj mreži. To su:

1. Sloj za izravnavanje (eng. *flatten layer*): uzima izlaz iz prethodnih slojeva (koji su u obliku matrice) te ih 'izravnava', odnosno transformira u vektor brojčanih vrijednosti.
2. Potpuno povezani sloj: na svoje ulaze prima vrijednosti prethodnog sloja za izravnavanje pomnožene s težinama veza i na njima primjenjuje aktivacijsku funkciju. Ovaj sloj se može smatrati standardnim skrivenim slojem u tradicionalnim umjetnim neuronskim mrežama.
3. Potpuno povezani izlazni sloj: na svojim izlazima daje konačne odluke neuronske mreže za određene ulazne primjere.[13]

Još jedan način na koji se može gledati potpuno povezane slojeve je taj da potpuno povezani slojevi čine tradicionalnu neuronsku mrežu koja je integrirana s konvolucijskom neuronskom mrežom.

# TRENIRANJE UMJETNIH NEURONSKIH MREŽA

U ovom poglavlju će se dati općeniti pregled procesa treniranja umjetnih neuronskih mreža. Nakon što je umjetna neuronska mreža strukturirana, potrebno ju je trenirati kako bi naučila preslikavati dani ulazni primjer na ispravan izlaz i u situacijama kada dobije podatke s kojima se prije nije susrela. Ovo poglavlje će opisati pojmove kao što su skupovi podataka za treniranje, provjeru i testiranje te će navesti razlog iza posjedovanja tri različita skupa podataka. Nakon toga će se opisati sam proces treniranja, koji se temelji na propagaciji prema natrag, dok će sama propagacija prema natrag biti opisana u zasebnom poglavlju. Poglavlje će objasniti kako dolazi do problema prenaučenosti i podnaučenosti i kako se ti problemi mogu reducirati ili čak zaobići. Na kraju će se navesti još nekoliko pojmova s kojima se često susreće tijekom treniranja neuronske mreže.

## FUNKCIJA GUBITKA

Prije nego što se opiše kako neuronska mreža uči, potrebno je objasniti što je to funkcija gubitka. Funkcija gubitka je funkcija koja govori koliko dobro određena mreža obavlja svoju funkciju preslikavanja ulaznih primjera u ispravne izlaze. Ona se intuitivno računa tako što se izlazna vrijednost neuronske mreže oduzme od željenog izlaza neuronske mreže za dani ulaz. Taj rezultat se kvadrira jer su negativni rezultati jednako loši kao i pozitivni rezultati. [27]

Najčešće korištena funkcija gubitka u neuronskim mrežama jest funkcija srednje kvadratne pogreške (eng. *mean squared error*), skraćeno MSE. [1]

Funkcija gubitka se matematički izražava na sljedeći način:

Ovo napisati u tablici.

Gdje *y* označava željenu izlaznu vrijednost, *y'* označava dobivenu izlaznu vrijednost, *i* označava redni broj ulaza, a *n* označava ukupan broj ulaza iz skupa za treniranje.

Ako je dobivena vrijednost gubitka velika, to znači da mreža nije naučila dobro predviđati (ili preslikavati).

Gubitak je razlika između željenog izlaza za neuronsku mrežu i stvarnog izlaza neuronske mreže. Kao što se može vidjeti iz dane formule, ukupan gubitak mreže čine akumulirani gubici za svaki pojedini izlaz. Taj akumulirani rezultat se podijeli s ukupnim brojem ulaznih primjera. [1]

## KAKO NEURONSKA MREŽA UČI

Tijekom strukturiranja neuronske mreže, težine veza se najčešće inicijaliziraju nasumičnim brojčanim vrijednostima. Očito je da rezultati predviđanja takve mreže neće biti zadovoljavajući. Cilj treniranja neuronske mreže jest dobiti mrežu koja ima visoku točnost predviđanja. U terminima funkcije gubitka, cilj procesa učenja je postići što manju vrijednost funkcije gubitka. [27]

Neuronska mreža tijekom procesa učenja uči preslikavati ulazne primjere u ispravne izlaze na temelju danog skupa primjera za treniranje. Znanje o načinu obrade ulaznih podataka implicitno je pohranjeno u težinama veza između neurona u mreži. Treniranje uključuje proces iterativnog pronalaženja skupa težina u neuronskoj mreži sve do trenutka kada je izlazna vrijednost mreže, provjerena na skupu podataka za provjeru, zadovoljavajuća. To podrazumijeva da se kroz skup svih mogućih rješenja napreduje korak po korak, donošenjem malih izmjena u težinama modela tijekom svake iteracije. Kako se težine mijenjaju, tako model postaje sve 'pametniji' u smislu da sve ispravnije preslikava ulazne primjere u odgovarajuće izlaze. [1][28][29]

Algoritam, koji se koristi za ispravno provođenje procesa učenja, naziva se optimizacijski algoritam (eng. *optimization algorithm, optimizer*). Najpopularniji optimizacijski algoritam naziva se 'stohastički gradijentni spust' (eng. *stochastic gradient descent*). Popularan je zbog toga što se jako jednostavno izvodi, ali s druge strane, zahtijeva mnogo iteracija. [1][26]

Proces treniranja rješava optimizacijski problem pronalaženja težina koje će rezultirati najmanjom vrijednošću funkcije gubitka. Optimizacijski proces se konceptualno može zamisliti kao traženje najboljeg rješenja u 'krajoliku' svih mogućih rješenja koji se sastoji od brežuljaka i dolina. [29]

Kod jednostavnog optimizacijskog problema, taj krajolik može nalikovati velikoj posudi kojoj je relativno jednostavno naći minimum. Takvi optimizacijski procesi se matematički opisuju kao konveksni. S druge strane, optimizacijski algoritam neuronskih mreža sadrži mnogo brežuljaka i dolina i matematički se opisuje kao ne-konveksni optimizacijski problem. [29]

## GRADIJENT

Kao što je spomenuto u prethodnom poglavlju, proces treniranja neuronske mreže rješava optimizacijski problem pronalaženja takvih težina koje će rezultirati minimalnim gubitkom. Isto tako je spomenuto kako se najpoznatiji optimizacijski algoritam naziva stohastički gradijentni spust. To je algoritam koji se, kao što mu samo ime kaže, temelji na gradijentu.

Prije nego se na matematički način počne objašnjavati uloga gradijenta tijekom procesa optimizacije, dobro je prisjetiti se što čini derivaciju funkcije u odnosu na neku varijablu.

Neka se, kao primjer, uzme funkcija .

Derivacija te funkcije je: .

Derivacija govori koliko se brzo funkcija mijenja kada se parametar x pomakne beskonačno malim korakom u pozitivnom smjeru. Matematički se to može izraziti kao:

Opis gornjeg izraza je sljedeći: koliko se aproksimativno vrijednost određene funkcije (izraz lijevo) promijeni, mjera te promjene će biti jednaka derivaciji te funkcije u odnosu na parametar *x* pomnoženoj s mjerom za koju se promijenila vrijednost parametra *x*.

Kod primjera , gdje je derivacija jednaka 1, kada bi funkcija uzela neki *epsilon* pomak u pozitivnom smjeru, rezultat funkcije će se promijeniti za 1 pomnožen tom stopom *epsilon*. Rezultat je u ovom slučaju sam *epsilon*. [27]

Gradijent je vektor parcijalnih derivacija. Njegovi elementi su derivacije određene funkcije u odnosu na varijable o kojima ta funkcija ovisi. Kada bi se kao primjer uzela već prije definirana funkcija , njen vektor bi sadržavao samo jedan element jer ova funkcija ovisi o samo jednom parametru. Kod složenijih funkcija (koje ovise o više parametara), elementi gradijenta će biti derivacije u odnosu na određene varijable. [27]

## STOHASTIČKI GRADIJENTNI SPUST

Kako bi se objasnio stohastički gradijentni spust, koristit će se sljedeća funkcija sa svojom prvom derivacijom:

Funkcija:

Prva funkcija derivacije:

Minimum ove funkcije se nalazi u točki .

Međutim, to računalo samo po sebi ne zna već ono samo mora izračunati minimum.

Neka računalo uzme neku nasumičnu vrijednost, na primjer . Vrijednost derivacije funkcije u točki je 2. To znači da, kada funkcija učini beskonačno mali korak u pozitivnom smjeru, njena vrijednost će se promijeniti proporcionalno faktoru dva, odnosno njena vrijednost će se povećati. S obzirom da je cilj stohastičkog gradijentnog spusta minimizirati funkciju gubitka, on će trebati učiti korak u suprotnom smjeru kako bi se vrijednost funkcije smanjila.

S druge strane, ako računalo postavi da je , vrijednost derivacije funkcije u točki -1 će biti -2. To znači da, ako funkcija učini beskonačno mali korak u pozitivnom smjeru njena će se vrijednost promijeniti za -2. Drugim riječima, vrijednost funkcije će se smanjiti što i je cilj stohastičkog gradijentnog spusta.

Ukratko, kada je i vrijednost derivacije veća od 0, stohastički gradijentni spust treba učiniti korak u negativnom smjeru. Kada je i vrijednost derivacije manja od 0, stohastički gradijentni spust treba učiniti korak u pozitivnom smjeru.

Međutim, kada je , računalo ne zna koliki korak mora napraviti u negativnom smjeru jer derivacija samo garantira da će se, ako se učini beskonačno mali korak u negativnom smjeru, vrijednost funkcije smanjiti. Korak, koji je potrebno učiniti u negativnom smjeru, naziva se 'stopa učenja' (eng. *learning rate*) i jedan je od hiper-parametara neuronske mreže. Što su to hiper-parametri, bit će objašnjeno u odvojenom poglavlju. [27]

Kao što je već spomenuto, gradijent je vektor čiji su elementi parcijalne derivacije dane funkcije u odnosu na neke određene varijable. Taj vektor pokazuje prema nekom smjeru u prostoru. Pokazuje u smjeru najstrmijeg rasta funkcije. S obzirom da je cilj minimizirati funkciju, potrebno je učiniti korak u smjeru suprotnom od onog prema kojem pokazuje gradijent. [27]

U neuronskoj mreži, vrijednosti *x* su ulazni primjeri koji se prosljeđuju neuronskoj mreži. Vrijednosti funkcije, odnosno *y* su izlazne vrijednosti neuronske mreže. Drugim riječima *y* čine predviđanja neuronske mreže za dane ulazne primjere *x*. Kao što je već spomenuto, proces treniranja rješava optimizacijski problem pronalaženja težina koje će rezultirati najmanjom vrijednošću funkcije gubitka. To znači da se funkcija gubitka derivira u odnosu na težine veza *w* u neuronskoj mreži jer su vrijednosti težine veza ono što treba mijenjati kako bi se poboljšalo predviđanje mreže.

Sada, neka je izračunat gradijent funkcije gubitka u odnosu na neku određenu vezu u mreži. Nadalje, neka se dobivena vrijednost gradijenta pomnoži sa stopom učenja. Dobivena vrijednost se oduzima od 'stare' težine veze. Rezultat jest nova težina te određene veze u odnosu na koju se računao gradijent. Matematički se to izražava na sljedeći način:

Objašnjenje formule je sljedeće:

|  |  |
| --- | --- |
| **izraz** | **definicija** |
|  | težina veze *j* |
| *L* | funkcija gubitka |
| *lr* | stopa učenja |

Ovaj proces se iterativno ponavlja sve dok se ne postigne vrijednost težine za koju će funkcija gubitka konvergirati u svoj minimum. Cijeli ovaj algoritam se naziva gradijentni spust.

Kao što je već spomenuto, *lr* označava stopu učenja. Ne postoji pravilo koliko se velika stopa učenja treba primijeniti tijekom svake iteracije. Međutim, ako se primijeni prevelika stopa učenja, postoji mogućnost da će dobiveni rezultat za određenu težinu veze 'preskočiti' minimum funkcije, odnosno postoji opasnost od dobivanja pretjeranog rezultata (eng. *overshooting*). Tada se kaže da je algoritam divergirao. S druge strane, ako se uzme premala stopa učenja, previše će vremena biti potrebno kako bi se postigla konvergencija u nekom lokalnom minimumu. [1][27]

Stopa učenja je broj čija se vrijednost najčešće nalazi između 0.01 i 0.0001. Vrijednost koja se dodjeljuje stopi učenja zahtijeva neko testiranje za svaku umjetnu neuronsku mrežu posebno kako bi se odredilo s kojom će stopom učenja ta mreža najefikasnije učiti. [1]

Postoji još nekoliko optimizacijskih algoritama koji se koriste u neuronskim mrežama:

1. Newton-ova metoda
2. Konjugirani gradijent
3. Polu-Newton-ova metoda
4. Levenberg-Marquardt-ov algoritam [26]

## SKUPOVI ZA TRENIRANJE, PROVJERU I TESTIRANJE

Za potrebe treniranja i testiranja odabrane neuronske mreže, skup podataka, koji se prosljeđuje toj neuronskoj mreži, dijeli se na tri dijela. Dobiveni skupovi podataka su sljedeći:

1. skup podataka za treniranje (eng. *training set*)
2. skup podataka za provjeru (eng. *validation set*)
3. skup podataka za testiranje (eng. *testing set*).

Primjeri iz skupa podataka za treniranje, odnosno učenje, služe za učenje u užem smislu (podešavanje težina veza i pragova). Tijekom svake epohe, model će uvijek iznova učiti značajke podataka nad istim skupom podataka za treniranje. Cilj je na taj način naučiti model da točno predvidi značajke podataka koje do tada nije primijetio. Mreža će svoja predviđanja temeljiti na onome što je naučila pomoću skupa podataka za treniranje. [1][28]

Pomoću primjera iz skupa podataka za provjeru vrši se, tijekom samog procesa učenja, provjera rada mreže s trenutnim težinama kako bi se postupak učenja zaustavio u trenutku degradacije performansi mreže. Taj skup je odvojen od skupa podataka za treniranje, ali se također koristi tijekom procesa učenja. Nadalje, rezultati preslikavanja skupa podataka za provjeru neće se koristiti kako bi se promijenile težine veza u mreži. Skup podataka za provjeru služi samo za to da provjeri performanse mreže tijekom procesa učenja. Drugim riječima, težine veza se neće mijenjati na temelju informacija o gubicima iz skupa podataka za provjeru. [1][28]

Umjetnu neuronsku mrežu moguće je prenaučiti (eng. *overfitting*) – nakon određenog broja iteracija mreža gubi sposobnost generalizacije. To znači da mreža jako dobro preslikava primjere iz skupa podataka za učenje, dok ostale primjere (iz skupa podataka za provjeru i testiranje kao i primjere s kojima se do tada nije susrela) jako loše preslikava. Ako se tijekom procesa učenja mreža istovremeno provjerava skupom za provjeru i, ako su rezultati skupa za provjeru jednako dobri, velika je vjerojatnost da neće doći do prenaučenosti. Malo kasnije će se opisati kako se može smanjiti prenaučenost. [1][28]

S druge strane, postoji i pojava podnaučenosti (eng. *underfitting*). Tada mreža nije sposobna ispravno preslikavati podatke na kojima je trenirana, a pogotovo nije sposobna ispravno preslikavati podatke iz skupova za provjeru i testiranje.

Konačno, skup podataka za testiranje se koristi na mreži koja je završila s procesom učenja. Taj skup je odvojen od skupova podataka za treniranje i provjeru.

Ovdje je potrebno spomenuti da se skupovi podataka za treniranje i provjeru sastoje od označenih podataka. Ti podaci moraju biti označeni kako bi se izračunao gubitak i ispravnost preslikavanja svake epohe. Nadalje, neuronske mreže pripadaju nadziranom učenju jer se nadzire i usmjerava proces učenja neuronskih mreža. Kod nadziranog učenja svaki podatak, koji se prosljeđuje mreži tijekom treniranja, čini par koji se sastoji od ulaznog objekta, odnosno primjera, i njemu odgovarajuće oznake (eng. *label*), odnosno željene izlazne vrijednosti. [1]

Nastavno na ono što je opisano u poglavlju o funkciji gubitka, mreža će preslikati dani ulazni primjer u određeni izlaz. Greška, odnosno gubitak, se računa tako da se izračuna razlika između vrijednosti oznake za dani primjer i onoga što je mreža dala na svom izlazu.

Također je važno napomenuti da se oznake kodiraju u nešto numeričko (0, 1, 2…). Isto tako, izlaz iz mreže nikada neće biti neka riječ ili nešto ne-numeričko.

Nakon što je mreža naučena i provjerena, proslijedit će joj se skup podataka za testiranje, koji nije označen, i za koji će mreža trebati znati predvidjeti odgovarajući izlaz. Glavna razlika između skupa podataka za testiranje i druga dva skupa je ta što skup za testiranje ne smije imati oznake pridružene podacima. Skupom za treniranje se provjerava generalizira li mreža dovoljno ispravno prije nego što ju se stavi u produkciju. [1]

Dolje prikazana tablica ukratko opisuje sve što je navedeno o skupovima podataka.

|  |  |  |
| --- | --- | --- |
| **skup podataka** | **mijenja li težine veza** | **opis** |
| skup podataka za učenje | da | Koristi se za treniranje modela. Cilj treniranja je da model dobro klasificira podatke, ali u isto vrijeme dobro generalizira. |
| skup podataka za provjeru | ne | Koristi se u vrijeme treniranja kako bi se provjerilo koliko dobro model generalizira. |
| skup podataka za treniranje | ne | Koristi se kako bi se konačno provjerila sposobnost mreže da dobro generalizira. |

## REDUCIRANJE PRENAUČENOSTI I PODNAUČENOSTI

Kao što je već opisano, prenaučenost se javlja kada mreža jako dobro preslikava primjere iz skupa podataka za treniranje, ali jako loše preslikava podatke koji se ne nalaze u tom skupu. Drugim riječima, mreža ne posjeduje sposobnost generalizacije.

Najjednostavniji način, kojim se može smanjiti prenaučenost, jest, ako je moguće, ubaciti još primjera u skup za treniranje. Što više primjera mreža uči, ona može bolje generalizirati.

Još jedan način pomoću kojeg se može smanjiti prenaučenost jest na neki način 'povećati' skup podataka. Pod tim se misli na to da se stvore dodatni, modificirani podaci koji su nastali razumnom modifikacijom primjera iz skupa za treniranje. Na primjer, kod slika, koje se koriste kod konvolucijskih neuronskih mreža, podaci se mogu modificirati na sljedeće načine:

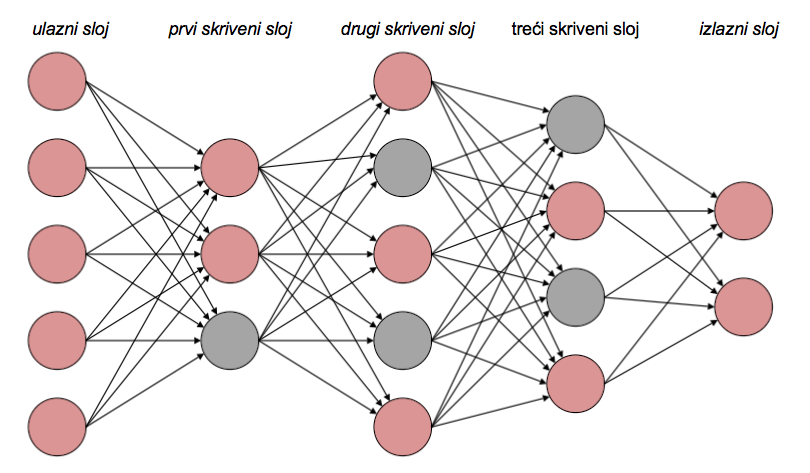
1. obrezivanjem
2. rotacijom
3. okretanjem
4. zumiranjem
5. primjenom nekog filtera (na primjer, zamutiti sliku ili ju učiniti crno-bijelom).[1]

Glavna ideja iza povećanja skupa podataka za treniranje jest da se u skup za treniranje ubacuju podaci koji su slični već postojećim podacima, ali su također razumno modificirani do te mjere da nisu potpuno isti već postojećim podacima.

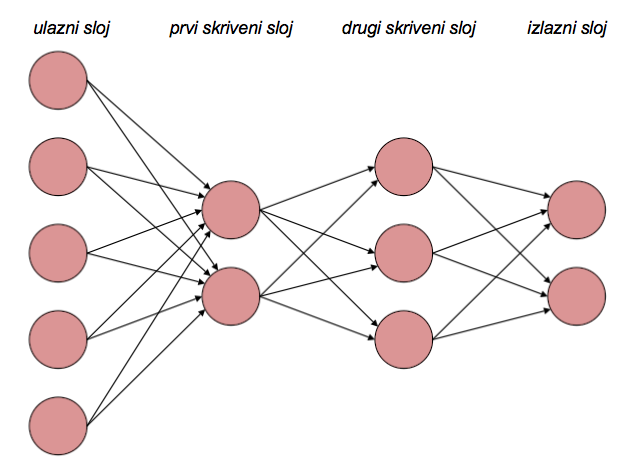
Nadalje, još jedan način pomoću kojeg se može smanjiti prenaučenost jest smanjiti složenost neuronske mreže. To se radi na način da se ukloni jedan ili više skrivenih slojeva ili se reducira broj neurona u određenom sloju ili slojevima. [1]

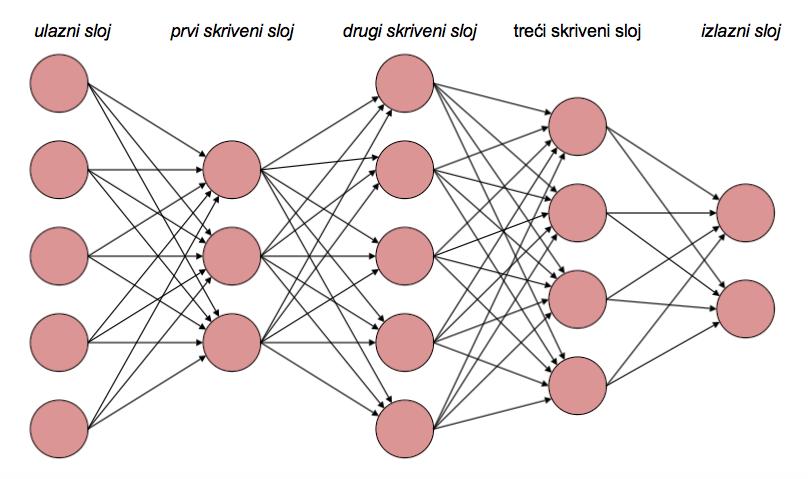
Konačno, još jedna od popularnih metoda za reduciranje prenaučenosti jest korištenje *dropout* tehnike. Ideja iza te tehnike jest da se nasumično ignoriraju grupe neurona u određenom sloju. Ta radnja će spriječiti ignorirane neurone da sudjeluju u obradi podataka. [1]



S druge strane, kaže se da je mreža podnaučena ako nije sposobna dobro preslikavati podatke iz skupa za treniranje, a kamoli podatke koji ne pripadaju tom skupu.

Prva tehnika, kojom se može reducirati podnaučenost mreže, jest povećati složenost mreže. To je tehnika suprotna onoj kojom se reducira prenaučenost. Ako su podaci iz skupa za treniranje složeni, a mreža relativno jednostavna, ona najvjerojatnije neće biti dovoljno sofisticirana kako bi mogla ispravno klasificirati ulazne primjere. [1]





Složenost modela se može povećati na sljedeće načine:

1. povećanjem broja slojeva u modelu
2. povećanjem broja neurona u svakom sloju
3. promjenom razmještaja i vrsti slojeva u mreži.

Nadalje, još jedan od načina reduciranja podnaučenosti jest dodavanje oznaka ulaznim primjerima iz skupa za treniranje. Dodatne oznake mogu pomoći mreži ispravnije klasificirati ulazne primjere.

Na primjer, neka mreža pokušava predvidjeti cijene dionice na temelju njezine cijene tijekom zatvaranja burze u posljednja tri dana. Ulazni primjeri bi se sastojali od sljedećih značajki:

1. cijena dionice na kraju prvog dana
2. cijena dionice na kraju drugog dana
3. cijena dionice na kraju trećeg dana.

Kada bi se dodale dodatne oznake ovim primjerima, kao na primjer cijene dionice tijekom otvaranja burze, možda bi to pomoglo mreži da ispravnije predviđa izlazne vrijednosti. [1]

Konačno, jedna od tehnika reduciranja podnaučenosti jest i reduciranje *dropout* tehnike. Tijekom korištenja *dropout* tehnike može se odrediti koliki će se postotak neurona ignorirati tijekom obrade podataka. Na primjer, ako se ignorira 50% neurona i rezultat toga je da je mreža podnaučena, potrebno je smanjiti postotak ignoriranih neurona. Neuroni koji su ignorirani, ignorirani su samo tijekom procesa učenja, ali ne i tijekom provjere neuronske mreže. Tako, ako mreža bolje preslikava podatke iz skupa za provjeru, dobro je smanjiti postotak ignoriranih neurona. [1]

## PRIMJER, GRUPA, EPOHA

Primjerom (eng. *sample*) u neuronskoj mreži se smatra jedan ulazni podatak iz određenog skupa podataka za koji mreža treba predvidjeti izlaznu vrijednost.

Grupu (eng. *batch*) čini više primjera iz skupa podataka za treniranje koji će se kao jedna cjelina u jednom trenutku proslijediti neuronskoj mreži.

Epohu (eng. *epoch*) čini jedan prolazak svih podataka iz skupa za treniranje. Bitno je napomenuti da grupa i epoha nisu istoznačnice.

Neka skup podataka za treniranje čini 10,000 primjera. Kada svih 10,000 primjera po jedan put prođe kroz neuronsku mrežu, kaže se da je završena jedna epoha. Tijekom procesa učenja, prolazi se kroz više epoha.

Nadalje, ako se mreži, tijekom procesa učenja, u jednom trenutku i kao jedna cjelina proslijedi 100 primjera iz skupa za učenje, kaže se da je da je veličina grupe jednaka 100.

Kako epohu čini jedan prolazak cijelog skupa za treniranje, bit će potrebno 100 grupa veličine 100 primjera kako bi se završila jedna epoha.

Općenito vrijedi da, što je veća grupa, neuronska mreža će brže završiti jednu epohu. Međutim, potrebno je uzeti u obzir računalnu snagu koju neuronska mreža ima na raspolaganju. Čak i ako računalo može obraditi velike grupe ulaznih primjera u jednom trenutku, performanse mreže se mogu degradirati. Ako je grupa, koja se šalje mreži u jednom trenutku, dovoljno velika, mreža možda neće moći dobro generalizirati. [1]

Veličina grupe je još jedan od hiper-parametara koji treba podesiti i testirati za svaku neuronsku mrežu i računalo koje podržava tu neuronsku mrežu.

## PRAG

Kad se govori o pragu u neuronskoj mreži, misli se na prag svakog neurona u mreži pojedinačno. Svaki neuron ima svoj prag i vrijednost praga je jedan od parametara, uz težine veza, koje mreža uči tijekom procesa učenja. Kao što stohastički gradijentni spust uči i mijenja težine veza, isto tako uči i mijenja vrijednosti pragova. Vrijednost praga odlučuje hoće li se izlazna vrijednost danog neurona propagirati dalje kroz mrežu. Drugim riječima, prag odlučuje hoće li se ili neće, i koliko, aktivirati neuron.

Kao što je objašnjeno u poglavlju o dubokom učenju i neuronskim mrežama, svaku neuron na svoj ulaz prima ponderiranu sumu ulaza iz prethodnog sloja. Ta ponderirana suma se potom prosljeđuje aktivacijskoj funkciji određenog neurona. Vrijednost praga se zbraja s ponderiranom sumom i ta nova vrijednost se onda prosljeđuje aktivacijskoj funkciji neurona. [1]

Neka neuronska mreža ima 3 ulazna neurona. Prvi neuron neka ima vrijednost 1, drugi 2 i treći 3. Ovaj primjer će se fokusirati samo na jedan neuron u prvom skrivenom sloju.

Nadalje, neka neuron u prvom skrivenom sloju koristi ReLU aktivacijsku funkciju. Težine veza su inicijalizirane nasumičnim vrijednostima. Također, neka neuron na početku nema nikakvu vrijednost praga, odnosno vrijednost mu je 0.

Ponderirana suma, koju prima neuron u skrivenom sloju, izgleda ovako:

Ova vrijednost se prosljeđuje ReLU aktivacijskoj funkciji. Kao što je već objašnjeno, vrijednost ReLU funkcije će biti 0 za sve negativne vrijednosti, a za pozitivne vrijednosti će imati istu vrijednost kao i ulaz.

U ovom primjeru, izlazna vrijednost iz ReLU će biti 0.

S obzirom da je rezultat ReLU 0, neuron se neće aktivirati.

Neka se sada prag neurona pomakne na -0.5. vrijednost praga će biti suprotna od -0.5, bit će 0.5.

Ponderiranoj sumi se dodaje vrijednost praga i nova vrijednost iznosi:

Kada se ova nova vrijednost proslijedi ReLU funkciji, rezultat će iznositi 0.22.

Neuron se sada smatra aktiviranim te se mreža smatra fleksibilnijom što se tiče prilagođavanja podacima. Posjeduje veći opseg u smislu koje će vrijednosti aktivirati određeni neuron. Ovaj isti proces se može koristiti i kako bi se suzio opseg izlaznih vrijednosti koje će aktivirati neuron.

# POVRATNA PROPAGACIJA

Ovo poglavlje je posvećeno povratnoj propagaciji (eng. *backpropagation*) i njenoj ulozi tijekom procesa treniranja neuronske mreže.

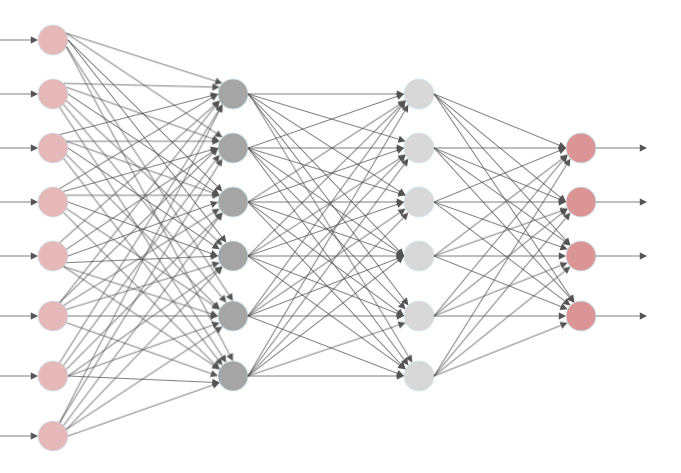
Prvo će se ukratko ponoviti neke bitne točke u vezi stohastičkog gradijentnog spusta koje su se spominjale u jednom od prethodnih poglavlja. Nakon toga će se pokazati gdje točno pripada povratna propagacija te će se veliki dio poglavlja baviti intuicijom iza toga što povratna propagacija zapravo radi.

## PREGLED STOHASTIČKOG GRADIJENTNOG SPUSTA (SGD)

Kao što je već spomenuto, tijekom procesa treniranja, stohastički gradijentni spust radi na tome da minimizira funkciju gubitka na način da mijenja težine veza nakon svake epohe ili grupe. Također je opisano da se to radi na način da se prvo izračuna gradijent, ili derivacija, funkcije gubitka u odnosu na težine u modelu. Međutim, ništa detaljnije nije bilo opisano.

## PROPAGACIJA PREMA NAPRIJED

Neka je dana proizvoljna neuronska mreža koja sadrži dva skrivena sloja. Zbog jednostavnosti, primjer će se temeljiti na jednom ulaznom primjeru koji se prosljeđuje mreži.



Slika 1: neuralna mreža s dva skrivena sloja

Kao podsjetnik, tijekom procesa učenja, svaki put, kada se ulazni primjer proslijedi mreži, taj primjer se prosljeđuje kroz mrežu prema naprijed sve dok ne stigne do njenog izlaznog sloja. Isto tako, svaki čvor u modelu prima ulaz iz prethodnog sloja i ulaz u dani neuron je ponderirana suma koju čini težina svake veze pomnožena izlaznom vrijednošću svakog pojedinog čvora u prethodnom sloju. Ta ponderirana suma se, kao ulazna vrijednost, prosljeđuje aktivacijskoj funkciji. Vrijednost, koju vraća aktivacijska funkcija, je izlazna vrijednost određenog čvora koja se zatim prosljeđuje kao dio ulazne vrijednosti u čvor koji se nalazi u sljedećem sloju. Ovaj se proces ponavlja za svaki sloj u mreži dok se ne stigne do izlaznog sloja i zato se on naziva 'propagacija prema naprijed'.[1]

Kada se dosegne izlazni sloj, dobiveni rezultat predstavlja izlaznu vrijednost, odnosno predviđanje mreže, za dani ulazni primjer.

## RAČUNANJE GUBITKA

Kada su dobiveni izlazni rezultati, računa se gubitak mreže. Način, na koji se računa gubitak, ovisi o odabranoj funkciji gubitka koja će se koristiti. Neka, radi jednostavnosti, funkcija gubitka samo računa koliko je mreža udaljena od ispravne klasifikacije za dani ulazni primjer. O gubitku se može misliti i kao o razlici između onoga što je neuronska mreža predvidjela i onoga što je je trebala predvidjeti.[1]

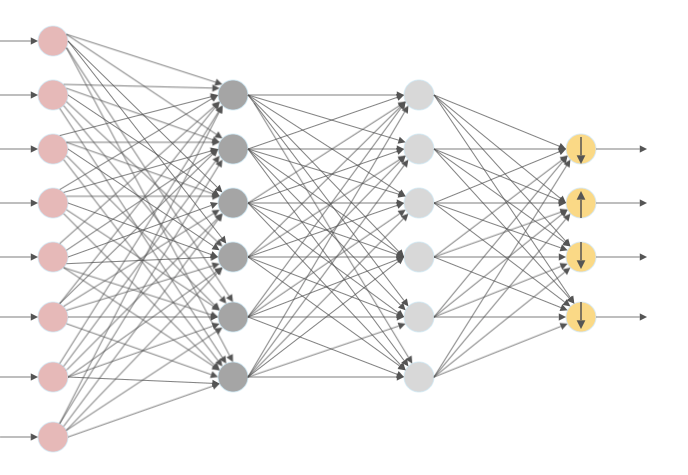
Kao što je spomenuto, cilj gradijentnog spusta jest minimizirati funkciju gubitka. To se radi uzimanjem derivacije, odnosno gradijenta, funkcije gubitka u odnosu na težine u modelu.

Povratna propagacija jest alat koji koristi stohastički gradijentni spust kako bi izračunao gradijent funkcije gubitka.

## INTUICIJA POVRATNE PROPAGACIJE

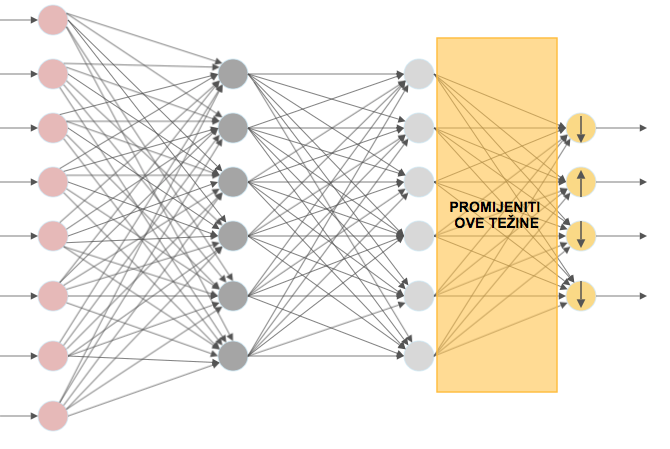
Kako bi ažurirao težine veza, gradijentni spust prvo provjerava rezultate aktivacijskih funkcija u izlaznom sloju.

Neka izlazni čvor, kojem je pridružena strelica koja pokazuje prema gore, predstavlja ispravan izlaz za dani ulazni primjer. U tom slučaju, gradijentni spust 'razumije' da se vrijednost tog čvora mora povećati, dok se vrijednosti ostalih čvorova moraju smanjiti. Tako se smanjuje gubitak za dani ulazni primjer.[1]

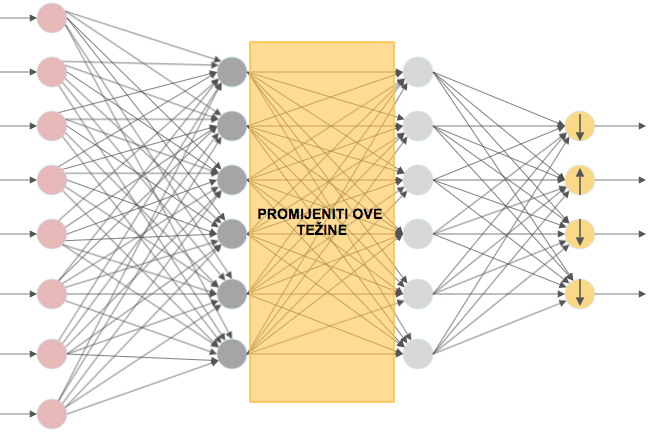


Slika 2: cilj gradijentnog spusta je povećati vrijednost ispravnog izlaznog čvora

Kada bi se htjele promijeniti vrijednosti izlaznih čvorova, jedan od načina kako bi se to moglo napraviti jest da se promijene težine veza koje spajaju izlazni sloj s njemu prethodnim slojem. Drugi način, na koji bi se to moglo napraviti, jest da se promijeni izlazna vrijednost aktivacijskih funkcija koje se nalaze u sloju koji prethodi izlaznom sloju.[1]



Nije moguće direktno promijeniti izlaz iz aktivacijskih funkcija jer on ovisi o težinama koje prethode predzadnjem sloju u neuronskoj mreži. Međutim, moguće je indirektno utjecati na promjenu izlaza tih aktivacijskih funkcija tako da se cijeli izračun gradijenta prebaci za još jedan sloj unatrag (sada se govori o težinama koje spajaju predzadnji sloj s njemu prethodnim slojem) i tako da se promijene težine veza na isti način koji je bio opisan za izlazni sloj.



Slika 3: kako bi se promijenio izlaz iz aktivacijske funkcije, potrebno je promijeniti težine tog sloja

Cijeli se proces pomicanja unatrag, kako bi se promijenile težine i na taj način utjecalo na izlaz aktivacijskih funkcija, ponavlja sve dok se ne dosegne ulazni sloj modela. Vrijednosti ulaznih čvorova se ne mogu promijeniti jer one sadrže stvarne vrijednosti ulaznih primjera koje se prosljeđuju neuronskoj mreži.[1]

Ukratko, gradijentni spust se pomiče natrag kroz mrežu, mijenja težine veza s desna na lijevo kako bi pomakao vrijednosti izlaznih čvorova u onom smjeru koji smanjuje gubitak. To znači da, za jedan ulazni primjer, stohastički gradijentni spust nastoji povećati vrijednost ispravnog čvora i smanjiti vrijednosti neispravnih čvorova na onaj način koji će najefikasnije smanjiti cjelokupan gubitak mreže.

Cijeli se ovaj proces, koji je pokazan na primjeru jednog ulaznog primjera, odvija i u slučaju više ulaznih primjera koji se u seriji prosljeđuju neuronskoj mreži. Razlika je u tome što će konačne promjene težina biti jednake prosječnoj izračunatoj promjeni težina za svaki pojedini ulazni primjer.

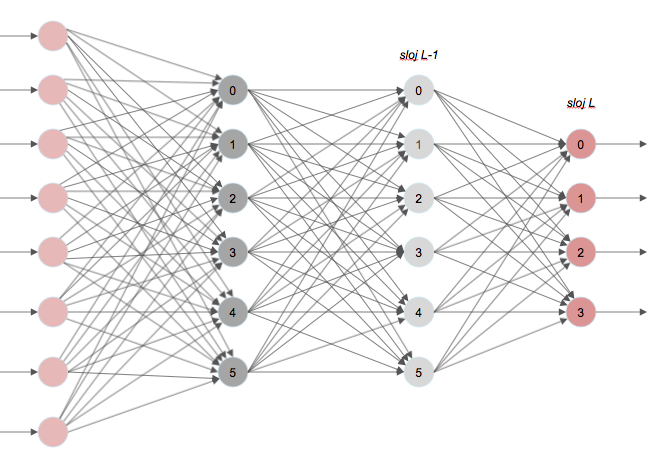
## MATEMATIČKA NOTACIJA POVRATNE PROPAGACIJE

Sljedeća tablica definira matematičke anotacije koje će se koristiti kako bi se proces povratne propagacije opisao na matematički način.

|  |  |
| --- | --- |
| **simbol** | **definicija** |
| L | broj slojeva u mreži |
| l | indeks (redni broj) sloja |
| j | indeks čvora u sloju l (počinje od nule) |
| k | indeks čvora u sloju l-1 (počinje od nule) |
|  | željeni izlaz čvora j u izlaznom čvoru L kod jednog uzorka za treniranje |
|  | funkcija gubitka kod jednog uzorka za treniranje |
|  | vektor težina koje povezuju čvorove iz l-1 a čvorom j u sloju l |
|  | težina koja povezuje čvor k u sloju l-1 s čvorom j u sloju l |
|  | ulazna vrijednost u čvor j koji se nalazi u sloju l |
|  | aktivacijska funkcija koja se koristi u sloju l |
|  | izlaz iz aktivacijske funkcije čvora j koji se nalazi u sloju l |

Kao što se može primijetiti, sve definicije iz prethodne tablice ovise o sljedećim indeksima:

|  |  |
| --- | --- |
| **simbol** | **definicija** |
| l | indeks sloja |
| j | indeks čvora u sloju l |
| k | indeks čvora u sloju l-1 |



## MATEMATIČKA PROMATRANJA ZA POVRATNU PROPAGACIJU

U ovom poglavlju će se prikazati kako se funkcija gubitka može matematički opisati, a nakon toga će se prikazati kako se matematički mogu prikazati ulazi i izlazi za bilo koji dani čvor. Na kraju će se definirati metoda deriviranja funkcije gubitka korištenjem povratne propagacije.

### FUNKCIJA GUBITKA

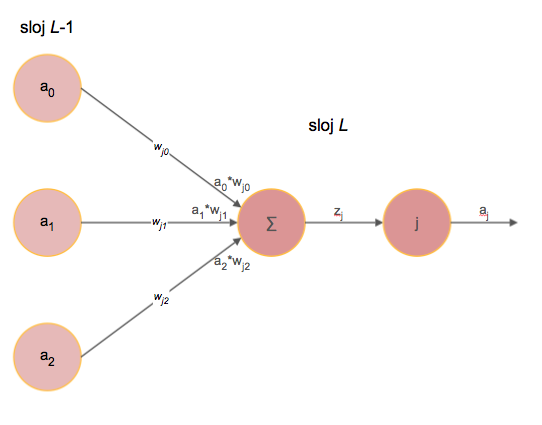
Funkcija

je kvadrat razlika izlazne vrijednosti aktivacijske funkcije i željenog izlaza za čvor *j* u izlaznom sloju *L*. Ovo se može protumačiti i kao gubitak čvora *j* u sloju *L*. Kako bi se izračunao ukupni gubitak, potrebno je zbrojiti rezultat kvadrata razlike za svaki čvor *j* u izlaznom sloju *L*.

Matematički se to opisuje na sljedeći način:

### ULAZNA VRIJEDNOST

Ulazna vrijednost čvora *j*, koji se nalazi u sloju *l*, jest ponderirana suma izlaza aktivacijskih funkcija iz prethodnog sloja *l-1*.

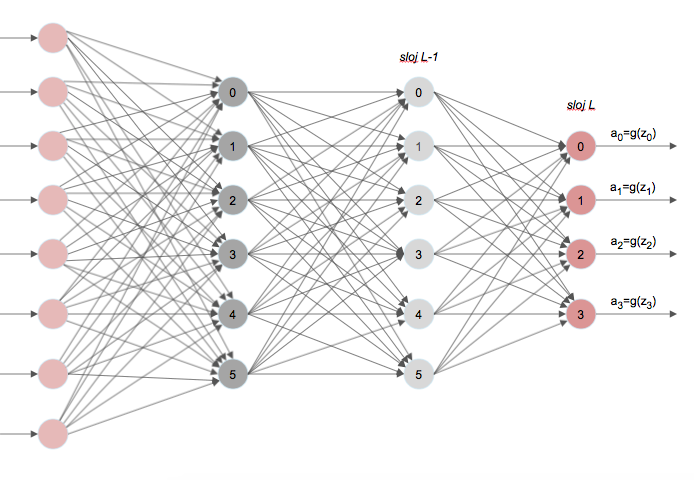


Individualna ulazna vrijednost izgleda ovako:

Tako, za dani čvor *j* u sloju *l*, ulaz izgleda ovako:

### IZLAZ IZ AKTIVACIJSKE FUNKCIJE

Izlaz iz aktivacijske funkcije za dani čvor *j* u sloju *l* je rezultat ulazne vrijednosti koja se prosljeđuje aktivacijskoj funkciji po vlastitom izboru i koja se označava kao: .



Matematički izraz za izlaz iz aktivacijske funkcije za dani čvor *j* u sloju *l* je:

### FUNKCIJA GUBITKA KAO KOMPOZICIJA FUNKCIJA

Kao što je već napisano, definicija funkcije gubitka izgleda ovako:

Gubitak jednog čvora *j* u izlaznom sloju *L* se matematički izražava na sljedeći način:

Kako je funkcija funkcija za jedan izlazni čvor *j* u sloju *L*, funkcija se može opisati kao funkcija od :

.

Promatranjem definicije za , može se primijetiti da ­­­­­ ovisi i o . Kako je konstanta, ­­­­­ se promatra samo kao funkcija od , dok se smatra parametrom koji pomaže definirati ovu funkciju.

Aktivacijski izlaz čvora *j* u izlaznom sloju *L* je funkcija izlaza za čvor *j*. Na temelju jednog od prethodnih promatranja, to se sada može izraziti kao:

.

Ulazna vrijednost čvora *j* je funkcija svih težina koje su vezane za čvor *j*. Tako se može izraziti kao funkcija od :

.

Tako da vrijedi sljedeći izraz:

.

Ovaj izraz je pokazao da je kompozicija funkcija.

Nadalje, vrijedi:

To znači da je i ukupan gubitak mreže za jedan ulaz kompozicija funkcija. Ovo je korisno znati kako bi se razumjela derivacija . Kako bi se derivirala kompozicija funkcija, koristi se takozvano 'lančano pravilo'.

## RAČUNANJE GRADIJENTA

Kako bi stohastički gradijentni spust promijenio težine u mreži, prvo treba izračunati gradijent gubitka u odnosu na te težine.

Težina koja spaja čvor *2* u sloju *L-1* sa čvorom *1* u čvoru *L* se označava na sljedeći način:

.

Derivacija funkcije gubitka u odnosu na ovu konkretnu težinu se izražava kao:

S obzirom da ovisi o , ovisi o i ovisi o , lančano pravilo kaže da, kako bi se derivirala u odnosu na , uzima se derivacija kompozicije funkcija.

To je matematički izraženo na sljedeći način:

Neka se promotri prvi izraz, .

Kao što je već spomenuto, zna se da vrijedi:

Stoga vrijedi:

Kada se proširi suma, može se vidjeti da vrijedi:

=

To znači da će gubitak mreže, u slučaju jedne ulazne vrijednosti, reagirati malom promjenom u aktivacijskom izlazu čvora 1 u sloju *L*. Količina promjene jednaka je razlici aktivacijskog izlaza za čvor 1 i željenog izlaza za čvor 1 koja je potom pomnožena brojem 2.

Neka se promotri drugi izraz:

Zna se da za svaki čvor *j* u izlaznom sloju *L* vrijedi:

i, s obzirom da je j = 1, vrijedi:

Stoga vrijedi:

Dakle, to je samo direktna derivacija od jer je direktna funkcija od .

Neka se, konačno, promotri treći izraz: .

Zna se da za svaki čvor *j* u izlaznom sloju *L* vrijedi:

S obzirom da je j = 1, vrijedi:

Stoga vrijedi:

= .

Kada se proširi suma, dobije se:

To znači da će ulaz za čvor 1 u sloju *L* reagirati na promjenu težine količinom koja je jednaka aktivacijskom izlazu čvora 2 u prethodnom sloju *L-1*.

Udruživanjem svih izraza, konačni rezultat je:

Opisano je kako se računa derivacija funkcije gubitka u odnosu na individualnu težinu kada je u pitanju jedan ulaz za treniranje. Kako bi se izračunala derivacija funkcije gubitka u odnosu na ovu istu težinu za svih *n* ulaznih vrijednosti iz skupa za treniranje, računa se prosječna derivacija funkcije gubitka u odnosu na svih *n* ulaza iz skupa za treniranje.

Matematički se to može izraziti kao:

Nakon toga bi se cijeli ovaj proces ponovio za svaku pojedinu težinu u mreži kako bi se izračunala derivacija funkcije gubitka u odnosu na svaku pojedinu težinu veze u mreži.

## POVRAT U POVRATNOJ PROPAGACIJI

Do sada je bilo opisano kako se može izračunati gradijent funkcije gubitka korištenjem povratne propagacije. Međutim, nije spomenuto gdje točno dolazi taj pokret prema natrag u povratnoj propagaciji.

U prethodnom poglavlju je prikazano kako se računa gradijent funkcije gubitka u odnosu na bilo koju težinu u mreži. Primjer je dan na jednoj konkretnoj težini u izlaznom sloju mreže, . Nakon toga se taj proces poopćio jer se on može primijeniti na svaku težinu veze pojedinačno.

Za ovu konkretnu težinu, pokazalo se da je derivacija gubitka u odnosu na tu težinu jednaka sljedećem izrazu

Kada je u pitanju težina koja se ne nalazi u izlaznom sloju, na primjer , formula za gradijent gubitka, koja je ponovljena maloprije, može se iskoristiti i za ovu težinu. Formula tada glasi:

Jedina razlika je ta što se razlikuju natpisi, jer se u ovom slučaju radi s trećim slojem u mreži, koji se označava s *L-1,* i indeksi jer se radi s težinom koja povezuje drugi čvor u drugom sloju s drugim čvorom u trećem sloju.

Međutim, iako su drugi i treći izraz na desnoj strani formule isti i računat će se na već definirani način, prvi izraz na desnoj strani jednakosti jest derivacija gubitka u odnosu na aktivacijski izlaz koji će se morati izračunati na drugačiji način.

Kada se računala derivacija gubitka u odnosu na težinu u izlaznom sloju, prvi izraz je činila derivacija gubitka u odnosu na aktivacijski izlaz čvora koji se nalazi u izlaznom sloju. Kao što je već objašnjeno, gubitak je direktna funkcija aktivacijskog izlaza za sve čvorove u izlaznom sloju. Gubitak je zbroj kvadriranih pogreški između pravih oznaka za podatke i aktivacijskih izlaza čvorova u izlaznom sloju.[1]

Tako, kada se računa derivacija gubitka u odnosu na težinu koja se nalazi u sloju *L-1*, prvi izraz s desne strane jest derivacija gubitka u odnosu na težinu koja se ne nalazi u izlaznom sloju *L*, već u sloju *L-1*. U ovom slučaju, za razliku od aktivacijskog izlaza čvora u izlaznom sloju, funkcija gubitka nije direktna funkcija izlaza čvora koji se nalazi u sloju *L-1*. Potrebno je uzeti u obzir gdje se nalazi aktivacijski izlaz u mreži i gdje se računa gubitak na kraju mreže. Izlaz se ne prosljeđuje direktno gubitku.[1]

Neka se usporede sljedeće funkcije:

i

Kako bi se prikazao način na koji se može izračunati derivacija funkcije u gubitka u odnosu na aktivacijski izlaz čvora koji se nalazi bilo gdje u mreži, a ne samo u izlaznom sloju, radit će se s jednim aktivacijskim izlazom. Konkretno, aktivacijskim izlazom čvora 2 u sloju *L-1*.

Aktivacijski izlaz je označen kao . Parcijalna derivacija funkcije gubitka u odnosu na ovaj aktivacijski izlaz označena je kao .

Neka se promotri da, za svaki čvor *j* u sloju *L*, funkcija gubitka C0 ovisi o , ovisi o . ovisi o svim težinama, koje čvor *j* spajaju s prethodnim slojem *L-1*, kao i svim aktivacijskim izlazima iz sloja *L-1*. To onda znači da ovisi i o .

Sada aktivacijski izlaz za svaki od čvorova u izlaznom sloju ovisi o ulazu u svaki od tih čvorova. Nadalje, ulaz u svaki od čvorova u izlaznom sloju ovisi i o težinama koje povezuju svaki od tih čvorova s prethodnim slojem *L-1* i o aktivacijskim izlazima čvorova koji se nalaze u sloju *L-1*.

Vraćajući se na konkretni primjer, može se vidjeti kako ulaz u svaki čvor u izlaznom sloju ovisi o aktivacijskom izlazu čvora 2 u sloju *L-1*.

Tako je funkcija gubitka opet kompozicija funkcija te se računa derivacija u odnosu na aktivacijski izlaz s kojim se radi. Koristi se lančano pravilo koje kaže da, kako bi izračunali derivaciju u odnosu na , uzima se derivacija kompozicije funkcija koja sada izgleda ovako:

Ovo je suma za svaki čvor *j* u izlaznom sloju *L*. Formula izgleda skoro identično kao i ona za izračun derivacije funkcije gubitka u odnosu na određenu težinu s najvećom razlikom da ova formula sadrži izraz za sumu. Razlog zašto nova formula u svom izrazu sadrži sumu je taj što će aktivacijski izlaz prethodnog sloja utjecati na svaki pojedini ulaz u svaki pojedini čvor *j* u sljedećem sloju *L*.

Još jedna od razlika je treći izraz u formuli koji računa derivaciju ulaza u bilo koji čvor *j* u izlaznom sloju *L* u odnosu na aktivacijski izlaz iz čvora 2 u sloju *L-1*, .

Za svaki čvor *j* u sloju *L* vrijedi:

Stoga se izraz može zamijeniti izrazom s desne strane jednakosti i uvrstiti u izraz koji računa derivaciju ulaza u bilo koji čvor *j* u izlaznom sloju *L* u odnosu na aktivacijski izlaz iz čvora 2 u sloju *L-1* te se dobije sljedeće:

Proširenjem sume se dobije:

Zbog toga što je operacija sume linearna operacija, može se uzeti derivacija svakog pojedinog izraza u odnosu na . Međutim, samo jedan izraz u sebi sadrži izraz . To znači da, kada se računaju derivacije koje ne sadrže izraz , vrijednosti tih izraza će biti jednake nuli. Rezultat će se dobiti deriviranjem izraza koji sadrži izraz .

Taj rezultat govori da će ulaz u bilo koji čvor *j* u sloju *L* odgovarati promjeni aktivacijskog izlaza čvora 2 u sloju *L-1* količinom jednakom težini veze koja povezuje čvor 2 u sloju *L-1* s čvorom *j* u sloju *L*.

### SPAJANJE IZRAZA

Tako konačna formula za izračun derivacije funkcije gubitka u odnosu na aktivacijski izlaz za dani primjer izgleda sljedeće:

Ukratko, potrebno je prvo izračunati derivacije koje ovise o komponentama koje se nalaze kasnije u mreži i onda iskoristiti te derivacije kod računanja gradijenta funkcije gubitka u odnosu na težine koje se nalaze ranije u mreži. To se postiže tako što se lančano pravilo primjenjuje unatrag kroz mrežu.

### PROSJEČNA DERIVACIJA FUNKCIJE GUBITKA

Kako bi se pronašla derivacija funkcije gubitka u odnosu na isti aktivacijski izlaz kod svih *n* uzoraka za treniranje, računa se prosječna derivacija funkcije gubitka kod svih *n* uzoraka za treniranje. To se matematički može izraziti na sljedeći način:

## NESTAJUĆI I EKSPLODIRAJUĆI GRADIJENT

Nažalost, tijekom treniranja se mogu javiti dva česta problema. Prvi je problem nestajućeg gradijenta (eng. *vanishing gradient problem*). Drugi je problem eksplodirajućeg gradijenta (eng. *exploding gradient problem*).

Općenito, problem nestajućeg gradijenta uzrokuje velike poteškoće tijekom treniranja neuronske mreže. To je problem koji vezan za težine koje se nalaze između ranijih slojeva u mreži.

Kao što je spomenuto u prethodnom poglavlju, stohastički gradijentni spust računa vrijednost gradijenta u odnosu na težine u neuronskoj mreži s ciljem da smanji vrijednost funkcije gubitka. U nekim situacijama se dogodi da je vrijednost gradijenta u odnosu na težine, koje se nalaze u prvim slojevima u mreži, postane jako malena, gotovo nestajuća. Pod pojmom male vrijednosti se misli na one vrijednosti koje su manje od 1. Nadalje, ako je vrijednost gradijenta tako mala, onda će i promjena težina biti jako mala.

Tako mala promjena težine neće dovoljno utjecati na mrežu kako bi se smanjio gubitak jer se mreža jedva pomaknula s mjesta na kojem je bila prije promjene težina. Posljedica je da je težina određene veze na neki način 'zapela' na mjestu i jako će se sporo mijenjati kako bi se približila svojoj optimalnoj vrijednosti. Posljedica toga je da se smanjuje i sama sposobnost mreže da uči.[1]

Vraćajući se na matematičke izvode za propagaciju prema natrag, može se primijetiti da, što se veza nalazi ranije u mreži, bit će potrebno više parametara kako bi se izračunao njen gradijent. Stoga, ako više tih parametara ima vrijednost manju od 1, umnožak tih parametara će biti još manji. Dobivena mala vrijednost se zatim koristi kako bi se promijenila težina određene veze. Ali, prije nego se dobivena vrijednost iskoristi za promjenu težine, ona se množi sa stopom učenja koja sama po sebi ima malu vrijednost, između 0.01 i 0.0001. Tako je ukupan rezultat još manji broj. Onda se taj broj oduzima od stare težine kako bi se dobila nova težina.[1]

Ovako, sama težina veze postaje na neki način 'zaglavljena'. Drugim riječima, jako se sporo odmiče od svoje stare vrijednosti i na taj način ne pomaže u minimiziranju gubitka.

S druge strane postoji problem eksplodirajućeg gradijenta. Taj problem se javlja kada su parametri potrebni za izračun gradijenta u odnosu na određenu težinu, koja se nalazi rano u mreži, puno veći od jedan.

Kao i kod problema nestajućeg gradijenta, kada se množi veliki broj parametara, čija je vrijednost puno veća od 1, vrijednost gradijenta će 'eksplodirati'. Međutim, umjesto da se težina jedva pomakne s mjesta, njena će se vrijednost jako promijeniti. Toliko će se promijeniti da može lako 'preskočiti' svoju optimalnu vrijednost.[1]

# PARAMETRI I HIPER-PARAMETRI U UMJETNIM NEURONSKIM MREŽAMA

Već se dosta toga spomenulo i opisalo o parametrima koje umjetna neuronska mreža može naučiti, kao i o hiper-parametrima koji se za svaku mrežu određuju prije procesa učenja. Međutim, do sada se nije dao koncizni popis što sve neuronska mreža može naučiti kao ni na što je potrebno obratiti pažnju tijekom konstruiranja neuronske mreže.

U ovom poglavlju će se definirati što je to parametar koji neuronska mreža može naučiti. Nakon toga će se opisati primjer kako se broj parametara računa kod tradicionalnih neuronskih mreža i kod konvolucijskih neuronskih mreža te će oba opisa biti potkrijepljena primjerima.

Nakon toga će se opisati što su to hiper-parametri i koji je njihov utjecaj na sam proces učenja neuronske mreže kao i na same performanse neuronske mreže

## PARAMETRI

Parametar je svaki pojam čiju optimalnu vrijednost neuronska mreža pokušava naučiti tijekom procesa učenja, to su težine veza i pragovi neurona. Broj parametara se računa za svaki sloj neuronske mreže pojedinačno. Ukupan broj parametara čini zbroj parametara svakog pojedinog sloja neuronske mreže.[1]

Kako bi se izračunao ukupan broj parametara u jednom sloju tradicionalne neuronske mreže, prvo je potrebno izračunati nekoliko stvari:

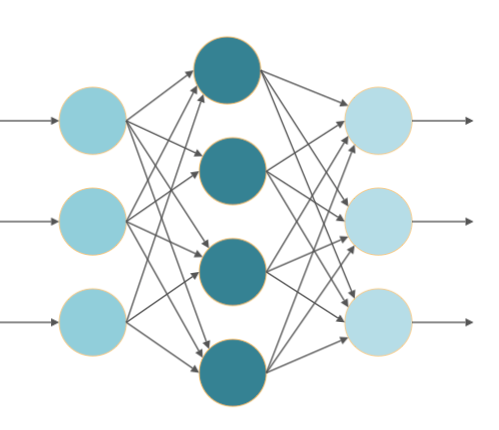
1. broj ulaza u određeni sloj
2. broj izlaza iz određenog sloja
3. ako sloj sadrži pragove, koliko ih sadrži.[1]

Ovdje je potrebno napomenuti da se misli na potpuno povezanu mrežu (eng. *fully.connected network*) koja je sačinjena od gustih slojeva(eng. *dense layers*).

Ukupan broj parametara za dani sloj se računa sljedećom formulom:

Na primjer, neka je dana potpuno povezana neuronska mreža koja ima 3 sloja sa sljedećim brojem čvorova:

1. ulazni sloj s 3 čvora
2. skriveni sloj s 4 čvora
3. izlazni sloj s 3 čvora



Također, neka mreža sadržava pragove. To znači da svaki sloj u skrivenom i izlaznom sloju sadrži prag.

Broj parametara ove neuronske mreže se računa na sljedeći način:

1. Ulazni sloj ne sadrži parametre koji se mogu naučiti jer on sadrži samo ulazne podatke.
2. Broj ulaza u skriveni sloj jednak je broju čvorova prethodnog, odnosno ulaznog, sloja. To je broj 3. Broj izlaza iz skrivenog sloja jednak je broju čvorova u skrivenom sloju, 4. Ta dva broja se pomnože i dobivena vrijednost jednaka je broju težina, 12. Kako bi se dobio ukupan broj parametara za skriveni sloj, umnožak se zbraja s brojem pragova koji je opet jednak broju čvorova skrivenog sloja, 4. Ukupan broj parametara u skrivenom sloju je 16.
3. Na isti način se računa broj parametara za izlazni sloj i dobije se broj od parametara koji se mogu naučiti u izlaznom sloju.
4. Zbroje se rezultati skrivenog i izlaznog sloja te se dobije ukupno 31 parametar koji ova neuronska mreža može naučiti tijekom procesa učenja.

Kao i kod tradicionalne neuronske mreže, broj parametara u konvolucijskoj mreži se računa za svaki sloj posebno. Ukupan broj parametara se dobije zbrajanjem parametara iz svakog pojedinog sloja.

Međutim, konvolucijski slojevi nisu gusti slojevi te sadrže filtere. O tome je potrebno voditi računa jer se formula za izračun broja parametara modificira s obzirom na to:

1. je li prethodni sloj konvolucijskog sloja bio gusti sloj. Ako jest, ulaz u konvolucijski sloj je broj čvorova koji se nalaze u prethodnom sloju.
2. Ako je prethodni sloj određenog konvolucijskog sloja također konvolucijski sloj, broj ulaza u taj određeni sloj je jednak broju filtera koje posjeduje taj prethodni konvolucijski sloj.[1]

Također, broj izlaza iz konvolucijskog sloja jednak je umnošku broja filtera s njihovim dimenzijama. Konačno, broj pragova jednak je broju filtera u danom konvolucijskom sloju.

Na primjer, neka konvolucijska neuronska mreža sadrži ulazni sloj, dva skrivena konvolucijska sloja i jedan gusti sloj kao izlazni sloj:

1. ulazni sloj koji sadrži podatke dimenzija 10x10x3
2. prvi skriveni konvolucijski sloj koji sadrži 3 filtera dimenzija 3x3
3. drugi skriveni konvolucijski sloj koji sadrži 2 filtera dimenzija 3x3
4. gusti izlazni sloj koji sadrži 2 izlazna neurona

Broj parametara se računa na sljedeći način:

1. Kao i kod tradicionalnih neuronskih mreža, ulazni sloj nema parametre koji se mogu naučiti jer on samo sadrži ulazne podatke.
2. Broj ulaza u prvi konvolucijski sloj je 3. Broj izlaza iz tog sloja je umnožak broja filtera s njihovim dimenzijama, odnosno . Broj ulaza u sloj se pomnoži s brojem izlaza iz sloja i dobije se težina u prvom konvolucijskom sloju koja se može naučiti. Ukupan broj parametara jednak je .
3. Broj ulaza u drugi konvolucijski sloj je 3 jer prethodni sloj sadrži 3 filtera. Broj izlaza iz drugog konvolucijskog sloja jednak je . Umnožak ulaza i izlaza daje broj . Ukupan broj parametara drugog konvolucijskog sloja je .
4. Neka se koristi *zero padding* kako bi se sačuvale dimenzije ulaznog kanala. Onda je broj ulaza u izlazni sloj jednak . Broj izlaza jednak je broju neurona u izlaznom sloju, 2. Konačan broj parametara za izlazni sloj jednak je .

Ukupan broj parametara koje konvolucijska mreža mora naučiti tijekom procesa učenja je .

## HIPERPARAMETRI

Hiperparametri su vanjski parametri neuronske mreže koje postavlja operator neuronske mreže. Oni imaju veliki utjecaj na preciznost neuronske mreže i njenu sposobnost učenja. Iako postoji više načina pronalaska optimalnih vrijednosti hiperparametara, najjednostavniji način je metoda pokušaja i pogreške.[30]

Hiperparametri su sljedeći:

1. Broj skrivenih slojeva – dodavanjem više skrivenih slojeva se povećava preciznost predviđanja, ali se isto tako povećava rizik od prenaučenosti.
2. *Dropout* – koliki će se postotak nasumično odabranih neurona ignorirati tijekom svake epohe kako bi se spriječila prenaučenost. Ali s druge strane, preveliki postotak ignoriranih neurona može dovesti do podnaučenosti.
3. Aktivacijska funkcija – koja se aktivacijska funkcija koristi za obradu ulaznih primjera. Aktivacijska funkcija utječe na preciznost i brzinu učenja mreže.
4. Inicijalizacija težina – prije samog učenja neuronske mreže, potrebno je njenim vezama pridijeliti neke težine. Dvije su osnovne opcije, postavljanje svih težina na nulu ili postavljanje nasumičnih vrijednosti. S obje opcije postoji opasnost od problema nestajućeg ili eksplodirajućeg gradijenta. Jedna od alternativa takvih inicijalizacija težina jest Xavier-ova heuristička inicijalizacija težina.
5. Stopa učenja neuronske mreže – koliko se brzo algoritam propagacije prema natrag približava minimalnom gubitku. Niža stopa učenja produljuje vrijeme učenja, ali zato izbjegava mogućnost dobivanja pretjeranog rezultata (eng. *overshooting*).
6. Broj epoha, veličina grupe – ovi parametri određuju kojom će se brzinom primjeri iz skupa za učenje prosljeđivati mreži.
7. Algoritam optimizacije – osnovna opcija jest korištenje stohastičkog gradijentnog spusta. Alternative su algoritmi kao što su Momentum, Nester-ov akcelerirani gradijent ili AdaDelta.[30]

# IMPLEMENTACIJA JEDNOSTAVNE KONVOLUCIJSKE MREŽE ZA SEMANTIČKU SEGMENTACIJU SLIKA

U ovom poglavlju će se implementirati jednostavne konvolucijske neuronske mreže korištenjem Keras API-a koji je podržan od strane TensorFlow 2 platforme za strojno učenje. Prvo će se ukratko opisati Keras API. Nakon toga će se navesti i opisati aktivacijske funkcije u situacijama kada se one zovu kao argument već postojećeg sloja i kada se one zovu kao zasebni, aktivacijski, slojevi.

Sam praktični dio će se baviti usporedbom jednostavnih konvolucijskih mreža kada se koriste različiti aktivacijski slojevi s različito postavljenim parametrima.

## KERAS API

Keras je API za duboko učenje napisan u Python programskom jeziku. Pokreže se na TensorFlow platformi za strojno učenje. Ovaj API je razvijen kako bi inženjerima i znanstvenicima omogućio brzo eksperimentiranje tijekom istraživanja.

### KERAS I TENSORFLOW 2

TensorFlow 2 je *end-to-end*, *open-source* platforma za strojno učenje. Ona kombinira 4 ključne sposobnosti:

1. Efikasno izvodi jednostavne operacije s matricama korištenjem CPU-a, GPU-a ili TPU-a.
2. Izračunava gradijent proizvoljnih izraza koji imaju sposobnost deriviranja.
3. Skalira izračune na mnogim uređajima.
4. Izvozi programe (grafove) na vanjske *runtime*-ove poput servera, web preglednika, mobilnih i ugradbenih uređaja.

Keras je TensorFlow-ov API visoke razine. To je jako pristupačno, visoko-produktivno sučelje koje se koristi za rješavanje problema strojnog učenja i koje je specijalizirano za duboko učenje. Pruža bitne apstrakcije i gradivne blokove za razvoj i izvoz rješenja strojnog učenja s velikom brzinom izvođenja iteracija.

Keras se može izvoditi na TPU-ima (*Tensor Processing Unit*) i velikim grozdovima GPU-a (*Graphic Processing Unit*). On dolazi u paketu s TensorFlow 2 platformom te mu se pristupa korištenjem izraza tensorflow.keras. Za korištenje Keras API-a, potrebno je samo instalirati TensorFLow 2.

Keras/TensorFlow 2 je kompatibilan s:

1. Python-ovim verzijama 3.5-3.8
2. Ubuntu 16.04 i kasnijim verzijama
3. Windows 7 i kasnijim verzijama
4. macOs 10.12.6 (Sierra) i kasnijim verzijama

Keras (κέρας) na grčki znači 'rog'. Njegovo ime je referenca na pjesničku sliku iz antičke grčke i latinske književnosti koja se prvi put susreće u Homerovoj Odiseji. Mitološki duhovi (*Oneiros*) koji prolaze kroz vrata roga objavljuju budućnost koja će se sigurno dogoditi. Keras je inicijalno bio razvijen u sklopu istraživanja na projektu ONEIROS (*Open-ended Neuro-Electronic Intelligent Robot Operating System*).[31]

## AKTIVACIJSKE FUNKCIJE U KERAS API-U

Aktivacijske funkcije se mogu koristiti odvojeno u aktivacijskom sloju ili korištenjem 'activation' argumenta u ostalim slojevima u neuronskoj mreži.

*Placeholder za primjer*

Ovo je jednako sljedećem kodu:

*Placeholder za inačicu*

Sve aktivacijske funkcije mogu se pozvati korištenjem njihovog *string* identifikatora:

*Placeholder za string identifikator*[32]

### RELU AKTIVACIJSKA FUNKCIJA

*Placehoder za kod s relu*

Argumenti ReLU aktivacijske funkcije su:

1. x: ulazna matrica ili varijabla
2. alpha: decimalni broj koji upravlja nagibom za vrijednosti koje su niže od vrijednosti praga
3. max\_value: decimalni broj kojim se postavlja najveća vrijednost koju će funkcija vratiti
4. threshold: decimalni broj kojim se postavlja vrijednost praga ispod koje će aktivacijska funkcija vratiti nulu

Ova funkcija na izlazu vraća matricu jednakih dimenzija kao i ulazna matrica x. Ta matrica predstavlja ulaznu matricu čije su vrijednosti izmijenjene korištenjem ReLU aktivacijske funkcije.[32]

### SIGMOIDNA AKTIVACIJSKA FUNKCIJA

*Placeholder za sigmoid*

Za male vrijednosti (manje od pet), sigmoidna funkcija će vratiti vrijednosti koje se nalaze blizu nule. Za velike vrijednosti (veće od pet) funkcija će vratiti vrijednosti koje se nalaze blizu jedan.

Sigmoidna aktivacijska funkcija ekvivalentna je softmax funkciji koja prima 2 argumenta, ali gdje je drugi argument postavljen na vrijednost nula. Sigmoidna funkcija će uvijek vratiti vrijednost koja se nalazi između nula i jedan.

Argumenti sigmoidne aktivacijske funkcije su:

1. x: ulazna matrica ili varijabla

Ova funkcija vraća matricu jednakih dimenzija kao i ulazna matrica. Vrijednosti izlazne matrice rezultat su djelovanja sigmoidne funkcije na vrijednosti ulazne matrice.[32]

### SOFTMAX AKTIVACIJSKA FUNKCIJA

*Placeholder za softmax*

Softmax funkcija uzima vektor realnih vrijednosti i vraća vektor kategoričkih vjerojatnosti. Elementi izlaznog vektora se nalaze u opsegu (0, 1). Zbroj svih elemenata izlaznog vektora daju jedan. Ako funkcija prima više vektora, ona će sa svakim vektorom raditi pojedinačno.

Softmax funkcija se često koristi kao aktivacijska funkcija u posljednjem sloju u neuronskoj mreži jer se koristi za klasifikaciju ulaznih primjera. Razlog je što se njen rezultat može tumačiti kao distribucija vjerojatnosti.

Argumenti softmax aktivacijske funkcije su:

1. x: ulazna matrica ili varijabla
2. axis: cjelobrojna vrijednost, označava os uz koju se primjenjuje softmax funkcija

Ova funkcija na izlazu vraća matricu čije su vrijednosti ne-negativne i čiji cjelokupni zbroj daje jedan.

Ovdje je potrebno napomenuti da korištenje ove funkcije može uzrokovati ValueError grešku ako je dimenzija matrice dim(x)==1.[32]

### SOFTPLUS AKTIVACIJSKA FUNKCIJA

*Placeholder za softplus*

Softplus aktivacijska funkcija izražava se kao: .

Argumenti softplus aktivacijske funkcije su:

1. x: ulazna matrica

Ova funkcija vraća softplus aktivaciju.

### SOFTSIGN AKTIVACIJSKA FUNKCIJA

*Placeholder za softsign*

Softsign aktivacijska funkcija izražava se kao: .

Argumenti softsign aktivacijske funkcije su:

1. x: ulazna matrica

Ova funkcija na izlazu daje vrijednost softsign aktivacije.

### TANH AKTIVACIJSKA FUNKCIJA

*Placeholder za tanh*

Argumenti tanh (tangens hiperboli) funkcije su:

1. x: ulazna matrica

Ova funkcija na izlazu daje matricu jednakog oblika i s istim tipovima vrijednosti kao i ulazna matrica x.[32]

### SELU AKTIVACIJSKA FUNKCIJA

*Placeholder za selu*

SELU je kratica od *Scaled Exponential Linear Unit* i definirana je kao:

* ako je x>0: vrati
* ako je x<0: vrati

Gdje su alpha i scale predefinirane konstante (alpha=1.67326324, scale=1.05070098).

SELU aktivacijska funkcija množi konstantu scale s izlazom tf.keras.activations.elu aktivacijske funkcija kako bi se dobio veći nagib za pozitivne ulaze.

Vrijednosti alpha i scale odabrane su na način da srednje vrijednosti i varijance ulaza budu sačuvane između dva uzastopna sloja dokle god su težine veza ispravno inicijalizirane i dokle god je broj ulaza 'dovoljno velik'.

Argumenti SELU funkcije su:

1. x: ulazna matrica ili skalarna vrijednost

Funkcija na izlazu daje skaliranu ELU aktivaciju: scale\*elu(x, alpha).[32]

### ELU AKTIVACIJSKA FUNKCIJA

*Placeholder za elu*

ELU je kratica od *Exponential Linear Unit*.

Ako je parametar alpha veći od nule, funkcija vraća sljedeće vrijednosti:

1. x ako je x>0
2. alpha\*(exp(x)-1) ako je x<0

ELU hiper-parametar alpha kontrolira vrijednost za koju će se ELU zasititi kod negativnih ulaza u mrežu. ELU funkcija rješava problem nestajućeg gradijenta.

ELU funkcija će vratiti negativne vrijednosti koje će srednju vrijednost aktivacije pomicati prema nuli. Srednje vrijednosti aktivacija koje se nalaze bliže nuli omogućuju brže učenje jer gradijent pomiču bliže prirodnom gradijentu.

Argumenti ELU funkcije su:

1. x: ulazna matrica
2. alpha: skalar, nagib kod negativne sekcije

Ova funkcija na izlazu daje vrijednost ELU funkcije.[32]

### EXPONENTIAL AKTIVACIJSKA FUNKCIJA

*Placeholder za exponential*

Argumenti koje prima exponential funkcija su:

1. x: ulazna matrica

Ova funkcija na izlazu daje matricu čije su vrijednosti rezultat djelovanja eksponencijalne funkcije exp(x) na vrijednosti ulazne matrice.[32]

## AKTIVACIJSKI SLOJEVI U KERAS API-U

Aktivacijski slojevi Keras API-a većinom čine ReLU i njene inačice s izuzetkom softmax i ELU sloja.

### ReLU SLOJ

*Placeholder za ReLU sloj*

S osnovno postavljenim vrijednostima, ovaj sloja vraća max(x, 0), dok inače slijedi pravila:

*Placeholder za pravila*

S tim da je input\_shape proizvoljan argument koji se koristi kada je ReLU sloj prvi skriveni sloj u neuronskoj mreži.

Izlaz iz ReLU sloja istog je oblika kao i ulaz.

Argumenti ReLU sloja su:

1. max\_value: decimalni broj koji je veći ili jednak nuli; predstavlja maksimalnu izlaznu vrijednost iz sloja. Osnovna postavka ovog parametra je postavljena na None što znači da je ona neograničena.
2. negative\_slope: decimalni broj koji je veći ili jednak nuli; predstavlja koeficijent negiba za negativne brojeve. Osnovna postavka ovog parametra je postavljena na nulu što znači da negativne vrijednosti neće imati nagib.
3. threshold: decimalni broj; predstavlja vrijednost praga ispod kojeg će neuron postati neaktivan. Osnovna postavka ovog parametra je postavljena na nulu. [33]

### SOFTMAX SLOJ

*Placeholder za softmax*

S tim da je input\_shape proizvoljan argument koji se koristi kada je softmax sloj prvi skriveni sloj u neuronskoj mreži. Izlaz ovog sloja jednakog je oblika kao i ulaz u sloj.

Argumenti softmax sloja su:

1. axis: cijeli broj ili lista cjelobrojnih vrijednosti. To je os duž koje se primjenjuje softmax normalizacija

Ostali, proizvoljni, argumenti su:

1. inputs: ulazi u softmax sloj
2. mask: maska s bool vrijednostima; istog je oblika kao i inputs argumenti. Osnovna vrijednost je postavljena na None. [33]

### LeakyReLU SLOJ

*Placeholder za leakyrelu*

S tim da je input\_shape proizvoljan argument koji se koristi kada je LeakyReLU sloj prvi skriveni sloj u neuronskoj mreži. Izlaz ovog sloja jednakog je oblika kao i ulaz u sloj.

LeakyReLU sloj ostvaruje mali gradijent kada neuron nije aktiviran čime se nastoji izbjeći problem nestajućeg ReLU-a.

*Placeholder za situacije*

Argumenti LeakyReLU sloja su:

1. alpha: decimalni broj koji je veći ili jednak od nule; predstavlja koeficijent nagiba za negativne vrijednosti. Osnovna vrijednost mu je postavljena na 0.3. [33]

### PReLU SLOJ

*Placeholder za prelu*

S tim da je input\_shape proizvoljan argument koji se koristi kada je PReLU sloj prvi skriveni sloj u neuronskoj mreži. Izlaz ovog sloja jednakog je oblika kao i ulaz u sloj.

PReLU je parametarska verzija ReLU aktivacije koja slijedi pravila:

*Placeholder za pravila*

Gdje je alpha parametar kojeg mreža uči tijekom procesa učenja i koji je istog oblika kao i ulaz u sloj x.

Argumenti PReLU sloja su:

1. alpha\_initializer: inicijalizacijska funkcija koja služi za postavljanje inicijalnih vrijednosti težina veza
2. alpha\_regularizer: regulator težina
3. alpha\_constant: ograničenja težina
4. shared\_axes: osi uz koje se dijele naučeni parametri aktivacijske funkcije. Na primjer, ulazna matrica značajki iz 2D konvolucije ima oblik (batch, height, width, channels) i cilj je da se ti parametri prostorno podijele tako da svaki filter ima samo jedan skup parametara. Tada se ovaj argument postavi na vrijednost shared\_axes=[1, 2]. [33]

### ELU SLOJ

*Placeholder za elu*

S tim da je input\_shape proizvoljan argument koji se koristi kada je PReLU sloj prvi skriveni sloj u neuronskoj mreži. Izlaz ovog sloja jednakog je oblika kao i ulaz u sloj.

Ova funkcija slijedi pravila:

*Placeholder za pravila elu-a*

Argumenti ELU sloja su:

1. alpha: koeficijent za negativni faktor[33]

### ThresholdedReLU SLOJ

*Placeholder za trelu*

S tim da je input\_shape proizvoljan argument koji se koristi kada je PReLU sloj prvi skriveni sloj u neuronskoj mreži. Izlaz ovog sloja jednakog je oblika kao i ulaz u sloj.

Ova funkcija slijedi pravila:

*Placeholder za pravila trelua*

Argumenti ThresholdedReLU sloja su:

1. theta: decimalni broj koji je veći ili jednak nuli; predstavlja lokaciju praga aktivacije.[33]

## USPOREDBA IZVEDBI AKTIVACIJSKIH FUNKCIJA

# LITERATURA

[1] <https://deeplizard.com>

[2] <https://ch.mathworks.com/discovery/deep-learning.html>

[3] <https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/>

[4] <https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-networks-image-classification/>

[5] <https://missinglink.ai/guides/neural-network-concepts/complete-guide-artificial-neural-networks/>, 3.3.2021.

[6] <https://ch.mathworks.com/discovery/convolutional-neural-network-matlab.html>

[7] <https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/>

[8] <https://arxiv.org/pdf/1511.08458.pdf>

[9] Udeme Udofia, <https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17>, 3.3.2021.

[10] <https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-tutorial-basic-advanced/>, 3.3.2021.

[11] <https://towardsdatascience.com/convolutional-neural-networks-mathematics-1beb3e6447c0>

[12] <https://machinelearningmastery.com/crash-course-convolutional-neural-networks/>

[13] <https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/>

[14] <https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/>

[15] <https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9>

[16] <https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529>

[17] <https://stackoverflow.com/questions/52272592/how-many-neurons-does-the-cnn-input-layer-have/52273707>

[18] Mukesh Chaudhary, <https://medium.com/@cmukesh8688/activation-functions-sigmoid-tanh-relu-leaky-relu-softmax-50d3778dcea5>, 3.3.2021.

[19] Sagar Sharma, https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, 3.3.2021.

[20] Thomas Wood, <https://deepai.org/machine-learning-glossary-and-terms/sigmoid-function>, 3.3.2021. ->[20]

[21] autor nije imenovan, <https://paperswithcode.com/method/tanh-activation>, 3.3.2021.

[22] Danqing Liu,<https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7>, 3.3.2021.

[23] Andre Ye,<https://medium.com/@neuralnets/swish-activation-function-by-google-53e1ea86f820>, 3.3.2021.

[24] Andre Ye,<https://towardsdatascience.com/swish-booting-relu-from-the-activation-function-throne-78f87e5ab6eb>, 3.3.2021.

[25] Thomas Wood, <https://deepai.org/machine-learning-glossary-and-terms/softmax-layer>, 3.3.2021.

[26] <https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network>

[27] <https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73>

[28] <https://www.fer.hr/_download/repository/UmjetneNeuronskeMreze.pdf>

[29] <https://machinelearningmastery.com/why-training-a-neural-network-is-hard/>

[30] <https://missinglink.ai/guides/neural-network-concepts/hyperparameters-optimization-methods-and-real-world-model-management/>

[31] <https://keras.io/about/>

[32] <https://keras.io/api/layers/activations/>

[33] <https://keras.io/api/layers/activation_layers/>