# TRENIRANJE UMJETNIH NEURALNIH MREŽA

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

## TRENIRANJE UMJETNE NEURALNE MREŽE

U ovom poglavlju će se opisati što to znači trenirati umjetnu neuronsku mrežu. U jednom od prethodnih poglavlja opisana je osnovna arhitektura opće umjetne neuralne mreže. Nakon što je konfigurirana arhitektura modela, sljedeći korak je treniranje tog modela.

### ŠTO ZNAČI TRENIRATI UMJETNU NEURALNU MREŽU

Treniranjem modela, pokušava se riješiti problem optimizacije, odnosno pokušavaju se optimizirati težine veza u danom modelu. Zadatak optimizacije jest pronaći težine koje najispravnije preslikavaju ulazne podatke u klase predviđanja. Preslikavanje je ono što model, odnosno umjetna neuralna mreža, treba naučiti.

U jednom od prethodnih poglavlja je prikazano kako je svakoj vezi između dva čvora pridružena neka težina. Tijekom treniranja, težine se iterativno ažuriraju prema svojim optimalnim vrijednostima.

### OPTIMIZACIJSKI ALGORITAM

Težine veza se optimiziraju takozvanim optimizacijskim algoritmom. Optimizacijski proces ovisi o odabranom optimizacijskom algoritmu. Optimizacijski algoritam se u literaturi još naziva i 'optimizatorom' (eng. *optimizer*). Najpoznatiji optimizacijski algoritam se naziva 'Stohastički gradijentni spust' (eng. *stochastic gradient descent*), ili skraćeno SGD.

Kod bilo kojeg problema optimizacije, važno je imati cilj optimizacije. Ovdje će se opisati cilj optimizacije SGD-a prilikom optimizacije težine veza u modelu.

Cilj SGD-a jest minimizirati funkciju gubitka (eng. *loss function*). SGD ažurira težine veza u modelu na način da funkciju gubitka približi što je više moguće svojoj minimalnoj vrijednosti.

### FUNKCIJA GUBITKA

Funkcija gubitka, koja se često koristi u neuralnim mrežama, jest funkcija srednje kvadratne pogreške (eng. *mean squared error*), ili skraćeno MSE. Uz MSE postoji još nekoliko često korištenih funkcija gubitaka.

Što je to točno gubitak će se objasniti preko sljedećeg primjera.

Tijekom treniranja modela, model se opskrbljuje podacima i odgovarajućim oznakama (eng. *labels*) za te podatke.

NAPRAVIT SVOJ PRIMJER

Na primjer, neka se model trenira s ciljem da uspješno klasificira dolazne fotografije na fotografije mački i fotografije pasa. Model se opskrbljuje fotografijama mački i pasa zajedno s oznakama za fotografije koje govore je li ulazna fotografija mačke ili psa.

Neka se pretpostavi da je modelu dana fotografija mačke. Kad prosljeđivanje unaprijed dođe kraju i fotografija mačke je prošla kroz mrežu, model će na izlazu dati neku vrijednost. Ta vrijednost će pokazati misli li model da je dobio fotografiju mačke ili psa.

U doslovnom smislu, izlaz će se sastojati od vjerojatnosti za mačku ili psa. Na primjer, model na izlazu može dodijeliti vjerojatnost od 75% da je ulazna fotografija ona s mačkom i vjerojatnost od 25% da je ulazna fotografija ona sa psom.

* 75% vjerojatnosti da je na fotografiji mačka
* 25% vjerojatnosti da je na fotografiji pas

Gubitak je pogreška ili razlika između što mreža predviđa za sliku i onoga što je prava oznaka slike. SGD će pokušati minimizirati ovu pogrešku tako da model postane što točniji kod predviđanja. Nakon što su svi podaci prošli kroz model, svi podaci se ponovno propuštaju kroz model – koji sad ima promijenjene težine!. Proces ponovnog propuštanja istih podataka kroz mrežu se smatra treniranjem. Tako, kroz ovaj proces koji se iterativno ponavlja u kombinaciji s SGD-om, model je u stanju učiti na danim podacima.

## KAKO NEURALNA MREŽA UČI

U prethodnom je poglavlju objašnjeno što to znači kad umjetna neuralna mreža uči te kako izgleda proces treniranja i kako se svaki podatak, koji se koristi u procesu treniranja, prosljeđuje kroz mrežu. Prosljeđivanje podataka od ulaza do izlaza se naziva *unaprijedno prosljeđivanje*. Rezultat izlaza ovisi o težini svake pojedine veze unutar mreže.

Kada su svi podaci za treniranje iz skupa podataka za treniranje proslijedili kroz mrežu, kaže se da je završena jedna epoha (eng. *epoch*). Epoha se odnosi na jedan prolazak cijelog skupa podataka kroz mrežu tijekom procesa treniranja. Ovdje je dobro spomenuti da se, tijekom procesa treniranja, kako model uči, 'odvijaju epohe'.

### ŠTO TO ZNAČI UČITI

Kako bi se objasnio pojam učenja, dobro je prisjetiti se da, kada se model inicijalizira, težine veza su postavljene na neke proizvoljne vrijednosti te da model, na izlazu iz mreže, daje određeni izlaz za dani ulaz.

Kada se dobije izlaz, računa se gubitak (ili pogreška) za dani izlaz tako da se usporedi vrijednost koju je model predvidio s pravom vrijednošću oznake. Računanje gubitka ovisi o odabranoj funkciji gubitka.

### GRADIJENT FUNKCIJE GUBITKA

Kada je izračunat gubitak, gradijent odabrane funkcije gubitka se računa u odnosu na svaku težinu veze unutar mreže. Ovdje se pod pojmom 'gradijent' misli na derivaciju funkcije s više varijabli.

Kako bi se što jednostavnije objasnio gradijent funkcije gubitka, opisat će se gubitak u odnosu na samo jedan izlaz. Kada se izračuna gubitak tog jednog izlaza, računa se gradijent tog gubitka u odnosu na jednu odabranu težinu. Gradijent se računa tehnikom koja se zove 'povratna propagacija' (eng. *backpropagation*).

Na temelju izračunate vrijednosti gradijenta funkcije gubitka, ažurira se težina veze u modelu. Gradijent govori koji smjer pomiče gubitak prema svojoj najmanjoj vrijednosti te je cilj mijenjati vrijednost težine veze u tom smjeru u kojem će gubitak približavati svojoj minimalnoj vrijednosti.

### STOPA UČENJA

Kada se izračuna vrijednost gradijenta funkcije gubitka, ona se tada množi s vrijednošću koja se zove 'stopa učenja' (eng. *learing rate*). Stopa učenja je broj koji se obično nalazi u rasponu između 0.01 i 0.0001. Naravno, prava vrijednost stope učenja može varirati.

Stopa učenja govori koliki se 'korak' mora napraviti u smjeru minimalne vrijednosti gubitka.

### AŽURIRANJE TEŽINA VEZA

Kako bi se izračunala nova vrijednost težine veze, kao što je napisano u prethodnom poglavlju, gradijent funkcije gubitka se množi sa stopom učenja. Dobivena vrijednost se tada oduzima od vrijednosti težine veze te se dobiva nova vrijednost težine veze.

nova težina = stara težina - (stopa učenja \* gradijent)

Ovaj primjer se fokusirao na samo jednu težinu veze kako bi se objasnio koncept, ali se isti ovaj proces odnosi na svaku težinu veze u modelu svaki putu kada kroz model prolaze podaci iz skupa podataka za treniranje.

Jedina razlika je ta što će vrijednost gradijenta biti različita za svaku pojedinu težinu jer se gradijent funkcije gubitka računa u odnosu na svaku pojedinu težinu veze u neuralnoj mreži.

Ako se pretpostavi da se veze ažuriraju na kraju svake epohe, one će se inkrementalno približavati svojim optimalnim vrijednostima tako što SGD nastoji minimizirati funkciju gubitka.

### MODEL UČI

Pod pojmom da model uči se misli na to da se ažuriraju težine veza. Učiti koje se vrijednosti trebaju pripisati svakoj pojedinoj vezi na temelju toga kako te inkrementalne promjene (vrijednosti težine veza) utječu na funkciju gubitka. Kako se težine mijenjaju, model postaje sve 'pametniji' u smislu da sve ispravnije preslikava ulaze u odgovarajuće izlaze.

## GUBITAK U NEURALNOJ MREŽI

U ovom poglavlju će se objasniti što je to funkcija gubitka i kako se ona koristi u umjetnoj neuralnoj mreži.

Funkcija gubitka je ono što stohastički gradijentni spust nastoji minimizirati tako što iterativno ažurira težine veza u neuralnoj mreži. Na kraju svake epohe tijekom procesa treniranja se izračunava gubitak korištenjem razlike izlaza, odnosno onoga što je model predvidio, i pravih oznaka za pojedini ulaz.

Kao što je dan primjer u jednom od prethodnih poglavlja, neka se pretpostavi da model klasificira fotografije mački i pasa te neka je oznaka za mačku 0, a za psa 1.

* Mačka: 0
* Pas: 1

Nadalje, neka se modelu proslijedi fotografija mačke i neka model na izlazu da vrijednost 0.25. U ovom slučaju, vrijednost između toga što je model predvidio i prave vrijednosti oznake jest 0.25 – 0.00 = 0.25. Ova razlika se naziva 'pogreška' (eng. *error*).

pogreška = 0.25 – 0.00 = 0.25

Ovaj proces računanja pogreške se ponavlja za svaki izlaz. Na kraju svake epohe, pogreška akumulira svoju vrijednost za svaki pojedini izlaz.

U sljedećem poglavlju će se opisati funkcija gubitka koja se obično koristi za računanje pogreške i koja se zove 'srednja kvadratna pogreška' (eng. *mean squared error*), skraćeno MSE.

### SREDNJA KVADRATNA POGREŠKA

Kod samo jednog uzorka, za računanje srednje kvadratne pogreške, prvo se izračuna razlika (pogreška) između onoga što je model predvidio za dani ulaz i prave vrijednosti oznake za dani ulaz. Ta razlika se potom kvadrira kako bi se dobila vrijednosti srednje kvadratne pogreške za samo jednu ulaznu vrijednost.

MSE(ulaz) = (izlaz – oznaka)(izlaz – oznaka)

Kada bi se modelu proslijedilo više uzoraka odjednom, serija uzoraka (eng. *batch of samples*), tada bi se uzele srednje kvadratne pogreške na svim uzorcima.

Ovaj primjer je ilustrirao matematiku jedne funkcije pogreške, MSE. Uz ovu funkciju pogreške, još ih se nekoliko koristi u praksi. Međutim, ova ideja računanja pogreške na pojedinom uzorku se koristi i kod ostalih funkcija gubitaka. Implementacija toga što se zapravo čini sa svakom pogreškom će ovisiti o algoritmu dane funkcije gubitka. Na primjer, u ovom poglavlju se izračunao prosjek pogreške kvadrata kako bi se izračunao MSE, dok će druge funkcije gubitka koristiti neke druge algoritme kako bi se izračunala vrijednosti gubitka.

Kada bi se modelu proslijedio cijeli skup podataka za treniranje odjednom, tada bi se proces računanja gubitka događao na kraju svake epohe tijekom treniranja. Ako se skup podataka podijeli na manje skupove, serije podataka (eng. *batches*) i kada bi se ti skupovi jedan po jedan prosljeđivali modelu, gubitak bi se računao na kraju svake serije podataka.

S bilo kojom metodom, s obzirom da gubitak ovisi o težinama veza u modelu, očekuje se neka promjena vrijednosti gubitka svaki put nakon što se težine ažuriraju. Kako je cilj stohastičkog gradijentnog spusta minimizirati gubitak, očekuje se da se gubitak smanjuje sa svakom epohom.

## STOPA UČENJA U NEURALNOJ MREŽI

U ovom poglavlju će se objasniti što je to stopa učenja te će pokazati na koji se način ona koristi za treniranje neuralne mreže.

U jednom od prethodnih poglavlja je spomenuto što to znači kada neuralna mreža uči te se spomenulo da je stopa učenja broj s kojim pomnožimo gradijent funkcije gubitka.

Cilj treniranja modela jest pomoću SGD-a minimizirati gubitak između stvarne vrijednosti oznake i onoga što je model predvidio za dani uzorak za treniranje na ulazu. Put prema minimizaciji gubitka se odvija u nekoliko koraka.

Kao što je već prije spomenuto, proces treniranja započinje tako da se težinama veza dodjele neke proizvoljne vrijednosti koje se onda inkrementalno ažuriraju kako se gubitak približava svojoj minimalnoj vrijednosti.

Veličina koraka, kojom se gubitak približava svojoj minimalnoj vrijednosti, će ovisiti o stopi učenja. Konceptualno, o stopi učenja danog modela se može misliti kao o veličini koraka.

Zna se da se tijekom treniranja, nakon što se gubitak računa za svaki ulaz, gradijent gubitka izračunava u odnosu na svaku pojedinu težinu veze u danom modelu. Kada se izračuna svaki pojedini gradijent, oni će se pomnožiti sa stopom učenja.

gradijenti \* stopa učenja

Stopa učenja je, kao što je prije navedeno, broj koji se obično nalazi u rasponu od 0.01 i 0.0001. Naravno, stvarna vrijednost stope učenja može varirati i svaka će vrijednost gradijenta, nakon što se pomnoži sa stopom učenja, postati jako mala.

### AŽURIRANJE TEŽINA VEZA U MREŽI

Kada se dobije vrijednost množenja gradijenta i stope učenja, svaka pojedina vrijednost se koristi kako bi se ažurirala određena težina veze tako da se vrijednost umnoška oduzme od stare vrijednosti težine veze.

nova težina veze = stara težina veze – (gradijent \* stopa učenja)

Stara težina veze se odbacuje i umjesto nje se na vezu stavlja nova određena težina.

Vrijednost koja se dodjeljuje stopi učenja zahtjeva neko testiranje. Stopa učenja jest jedan od *hiperparametara* (eng. *hyperparameters*) koji se moraju testirati i podesiti na svakom modelu posebno kako bi se odredilo s kojom će stopom učenja model najbolje učiti. Ali, kao što je prije spomenuto, obično se vrijednost stope učenja nalazi negdje između 0.01 i 0.0001.

Kada bi se stopa učenja postavila na neku vrijednost koja je bliže vrijednosti 0.01, postoji opasnost od mogućnosti dobivanja pretjeranog rezultata (eng. *overshooting*). To se događa kada se uzme stopa, u smjeru minimuma, koja je prevelika te ona preskače minimalnu vrijednost gubitka.

Kako bi se izbjeglo prebacivanje, vrijednost stope učenja se može postaviti na broj koji je bliže vrijednosti 0.0001. Na ovaj način, kako su koraci jako mali, postizanje minimalne vrijednosti gubitka će zahtijevati puno više vremena.

Čin izbora između više i niže vrijednosti stope učenja postaje idejom kompromisa.

## SKUPOVI ZA TRENIRANJE, TESTIRANJE I VALIDACIJU

### SKUPOVI PODATAKA ZA DUBOKO UČENJE

U ovom poglavlju će se objasniti razlika između različitih skupova podataka koji se koriste kod treniranja i testiranja neuralne mreže.

Za potrebe treniranja i testiranja odabranog modela, skup podataka se dijeli na tri različita dijela. Tako su skupovi podataka sljedeći:

* skup podataka za treniranje
* skup podataka za validaciju
* skup podataka za testiranje.

### SKUP PODATAKA ZA TRENIRANJE

Skup podataka za treniranje čine podaci na kojima se model trenira. Tijekom svake epohe, model će se iznova i iznova trenirati na tom istom skupu podataka za treniranje te će nastavljati učiti o značajkama tih podataka.

Cilj je na taj način naučiti model koji će poslije moći točno predvidjeti značajke podataka koje prije nije vidio. Model će svoje pretpostavke temeljiti na onome što je naučio na skupu podataka za treniranje.

### SKUP PODATAKA ZA VALIDACIJU

Skup podataka za validaciju je skup podataka koji je odvojen od skupa podataka za treniranje, ali se koristi tijekom treniranja kako bi se validirao model. Validacijski proces pomaže kod dobivanja informacija koje poslije mogu pomoći kod podešavanja hiperparametara. Tijekom svake epohe, model će učiti na skupu podataka za treniranje, ali će u isto vrijeme biti validiran na skupu podataka za validaciju.

Tijekom procesa treniranja, model će svaki ulaz iz skupa za treniranje preslikavati u određeni izlaz, odnosno klasu predviđanja. Kada model završi klasifikaciju svakog ulaza, izračunava se gubitak preko kojeg se podešavaju težine veza u modelu. Tijekom sljedeće epohe se isti ulazi ponovno klasificiraju u izlaze.

Također tijekom treniranja, model će istovremeno klasificirati svaki ulaz iz skupa za validaciju. Model će svoju validaciju temeljiti samo na onome što je naučio na temelju skupa podataka za treniranje. Težine veza se neće ažurirati na temelju gubitka iz skupa podataka za validaciju.

Kao što je već spomenuto, skup podataka za validaciju je odvojen od skupa podataka za treniranje. Tako, kada se model validira na temelju skupa podataka za validaciju, ti podaci se ne sastoje od uzoraka koje model već poznaje iz skupa podataka za treniranje.

Jedan od glavnih razloga zašto postoji skup podataka za validaciju jest taj da se model previše ne prilagodi skupu podataka za treniranje. Ta pojava se zove *ovefitting*. Ideja *overfitting*-a je ta da model postane jako dobar u klasificiranju podataka iz skupa za treniranje, ali nije sposoban generalizirati i ispravno klasificirati podatke na kojima nije bio treniran.

Ako se tijekom treniranja u isto vrijeme provjerava model pomoću skupa za validaciju i, ako su rezultati skupa za validaciju jednako dobri kao rezultati skupa za treniranje, velika je vjerojatnost da neće doći do pojave *overfitting*-a. Ako su model vrlo dobro klasificira podatke iz skupa za treniranje, a loše klasificira podatke iz skupa za validaciju, došlo je do pojave *overfitting*-a.

Pomoću skupa podataka za validaciju se provjerava koliko dobro model može generalizirati tijekom treniranja.

### SKUP PODATAKA ZA TESTIRANJE

Skup podataka za testiranje se koristi na modelu koji je istreniran. Skup podataka za testiranje je odvojen i od skupa za treniranje i od skupa za validaciju.

Nakon što je model treniran i validiran na skupovima za treniranje i validaciju, proslijedit će mu se skup podataka za testiranje, koji nije označen, za koji će trebati predvidjeti odgovarajući izlaz.

Glavna razlika između skupa za testiranje i druga dva skupa jest da skup za testiranje ne smije imati oznake. Skupovi za treniranje i validaciju moraju biti označeni kako bi se izračunao gubitak i točnost preslikavanja svake epohe. Ovako model prolazi kroz isti proces kao i kada bi model koristio podatke iz stvarnog svijeta.

Skup za testiranje pruža konačnu provjeru je li model dobro generalizira prije nego ga se stavi u produkciju.

Cilj treniranja, validiranja i testiranja modela jest taj da model dobro klasificira podatke koje prije nije vidio. Cilj dubokog učenja jest razviti modele koji mogu dobro generalizirati.

| Skupovi podataka u dubokom učenju | | |
| --- | --- | --- |
| **Skup podataka** | **Ažuriranje veza** | **Opis** |
| Skup za treniranje | Da | Koristi se za treniranje modela. Cilj treniranja jest da model dobro  klasificira podatke, ali u isto vrijeme dobro generalizira. |
| Skup za validaciju | Ne | Koristi se tijekom treniranja kako bi se provjerilo koliko dobro  model generalizira. |
| Skup za testiranje | Ne | Koristi se kako bi se konačno provjerila modelova sposobnost  generalizacije prije nego ga se stavi u produkciju |

Glavni razlog za imati tri različita skupa podataka jest osigurati da će model moći dobro generalizirati te dobro klasificirati podatke koje prije nije vidio. Ako model ne može dobro generalizirati, najčešće je došlo do pojave *overfitting*-a ili *underfitting*-a.

## PREDVIĐANJE NEURALNE MREŽE

U prethodnom poglavlju je objašnjeno što to znači trenirati mrežu. Kada je proces treniranja završen i ako je model postigao zadovoljavajuće rezultate na skupu za treniranje i validaciju, model se provjerava na skupu podataka za testiranje.

Za razliku od skupa podataka za treniranje i validaciju koji se modelu prosljeđuju zajedno s odgovarajućim oznakama, skup za testiranje se prosljeđuje modelu bez odgovarajućih oznaka.

### PROSLJEĐIVANJE UZORAKA BEZ OZNAKA

Kod predviđanja, modelu se prosljeđuje skup podataka za testiranje bez odgovarajućih oznaka te model mora sam predvidjeti izlaz za svaki uzorak iz skupa za testiranje. Predviđanje se temelji na onome što je model naučio tijekom treniranja.

Na primjer, neka je model treniran kako bi mogao klasificirati različite pasmine na temelju fotografija. Za svaki ulazni uzorak, donosno fotografiju, model predviđa vjerojatnost za odgovarajuću pasminu.

Neka se modelu proslijedi skup za testiranje u kojem se nalaze fotografije sa psima na kojima model nije bio treniran. Kao što je već spomenuto, model nema pristup oznakama za fotografije iz skupa za testiranje. Ovaj proces testiranja će pokazati koliko dobro model radi s podacima koje prije nije vidio na temelju toga koliko se dobro predviđanja modela preklapaju s pravim oznaka za ulazne podatke.

Također, ovaj će proces dati dobar uvid u to što model nije naučio. Na primjer, neka je model treniran samo na fotografijama velikih pasmina, dok skup za testiranje sadrži fotografije s malim pasminama. Kada se modelu proslijedi jedna fotografija s malim psom, on najvjerojatnije neće dobro predvidjeti odgovarajuću pasminu jer nije bio dobro treniran na malim pasminama.

To znači da se mora osigurati da skupovi za treniranje i validaciju dobro predstavljaju prave podatke na kojima će model trebati donositi predviđanja.

## *OVERFITTING* U NEURALNOJ MREŽI

Do *overfitting*-a dolazi kada model postane jako dobar u klasificiranju ili predviđanju podataka koji su bili uključeni u skup za treniranje, ali nije jednako dobar u klasificiranju podataka na kojima nije bio treniran. Tada se kaže da se model pretjerano poklapa s podacima iz skupa za treniranje.

### KAKO UOČITI PRETJERANO POKLAPANJE

Je li došlo do pretjeranog poklapanja se može uočiti na temelju metrike za dani skup za treniranje i skup za validaciju. Tijekom treniranja, dobiju se rezultati točnosti i gubitka za skup za validaciju kao i za skup za treniranje.

Ako su rezultati skupa za validaciju primjetno gori od skupa za treniranje, to je indikacija da se model pretjerano poklapa sa skupom za treniranje. Također se može primijetiti je li došlo do pretjeranog poklapanja ako su rezultati treniranja jako dobri, dok model neispravno klasificira podatke iz skupa za testiranje.

Koncept pretjeranog poklapanja se svodi na činjenicu da model ne može dobro generalizirati podatke. Model je jako dobro naučio značajke podataka iz skupa za treniranje, ali, ako mu se daju podaci koji se neznatno razlikuju od podataka iz skupa za treniranje, model ne može dobro generalizirati i predvidjeti odgovarajući izlaz.

### SMANJENJE PRETJERANOG POKLAPANJA

#### UBACIVANJE PODATAKA U SKUP ZA TRENIRANJE

Najjednostavnija stvar koja se može napraviti, ako je moguće, jest ubaciti još podataka u skup za treniranje. Na što se više podataka trenira model, on će moći više naučiti. Također, s više podataka se dodaje veća raznolikost skupu za treniranje.

#### POVEĆANJE PODATAKA

Još jedan način pomoću kojeg se može reducirati pretjerano poklapanje jest povećanje podataka. To je proces kojim se stvaraju dodatni, izmijenjeni podaci tako što se razumno modificiraju podaci iz skupa za treniranje. Kod slikovnih podataka, podaci se mogu modificirati na sljedeće načine:

* obrezivanjem
* rotiranjem
* okretanjem
* zumiranjem

Glavna ideja povećanja podataka jest to da se u skup za treniranje ubacuju podaci koji su slični već postojećim podacima, samo što su oni razumno izmijenjeni do neke mjere tako da nisu potpuno isti.

#### REDUCIRANJE SLOŽENOSTI MODELA

Nešto što se može napraviti kako bi se smanjilo pretjerano poklapanje jest smanjiti složenost modela. Složenost se može smanjiti tako što se može ukloniti pojedini sloj iz modela ili se može smanjiti broj neurona u sloju. Na taj način model može bolje generalizirati podatke koje prije nije vidio.

#### ISPUŠTANJE

Ideja iza ispuštanja (eng. *dropout*) jest ta da se nasumično ignoriraju podskupovi neurona u sloju. Ta radnja će spriječiti ispuštene neurone da sudjeluju u predviđanju podataka.

Ova tehnika također može pomoći modelu bolje generalizirati podatke koje do tada nije vidio.

## NEDOVOLJNO POKLAPANJE U NEURALNOJ MREŽI

U ovom poglavlju će se objasniti što to znači kada se model nedovoljno poklapa (eng. *underfitting*). Također će se navesti tehnike pomoću kojih se može reducirati nedovoljno poklapanje kada do njega dođe.

Kaže se da se model nedovoljno poklapa kada nije sposoban niti klasificirati podatke na kojima je treniran, a pogotovo podatke koje do tada nije vidio. Model tada ima loše rezultate na skupu podataka za treniranje, ispravnost preklapanja je niska, a gubitak je velik. Ako model nije sposoban klasificirati podatke na kojima je bio treniran, najvjerojatnije neće dobro klasificirati podatke koje do tada nije vidio.

### REDUCIRANJE NEDOVOLJNOG POKLAPANJA

#### POVEĆANJE SLOŽENOSTI MODELA

Način na koji se može reducirati nedovoljno poklapanje jest povećati složenost modela. To je tehnika suprotna od one kojom se reducira pretjerano poklapanje. Ako su podaci u skupu na kojem treniramo model složeni, a model je relativno jednostavan, model najvjerojatnije neće biti dovoljno sofisticiran kako bi mogao ispravno klasificirati složene podatke.

Složenost modela se može povećati na sljedeće načine:

* povećanjem broja slojeva u modelu
* povećanjem broja neurona u svakom sloju
* promjenom tipa i mjesta sloja koji se koristi u modelu.

#### DODAVANJE OZNAKA ULAZNIM UZORCIMA

Jedna od tehnika kojom možemo reducirati nedovoljno poklapanje jest dodavanjem značajki ulaznim uzorcima, ako je to moguće, iz skupa za treniranje. Te dodatne značajke mogu pomoći modelu kako bi ispravnije klasificirao ulazne podatke.

Na primjer, neka model pokušava predvidjeti cijene dionice na temelju njene cijene prilikom zatvaranja u posljednja tri dana. Ulazni podaci bi se sastojali od sljedećih značajki:

* cijena na kraju prvog dana
* cijena na kraju drugog dana
* cijena na kraju trećeg dana.

Kada bi se dodale dodatne značajke ovim podacima, kao na primjer cijene dionice prilikom otvaranja burze, možda bi to pomoglo modelu da ispravnije klasificira podatke.

#### REDUCIRANJE ISPUŠTANJA

Kod korištenja ispuštanja (eng. *dropout*), može se odrediti koliki će se postotak neurona ili čvorova ispustiti iz mreže. Na primjer, ako se koristi stopa ispuštanja od 50%, i model se nedovoljno poklapa, potrebno je smanjiti postotak ispuštenih neurona tako da taj postotak bude manji od 50%.

Neuroni, koji su ispušteni, ispušteni su samo za potrebe treniranja, ali nisu ispušteni tijekom validacije modela. Tako, ako model bolje preslikava podatke iz skupa za validaciju od skupa za treniranje, tada je dobro smanjiti postotak ispuštenih neurona.

## NADZIRANO UČENJE KOD STROJNOG UČENJA

### OZNAČENI PODACI

Kod nadziranog učenja, podaci su u skupu za treniranje označeni. Oznake se koriste kako bi se nadzirao i usmjeravao proces učenja.

Kako je spomenuto u jednom od prethodnih poglavlja, podaci u skupu za treniranje i u skupu za validaciju imaju svoje oznake te su, zajedno, prosljeđeni modelu. U ovom slučaju se radi o nadziranom učenju,

Kod nadziranog učenja, svaki podatak, koji se prosljeđuje modelu tijekom treniranja, je par koji se sastoji od ulaznog objekta, ili uzorka, i odgovarajuće oznake (eng. *label*), ili izlazne vrijednosti. Ono što je bitno, kod nadziranog učenja, jest to da model uči kako preslikavati dani ulaz u odgovarajuće izlaze na temelju toga što je naučio iz označenih podataka za treniranje.

Na temelju onoga što je spomenuto u poglavlju o treniranju modela, model će preslikati dani ulaz u određeni izlaz te će onda procijeniti pogrešku za taj ulaz tako što će izračunati razliku između vrijednosti koju je previdio i prave oznake za taj ulaz.

### OZNAKE SU NUMERIČKE

Oznake, koje se pridružuju ulaznim uzorcima, enkodiraju se u nešto numeričko (0, 1, 2…)

Nakon toga se prolazi kroz proces utvrđivanja pogreške ili gubitka za sve podatke iz skupa za treniranje za svaku definiranu epohu. Cilj treniranja jest minimizirati gubitak tako da model može ispravno predviđati na podacima na kojima nije bio treniran. Model će svoje pretpostavke temeljiti na označenim podacima koje je vidio tijekom treniranja.

## POVEĆANJE PODATAKA ZA STROJNO UČENJE

U ovom poglavlju će se obraditi proces povećanja podataka te će se spomenuti u kojim slučajevima je to dobro napraviti.

Povećanje podataka se obavlja kada se žele stvoriti novi podaci tako što se izmijene postojeći podaci. Stvaraju se novi podaci tako da se razumno izmijene postojeći podaci iz skupa za treniranje.

Na primjer, fotografija se može izmijeniti tako da se:

* okrene horizontalno
* okrene vertikalno
* rotira
* poveća
* smanji
* obreže
* promijeni boja

### ZAŠTO POVEĆATI PODATKE

Jedan od razloga zašto povećati količinu podataka jest taj da se dodaje više podataka u skup za treniranje.

Još jedan razlog je taj da se smanji pretjerano poklapanje. Ako se model pretjerano poklapa s podacima za treniranje, u skup za treniranje se na taj način doda više podataka.

## PRISTRANOST U UMJETNOJ NEURALNOJ MREŽI

U literaturi, uz sam naziv *bias*, može se naići i na pojmove *bias* neuroni, *bias* čvorovi ili *bias* jedinice u neuralnim mrežama.

U ovom poglavlju će se objasniti što je to pristranost (ili *bias*) u neuralnoj mreži. Nakon toga će se objasniti kako se ona implementira te će se na kraju opisati jednostavan primjer kako bi se ilustrirao utjecaj *bias-*a u neuralnoj mreži.

Kod pristranosti, misli se na pristranost svakog neurona pojedinačno. Svaki neuron ima svoju pristranost te je cijela mreža sastavljena od više pristranosti. Vrijednosti, koje su pripisane pristranosti, se mogu naučiti, kao i težine veza. Kao što SGD povratnom propagacijom uči i ažurira težine veza, tako isto uči i ažurira vrijednosti pristranosti.

Konceptualno se o pristranosti može misliti kao o nekoj vrsti praga. To je zbog toga što će vrijednost pristranosti odrediti hoće li se izlaz iz neurona propagirati kroz mrežu. Drugim riječima, pristranost će odrediti hoće li se ili neće, i koliko, aktivirati neuron.

### GDJE SE NALAZI PRISTRANOST

Kao što je objašnjeno u jednom od prethodnih poglavlja, svaki neuron prima ponderiranu sumu ulaza od prethodnog sloja te se ta ponderirana suma prosljeđuje aktivacijskoj funkciji.

Vrijednost pristranosti neurona će se zbrojiti s ponderiranom sumom ulaza u neuron te će onda ta nova vrijednost proslijediti aktivacijskoj funkciji.

### PRIMJER GDJE SE KORISTI PRISTRANOST

Neka neuralna mreža ima dva ulazna neurona. Prvi neuron neka ima vrijednost 1, a drugi neka ima vrijednost 2. Primjer će se fokusirati na jedan neuron u prvom skrivenom sloju.

Aktivacijska funkcija koja se koristi u prvom aktivacijskom sloju će biti ReLU. Težinama veza se pridjeljuju nasumične vrijednosti. Neuron na početku neće imati pristranost.

Ponderirana suma koju neuron prima na ulaz izgleda ovako:

Ova vrijednost se zatim prosljeđuje aktivacijskoj funkciji. Kao što je već objašnjeno, izlazna vrijednost iz ReLU će biti nula za sve negativne vrijednosti, a za sve pozitivne vrijednost će ostati nepromijenjena.

U ovom slučaju, izlazna vrijednost će biti 0: ReLU(-0.35) = 0.

S obzirom da je izlazna vrijednost aktivacijske funkcije 0, neuron se smatra neaktiviranim. U ovom slučaju je prag aktivacije neurona vrijednost 0.

Neka se sad prag neurona pomakne na -1. Vrijednost pristranosti će tada biti suprotna od -1, odnosno bit će +1.

Zbroj ponderirane sume i pristranosti neurona iznosit će:

Kada se ova vrijednost proslijedi ReLU, rezultat će iznositi:

ReLU(0.65) = 0.65

Neuron se sada smatra aktiviranim te model sada ima povećanu fleksibilnost što se tiče prilagođavanja podacima jer sada posjeduje veći opseg u vezi toga koje će ga vrijednosti aktivirati, a koje ne.

Ovaj isti proces se može ponoviti i u suprotnom slučaju kako bi se suzio opseg vrijednosti koje će aktivirati neuron. Na primjer, ako se smatra da bi neuron trebao biti aktiviran ako je izlazna vrijednost aktivacijske funkcije veća ili jednaka 5, tada se pristranost neurona postavlja na vrijednost -5.

U ovom primjeru se eksplicitno postavila vrijednost pristranosti neurona, kao i vrijednosti težina. Nakon što su inicijalizirane vrijednosti pristranosti neurona nasumičnim brojevima ili nulama, vrijednosti pristranosti će se ažurirati tijekom procesa učenja dopuštajući modelu da nauči kada će aktivirati i kada neće aktivirati svaki pojedini neuron.

## VELIČINA PODSKUPOVA U UMJETNOJ NEURALNOJ MREŽI

U ovom poglavlju će se opisati što to znači specificirati veličinu podskupa iz sveukupnog skupa podataka za treniranje.

Veličinu podskupa će činiti broj uzoraka iz cjelokupnog skupa podataka za treniranje koji će se kao cjelina proslijediti mreži u jednom trenutku. Podsjetnik, epohu čini jedan prolazak cijelog skupa podataka za treniranje. Podskup i epoha nisu ista stvar.

### PODSKUPOVI U EPOHI

Neka skup podataka za treniranje čini 1000 slika automobila pomoću koji će se model naučiti da razlikuje različite modele automobila. Ako je batch veličina jednaka 10, to znači da će se skup od 10 slika automobila proslijediti modelu u jednom trenutku.

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

### RAZLOG KORIŠTENJA PODSKUPOVA

Općenito vrijedi da, što je veća veličina podskupa, model će brže završiti svaku epohu tijekom treniranja. To je zbog toga što, ovisno o računalnim resursima, računalo može procesirati toliko više od jednog uzorka u jednom trenutku.

Međutim, čak i ako računalo može obraditi velike podskupove u jednom trenutku, kvaliteta modela može degradirati jer, ako je podskup, koji se šalje modelu u jednom trenutku, dovoljno velik, model možda neće moći dobro generalizirati podatke koje do tada nije vidio.

Općenito, veličina podskupa je još jedan u nizu hiperparametara koji se moraju testirati i podesiti na temelju toga koliko ispravno konkretni model preslikava podatke tijekom učenja. Ovaj parametar se treba testirati i u odnosu na to koliko dobru izvedbu ima samo računalo u smislu iskorištavanja vlastitih resursa kada se koriste različite veličine podskupova.

Na primjer, kada bi veličina podskupa bio relativno veliki broj, kao 100, tada računalo možda neće raspolagati dovoljnom računalnom moći da paralelno obradi svih 100 slika i to bi bio znak da se veličina skupa mora postaviti na neki manji broj.

### MINI-BATCH GRADIJENTNI SPUST

Korištenjem *mini-batch* gradijentnog spusta, gradijent će promijeniti svoju vrijednost nakon prolaska svakog podskupa (per-batch basis) kroz model. Ovo je suprotno od stohastičkog gradijentnog spusta koji implementira promjene gradijenta nakon prolaska svakog uzorka i suprotno je od *batch* gradijentnog spusta koji implementira promjenu gradijenta nakon svake epohe.

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

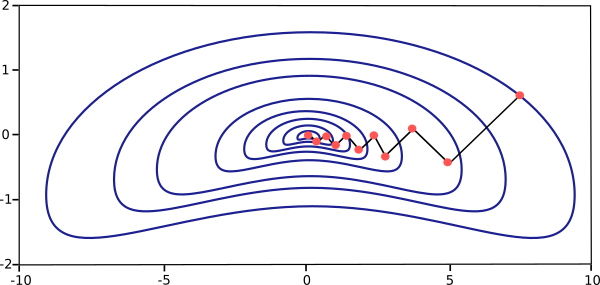
Postupak koji se koristi za provođenje procesa učenja naziva se optimizacijski algoritam (eng. *optimization algorithm, optimizer*).

Postoji nekoliko različitih algoritama za optimizaciju. Svaki od njih ima različite karakteristike i performanse što se tiče memorijskih zahtjeva, brzine izvođenja i numeričke preciznosti.

Postoji nekoliko optimizacijskih algoritama koji se koriste za učenje neuronskih mreža:

1. gradijentni spust
2. Newton-ova metoda
3. Konjugirani gradijent
4. Kvazi-Newton-ova metoda
5. Levenberg-Marquardt algoritam

Gradijentni spust je najjednostavniji algoritam za učenje. Međutim, najveći nedostatak mu je što zahtijeva mnogo iteracija za funkcije koje imaju duge i uske strukture doline.

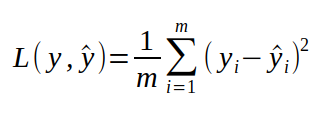


Korištenje gradijentnog spusta se preporuča kod jako velikih neuronskih mreža s više tisuća parametara.

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

FUNKCIJA GUBITKA

Funkcija gubitka je funkcija koja govori koliko dobro određena neuronska mreža obavlja svoju funkciju za određeni zadatak. Intuitivan način kako se ona računa je da se za svaki ulaz dobije neki izlaz. Taj izlaz se potom oduzme od vrijednosti željenom izlaza i dobiveni rezultat se kvadrira (jer su negativni brojevi jednako loši kao i pozitivni).



Y označava željeni broj, y s kapicom označava dobiveni broj, i označava redni broj ulaza, a m označava ukupan broj ulaza iz skupa.

Ako je vrijednost gubitka velika, to znači da mreža nema dobro izvođenje.

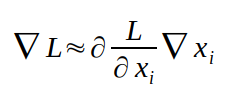
TRENIRANJE

Kada se tek krene s radom neuronske mreže, težine veza se inicijaliziraju nasumičnim brojevima. Očito je da rezultati neće biti zadovoljavajući. Tijekom procesa treniranja, cilj je započeti s neuronskom mrežom koja ima loše izvođenje i završiti s mrežom koja ima visoku točnost preslikavanja. Odnosno u terminima funkcije gubitka, cilj je do kraja procesa treniranja postići što manju vrijednost funkcije gubitka.

Problem treniranja mreže ekvivalentan je problemu minimiziranja gubitka. Uz to, jednostavnije je implementirati minimizaciju, negoli optimizaciju.

Postoji mnogo algoritama koji optimiziraju funkcije. Ti algoritmi se mogu temeljiti na gradijentu u smislu da ne koriste samo informacije koje pruža funkcija sama po sebi već i informacije koje daje njen gradijent. Jedan od najjednostavnijih algoritama koji se temelje na gradijentu se zove stohastički gradijentni spust.

Prvo, dobro je prisjetiti se što čini derivaciju funkcije u odnosu na neku varijablu. Neka funkcija bude f(x) = x. Ovdje je derivacija za svaku vrijednost broja x jednaka 1. Derivacija je stopa koliko se brzo funkcija mijenja kada se pomakne beskonačno malim korakom u pozitivnom smjeru. Matematički se to može napisati kao:



Ovo znači: koliko se određena funkcija (lijevi izraz) približno mijenja, to je jednako derivaciji te funkcije u odnosu na neku varijablu x pomnoženoj s tim koliko se promijenila ta varijabla.

Kod primjera f(x)=x, gdje je derivacija 1, to znači da kada bi se uzeo neki *epsilon* korak u pozitivnom smjeru, rezultat funkcije će se promijeniti za 1 pomnožen s tom stopom *epsilon*. U ovom slučaju, to bi bio epsilon.

Gradijent je vektor parcijalnih derivacija. Njegove elemente sačinjavaju derivacije u odnosu na neku varijablu o kojoj funkcija ovisi. Kada bi se kao primjer uzela jednostavna funkcija kao f(x)=x, njen vektor bi sadržavao samo jedan element jer ova funkcija prima samo jedan element. Kod složenijih funkcija, gradijent će sadržavati derivacije u odnosu na bilo koju željenu varijablu.

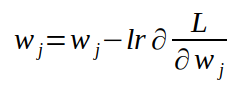
Neka se sada koristi funkcija f(x)=x2. Njena derivacija je 2x. Minimum ove funkcije se nalazi u x=0. Međutim, to računalo ne zna, mora to izračunati. Računalo uzima neku nasumičnu vrijednost x=2. Derivacija funkcije u kojoj je x=2 ima vrijednost 4. To znači da, kada bi se učinio korak u pozitivnom smjeru, funkcija će se promijeniti proporcionalno 4. Znači da će se povećati. S obzirom da se teži minimizaciji funkcije, potrebno je učiniti korak u suprotnom, negativnom, smjeru kako bi se funkcija smanjila. Međutim, računalo ne znam koliki se korak mora napraviti u negativnom smjeru jer derivacija samo garantira da će se funkcija smanjiti ako se učini beskonačno mali korak. Taj korak se naziva stopa učenja i jedan je od hiper-parametara. S druge strane, ako se x postavi na -2, derivacija će biti jednaka -4. To znači da, ako se uzme beskonačno mali korak u pozitivnom smjeru, funkcija će se proporcionalno promijeniti za -4, odnosno smanjit će se i to je željeni scenarij.

Ukratko, kada je x>0 i derivacija je veća od 0, potrebno je ići u negativnom smjeru. Kada je x<0 i derivacija manja od nula, potrebno je ići u pozitivnom smjeru.

Gradijent je vektor koji pokazuje u nekom smjeru u prostoru. Pokazuje u smjeru najstrmijeg porasta funkcije. S obzirom da je cilj minimizirati funkciju, potrebno je učiniti korak u suprotnom smjeru od gradijenta.

U neuralnoj mreži se o vrijednostima x misli kao o ulaznim vrijednostima, dok su izlazi y neki fiksni brojevi. Varijable u odnosu na koje će se računati derivacije su težine w s obzirom da su to vrijednosti koje se trebaju promijeniti kako bi se poboljšalo izvođenje mreže.

Ako se izračuna gradijent funkcije gubitka u odnosu na težine i uzme mali korak u smjeru suprotnom od onog koji pokazuje gradijent, gubitak će se smanjivati sve dok ne konvergira u neki lokalni minimum. Ovaj algoritam se naziva gradijentni spust. Pravilo mijenjanja težina nakon svake iteracije gradijentnog spusta se matematički izražava formulom:



Lr označava stopu učenja. Nema pravila koliko se velika stopa učenja treba uzeti nakon svake iteracije. To je jedan od hiper-parametara. Međutim, ako se uzme prevelika stopa učenja, postoji mogućnost da će rezultat u jednom trenutku 'preskočiti' minimum. Tada se kaže da će algoritam divergirati. Ako se odabere premaleni korak, previše će vremena biti potrebno kako bi se postigla konvergencija u nekom lokalnom minimumu.

U literaturi se češće nailazi na pojam stohastičkog gradijentnog spusta. Pod tim se obično misli na gradijentni spust neke mini-grupe. Sama veličina grupe za SGD je proizvoljna.

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

Za razliku od konvencionalnih tehnika obrade podataka gdje je postupak obrade potrebno analitički razložiti na određeni broj algoritamskih koraka, kod ovog tipa neuronskih mreža takav algoritam ne postoji. Znanje o obradi podataka, tj. znanje o izlazu kao funkciji ulaza, pohranjeno je implicitno u težinama veza između neurona. Te se težine postupno prilagođavaju kroz postupak učenja neuronske mreže sve do trenutka kada je izlaz iz mreže, provjeren na skupu podataka za testiranje, zadovoljavajući. Pod postupkom učenja kod neuronskih mreža podrazumijevamo iterativan postupak predočavanja ulaznih primjera (uzoraka, iskustva) i eventualno očekivana izlaza.

Ovisno o tome da li nam je u postupku učenja á priori znan izlaz iz mreže, pa ga pri učenju mreže koristimo uz svaki ulazni primjer, ili nam je točan izlaz nepoznat, razlikujemo dva načina učenja:

• učenje s učiteljem (engl. supervised learning) – učenje mreže provodi se primjerima u obliku para (ulaz, izlaz),

• učenje bez učitelja (engl. unsupervised learning) – mreža uči bez poznavanja izlaza.

Skup primjera za učenje često se dijeli na tri odvojena skupa: skup za učenje, skup za testiranje i skup za provjeru (validaciju). Primjeri iz prvog skupa služe za učenje u užem smislu (podešavanje težinskih faktora). Pomoću primjera iz drugog skupa vrši se tijekom učenja provjera rada mreže s trenutnim težinskim faktorima kako bi se postupak učenja zaustavio u trenutku degradacije performanse mreže. Umjetnu neuronsku mrežu moguće je, naime, pretrenirati - nakon određenog broja iteracija mreža gubi svojstvo generalizacije i postaje stručnjak za obradu podatka iz skupa primjera za učenje dok preostale podatke obrađuje loše. Stalnim praćenjem izlaza iz mreže dobivenog pomoću primjera iz skupa za testiranje moguće je otkriti iteraciju u kojoj dobiveni izlaz najmanje odstupa od željenog (slika 2.3). Točnost i preciznost obrade podataka moguće je naposlijetku provjeriti nad trećim skupom primjera – skupom za provjeru.

Uz pojam učenja umjetne neuronske mreže vezani su pojmovi iteracije i epohe. Pod iteracijom podrazumijevamo korak u algoritmu postupka za učenje u kojem se odvija podešavanje težinskih faktora, dok je epoha jedno predstavljanje cjelokupnog skupa za učenje. Ovisno o broju primjera predočenih mreži za trajanje jedne iteracije, razlikujemo:

* pojedinačno učenje (engl. on-line training)– u jednoj iteraciji predočavamo samo jedan primjer za učenje (tj. kod svakog primjera za učenje vrši se prilagodba težinskih faktora),
* grupno učenje (engl. batch training) –u jednoj iteraciji predočavamo sve primjere za učenje (tj. iteracije se podudaraju s epohama).
* Algoritam BACKPROPAGATION dan je u tablici 5.1. Prikazana je stohastička verzija algoritma za pojedinačno (engl. on-line) učenje. Korištena je slijedeća notacija: xij je ulaz s jedinice i u jedinicu j (izlaz jedinice i), ωij je odgovarajuća težina, δn je pogreška izlaza jedinice n. Veličine su skicirane na slici 5.6. Algoritam kao parametre uzima skup za učenje D, stopu učenja η, broj čvorova ulaznog sloja ni, broj čvorova izlaznog sloja no i broj čvorova skrivenog sloja nh. Mreži se predočavaju primjeri za učenje u obliku para (x, t) gdje je x vektor ulaznih vrijednosti a t vektor ciljnih izlaznih vrijednosti.
* Algoritam nakon inicijalnog postavljanja težina u glavnoj petlji ponavlja predstavljanje sviju primjera mreži sve dok nije ispunen uvijet zaustavljanja. Kao uvjet može poslužiti maksimalni dozvoljeni iznos pogreške dobivene obradom primjera iz skupa za učenje ili skupa za testiranje, zatim postupak se može zaustaviti nakon fiksnog broja iteracija ili epoha i sl. Uvjet zaustavljanja ključan je parametar jer premalo iteracija rezultira lošom obradbenom sposobnosti mreže dok preveliki broj iteracija dovodi do njezina pretreniranja.
* Za svaki predstavljeni primjer računa se izlaz iz mreže na način da se signali proslijeđuju od ulaznih čvorova ka izlaznima te računa izlaz svakog pojedinog čvora. U ovoj fazi algoritma signali propagiraju unaprijed, od ulaznog sloja ka izlaznom. Na osnovi odstupanja stvarnog izlaza od ciljnog, računa se pogreška i ugađaju svi težinski faktori u cilju njezine minimizacije.

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

Neuronska mreža tijekom procesa učenja uči preslikavati ulaze u izlaze na danom skupu vrijednosti za treniranje.

Treniranje uključuje proces pronalaženja skupa težina u mreži koje dobro ili dovoljno dobro rješavaju specifični problem.

Proces treniranja je iterativni proces. To znači da napreduje korak po korak donošenjem malih izmjena na težinama modela tijekom svake iteracije i tako donosi promjenu i u izvođenju modela.

Proces treniranja rješava optimizacijski problem pronalaženja težina koje će rezultirati izlazom iz modela s najmanjim gubitkom.

*Optimizacija je općenito jako težak zadatak […] Kod treniranja neuralnih mreža, potrebno je suočavati se s generalnim ne-konveksnim slučajevima.*

— strana 282, [Deep Learning](https://amzn.to/2rjgvLI), 2016.

Optimizacijski proces se konceptualno može razumjeti kao traženje rješenja, koje se nalazi u krajoliku svih mogućih rješenja, koje će zadovoljiti dane kriterije.

Točku na krajoliku čini specifični skup težina u modelu. Povišenje te točke je procjena određenog skupa težina. Udubljenja predstavljaju dobre modele s malim vrijednostima gubitaka.

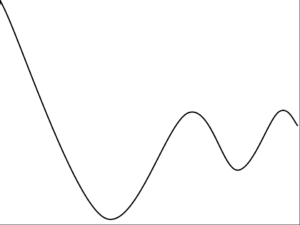
Krajolik se još može nazivati 'površinom pogreški' (eng. *error surface*).

Međutim, funkcija gubitka za težine je višedimenzionalna funkcija koju je nemoguće vizualizirati. Ali, kada bi se mogla vizualizirati, izgledala bi poput krajolika s planinama i dolinama.

Optimizacijski algoritam iterativno korača tim krajolikom, mijenja težine i traži dobra područja, odnosno spuštena područja, doline.

Kod jednostavnog optimizacijskog problema, krajolik može sličiti velikoj posudi i tako bi pronalaženje dna bilo jednostavno. Takvi optimizacijski procesi se matematički opisuju kao konveksni.

Površina pogreški kroz koju mora proći optimizacijski algoritam neuronskih mreža sadrži mnogo brežuljaka i dolina. Takav optimizacijski problem se matematički opisuje kao ne-konveksni.



Odnosno, ne postoji algoritam koji bi mogao riješiti problem pronalaženja optimalnog skupa težina za neuronsku mrežu u polinomskom vremenu.

Matematički, optimizacijski problem koji se rješava kod treniranja neuronskih mreža se naziva NP-kompletnim (jako ih se teško rješava).