# TRENIRANJE UMJETNIH NEURONSKIH MREŽA

U ovom poglavlju će se dati općeniti pregled procesa treniranja umjetnih neuronskih mreža. Nakon što se umjetna neuronska mreža strukturirana, potrebno ju je trenirati kako bi naučila preslikavati dani ulaz na ispravan izlaz i kada dobije podatke koje prije nije vidjela. Ovo poglavlje će opisati pojmove kao što su skupovi podataka za treniranje, validaciju i testiranje te će navesti potrebu 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 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 vrijednosti u izlazne vrijednosti. Ona se intuitivno računa tako što se izlazna vrijednost neuronske mreže oduzme od željenog izlaza neuronske mreže za dani ulaz. Taj rezultat se konačno kvadrira jer su negativni rezultati jednako loši kao i pozitivni rezultati.[3]

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.

Gubitak je razlika između željenog izlaza za neuronsku mrežu i izlaza neuronske mreže. Kao što se može vidjeti iz 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

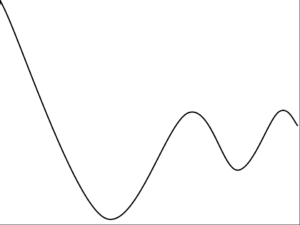
Tijekom strukturiranja neuronske mreže, težine veza se inicijaliziraju nasumičnim brojčanim vrijednostima. Očito je da rezultati predviđanja takve mreže neće biti zadovoljavajući. Cilj treniranja neuronske mreže jest završiti s mrežom koja ima visoku točnost predviđanja. U terminima funkcije gubitka, cilj procesa učenja je postići što manju vrijednost funkcije gubitka.[3]

Neuronska mreža tijekom procesa učenja uči preslikavati ulaze u izlaze na temelju danog skupa primjera za treniranje. Znanje o načinu obrade ulaznih podataka implicitno je pohranjeno u težinama veza između neurona. Treniranje uključuje proces iterativnog pronalaženja skupa težina u neuronskoj mreži sve do trenutka kada je izlazna vrijednost mreže, provjerena na skupu podataka za treniranje, 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 ulaze u odgovarajuće izlaze.[1][4][5]

Algoritam, koji se koristi za 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 jer se jako jednostavno izvodi, ali s druge strane, zahtijeva mnogo iteracija.[1][2]

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.[5]

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. [5]



## 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 samo ime kaže, temelji na gradijentu.

Prije nego se na matematički način krene 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 određena funkcija (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*.[3]

Gradijent je vektor parcijalnih derivacija. Njegovi elementi su derivacije funkcije u odnosu na varijable o kojima 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.[3]

* Potraži bolje objašnjenje na hrvatski za definiciju derivacije

## STOHASTIČKI GRADIJENTNI SPUST

Neka primjer čine sljedeća funkcija sa svojom derivacijom:

Funkcija:

Prva funkcija derivacije:

Minimum ove funkcije se nalazi u točki .

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

Neka računalo uzme neku nasumičnu vrijednost, na primjer . Derivacija funkcije u točki je 2. To znači da, kada funkcija učini beskonačno mali korak u pozitivnom smjeru, njena vrijednost će se promijeniti proporcionalno 2, 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 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 uzeti korak u pozitivnom smjeru.

Međutim, računalo ne zna koliko korak mora napraviti u negativnom smjeru jer derivacija samo garantira da će se, ako se uzme 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 zasebnom poglavlju.[3]

Kao što je već spomenuto, gradijent je vektor čiji su elementi parcijalne derivacije funkcije u odnosu na neke određene varijable. Taj vektor pokazuje u 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.[3]

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 one koje se trebaju mijenjati kako bi se poboljšalo predviđanje mreže.

Sada, neka se 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. Nema pravila koliko se velika stopa učenja treba uzeti tijekom svake iteracije. Međutim, ako se uzme prevelika stopa učenja, postoji mogućnost da će dobiveni rezultat za određenu težinu '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][3]

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 svaki model posebno kako bi se odredilo s kojom će stopom učenja model najbolje 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[2]

## 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). Tijekom svake epohe, model će iznova i iznova učiti značajke podataka na istom skupu podataka za treniranje. Cilj je na taj način naučiti model da točno predvidi značajke podataka koje do tada nije primijetio. Mreža će svoje predviđanja temeljiti na onome što je naučila na skupu podataka za treniranje.[1][4]

Pomoću primjera iz drugog skupa podataka 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 isto koristi tijekom procesa učenja. Također, 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 gubitaka iz skupa podataka za provjeru.[1][4]

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 koje do tada nije vidjela) 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][4]

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

Tu još malo govorit o enkodiranju podataka

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 ostala 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 ju se stavi u produkciju.[1]

Donje prikazana tablica ukratko opisuje sve što je prije 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 je 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 nema 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, može bolje generalizirati.

Još jedan način pomoću kojeg se može smanjiti prenaučenost jest na neki način 'povećati' podatke. Pod povećanjem podataka se misli na to da se stvore dodatni, izmijenjeni podaci koji su nastali razumnom modifikacijom primjera iz skupa za treniranje. Na primjer, kod slikovnih podataka, koji 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 učiniti ju crno-bijelom).[1]

Glavna ideja iza povećanja podataka 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 i slojevima.[1]

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

S druge strane, kaže se da je mreža podnaučena ako nije sposobna dobro preslikavati ni 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 podatke.[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 vrste slojeva u mreži.

Nadalje, još jedan od načina reduciranja podnaučenosti jest dodavanje oznaka ulaznim primjerima u skupu za treniranje. Dodatne oznake mogu pomoći mreži ispravnije klasificirati podatke.

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, još jedna od tehnika reduciranja podnaučenosti jest reduciranjem dropout-a. Kod korištenja te tehnike se može odrediti koliki će se postotak neurona ignorirati prilikom 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 cijelog skupa 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 kao 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 vrijedni da, što je veća grupa, neuronska mreža će brže završiti jednu epohu. Međutim, potrebno je uzeti u obzir na računalnu moć kojom raspolaže neuronska mreža. Čak i ako računalo može obraditi velike grupe 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 za računalo koje podržava tu određenu neuronsku mrežu.

## PRAG

Kad se misli na pojam praga 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, tako isto uči i mijenja vrijednosti pragova. Vrijednost praga odlučuje hoće li se izlazna vrijednost i 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 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]

NAPRAVIT SLIKU ZA OVAJ PRIMJER

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 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 vezi toga koje će vrijednosti aktivirati određeni neuron. Ovaj isti proces se može koristiti i kako bi se suzio opseg vrijednosti koje će aktivirati neuron.

# KORIŠTENA LITERATURA

[1] [www.deeplizard.com](http://www.deeplizard.com)

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

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

[4] https://www.fer.hr/\_download/repository/UmjetneNeuronskeMreze.pdf

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