Duboko učenje: pregled područja

Fumić, Patrik

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Patrik Fumić

DUBOKO UČENJE: PREGLED PODRUČJA

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Patrik Fumić

Matični broj: 44022/15–R

Studij: Informacijsko i programsko inženjerstvo

DUBOKO UČENJE: PREGLED PODRUČJA

DIPLOMSKI RAD

Mentorica:

Prof. dr. sc. Jasminka Dobša

Varaždin, Kolovoz 2021.

Patrik Fumić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj rad obrađuje područje dubokog učenja na način da prije svega objašnjava što je to duboko učenje i u kojem je odnosu na njegove nadskupove koji su strojno učenje, a zatim i sama umjetna inteligencija. Također se objašnjava potreba za dubokim učenjem u djelovanju čovjeka kao i primjena dubokog učenja u određenim područjima. Nakon toga se obrađuju neuronske mreže i razni algoritmi koji se primjenjuju kod istih, uz prikazivanje detaljnijih primjera navedenih algoritama. Prethodno navedeni sadržaj je dodatno razjašnjen putem konkretnog primjera instalacije razvojnog okruženja za duboko učenje kao i konkretnim primjerom dubokog učenja u programskom jeziku Python.

Ključne riječi: Umjetna inteligencija, strojno učenje, duboko učenje, neuronska mreža, neuroni, modeli, algoritmi, strukture, podatci

Sadržaj

1. Uvod

U današnje vrijeme informacijske tehnologije imaju veću ulogu nego ikada. Informacijske tehnologije su prisutne u svakom segmentu ljudske djelatnosti i neprestano raste njihov utjecaj. Napretkom čovječanstva, kao i već navedenom, sve izraženijom integracijom informacijskih tehnologija, se dolazi do sve većih izazova i za rješavanje danih izazova pojavljuje se potreba za novim načinom dolazaka do rješenja na znatno brži i efektivniji način. Umjetna inteligencija se ovdje nameće kao novo poglavlje napretka čovječanstva. Radi se sinergiji znanstvenih i inženjerskih pothvata koji zajedno čine strojeve inteligentnima, odnosno sposobnima da percipiraju svoju okolinu iz dane okoline donose konkretne zaključke (Mccarthy, 2004). Tema ovog rada je duboko učenje, područje koje je podskup same umjetne inteligencije i koje pronalazi svoju namjenu u širokom spektru. Radi se o dijelu umjetne inteligencije koji ima najviše uspjeha s nestrukturiranim podacima, što će biti detaljno objašnjeno u nastavku ovog rada.

Cilj ovog rada je dati osnovni uvid u temu dubokog učenja i pokazati prednosti dubokog učenja u razvoju informacijskih sustava i čovječanstva općenito. Rad će detaljno objasniti osnove dubokog učenja na način da će se prvo proći teoretski dio, gdje će se detaljnije obraditi pitanje što je to točno duboko učenje, gdje se pozicionira područje dubokog učenja u odnosu na sam koncept umjetne inteligencije i strojnog učenja, koje su primjene dubokog učenja i na koji način duboko učenje funkcionira. Nakon navedenog teoretskog dijela će na praktičnom primjeru u programskom jeziku Python biti prikazana jedna konkretna implementacija dubokog učenja gdje će se ujedno detaljno objasniti sam način kreiranja i funkcioniranja jednog takvog sustava.

Osobno sam pronašao interes u ovoj temi iz razloga što sam uvjeren u to da se budućnost temelji na razvoju umjetne inteligencije. Umjetna inteligencija svakim danom ima sve veći i veći utjecaj i duboko učenje kao dio same umjetne inteligencije se pokazalo kao jedno od izvanrednih područja koje omogućava strojevima razumijevanje velikog skupa nestrukturiranih podataka. Strojevi dobivaju sposobnost razumijevanja načina na koji ljudi razmišljaju što na kraju rezultira beskonačnim mogućnostima primjene dane tehnologije u stvarnom svijetu.

2. Korištene metode i alati

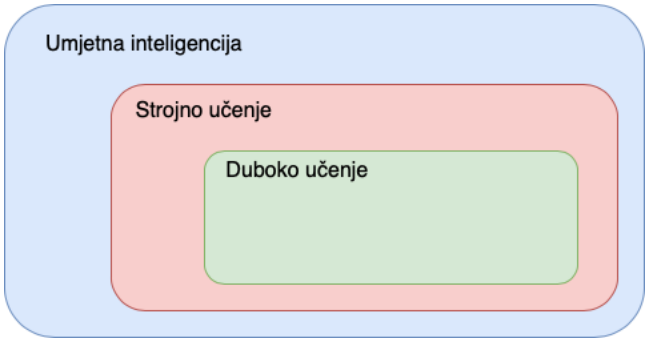
Tema je ovoga rada duboko učenje kao područje čiji se začeci naziru 1943. godine gdje su McCulloch i Pitts prvi puta predstavili ideju neuronskih mreža, ideju na kojoj se temelji cijelo područje dubokog učenja (Heaton, 2015). U početku samo ideja koja će tek kasnije doživjeti svoj pun potencijal.

U sklopu teoretskog dijela ovog rada će uz pomoć metode analize biti raščlanjeno duboko učenje na više segmenata koji sačinjavaju samo duboko učenje. Navedeni segmenti će biti detaljno razrađeni u zasebnim cjelinama i naknadno će biti prikazano kako grade potpuno duboko učenje. S druge strane, u drugom dijelu ovoga rada će biti prikazan praktična primjena dubokog učenja. Uz pomoć programskog jezika Python i Googleove platforme pod nazivom Google Colab bit će prikazano korištenje dubokog učenja kako bi se kreirao sustav koji je sposoban automatski generirati naslove pojedinih slika.

Python je interpreterski programski jezik visoke razine. Njegova glavna prednost je način na koji je dizajniran a to je da nastoji učiniti kod što čitljivijim i lakšim za razumijevanje. Također se radi o jeziku koji je objektno-orijentiran i kao takav omogućuje programerima pisanje jasnog, logičkog koda za projekte svih veličina. Druga važna komponenta u praktičnom primjeru ovog rada je Google Colab. Google Colab je platforma razvijena od strane Google Research odjela, a namijenjena je za pokretanje proizvoljnog python koda putem web preglednika. Iako Python kod može biti bilo kakve namjene, ova platforma je naročito pogodna i najčešće se koristi za strojno učenje, analizu podataka i edukaciju. Colab je zapravo platforma koja pokreće Jupyter notebook i pruža razvojnim programerima besplatan ili napredan pristup računalnim resursima uključujući grafičke procesorske jedinice.

3. Strojno učenje

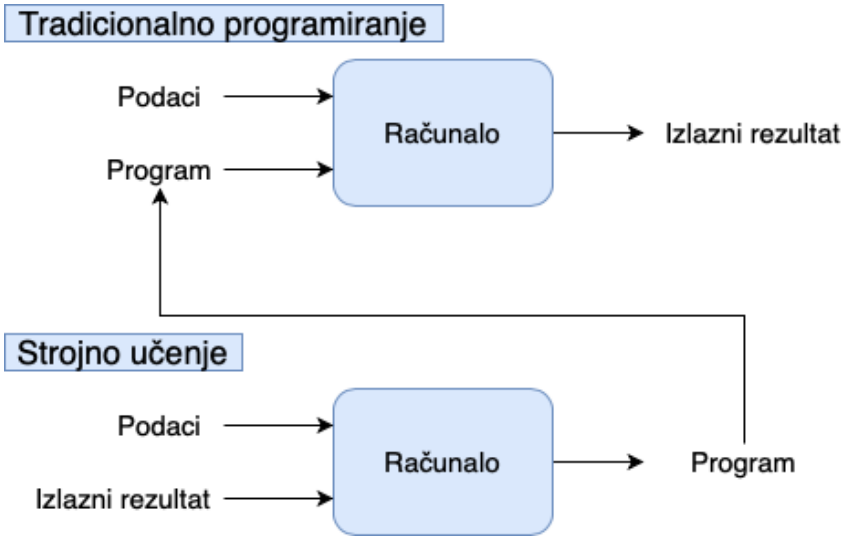
Strojno je učenje (*eng. Machine learning)* područje umjetne inteligencije i računalnih znanosti koje se bazira na korištenju podataka i algoritama kako bi se imitiralo učenje čovjeka (Alpaydin, 2016). Krajnji je cilj strojnog učenja postepeno poboljšavati preciznost i pouzdanost dobivanja rezultata. Duboko učenje predstavlja dio strojnog učenja i iz tog razloga će se u ovom radu proći kroz osnove strojnog učenja kako bi se lakše razumjeli određeni koncepti kasnije u samom dubokom učenju.



Slika 1. Hijerarhija umjetne inteligencije, strojnog učenja i dubokog učenja (Gavrilova, 2020)

Strojno učenje nastoji omogućiti način računalima da sama dođu do načina kako da izvode određene radnje, odnosno kako da dolaze do zaključaka, bez da su ta ista računala prethodno bila eksplicitno programirana za taj zadatak. Ovaj zadatak se nastoji izvršiti na način da se računala sama uče iz skupa podataka i kao rezultat naučenog, ista budu sposobna izvršavati određene radnje. Potreba za ovim načinom rješavanja problema se javila u trenutku kada su ljudi shvatili da je, pri pokušaju rješavanja određenih kompleksnih problema, bilo efektivnije napraviti sustav koji je sposoban sam stvoriti algoritam, bez da se on unaprijed definira, koji će se kasnije koristiti za rješavanje prethodno navedenog problema (Alpaydin, 2020).

Za lakše razumijevanje istaknut ćemo razliku između tradicionalnog programiranja i strojnog učenja. Kod tradicionalnog programiranja razvojni programer piše kod koji je spreman primiti određeni skup podataka. Isti taj kod naknadno obrađuje ulazni skup podataka i daje očekivani izlazni rezultat. S druge strane strojno učenje funkcionira na način da se računalu daju primjeri rezultata, odnosno onoga što želimo da računalo radi. To su najčešće podaci s određenim oznakama ili kategorizacija različitih klasa podataka. Na temelju prethodno navedenih ulaznih podataka strojno učenje ne vraća konkretan rezultat, već vraća algoritam, odnosno program, koji se može koristiti za dobivanje novih zaključaka odnosno rješavanje novih problema (Grimson, 2017).



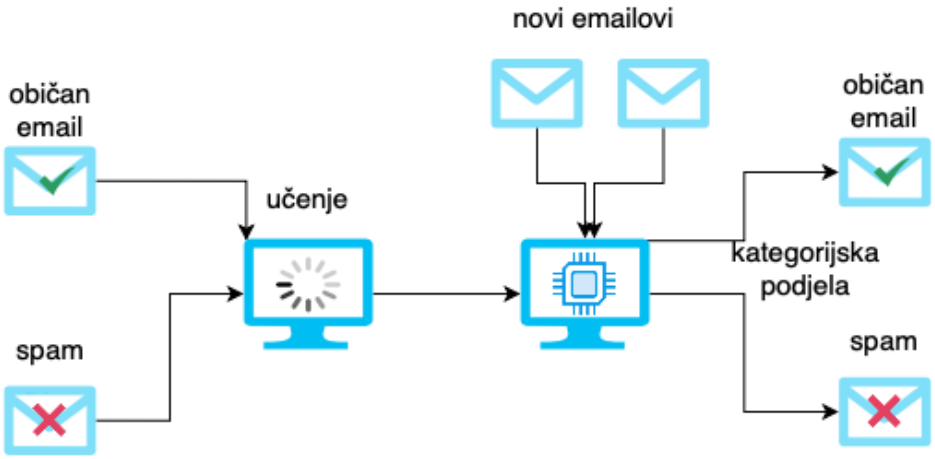
Slika 2. Tradicionalno programiranje u usporedbi sa strojnim učenjem (ulazi i izlazi) (Grimson, 2016)

3.1. Metode strojnog učenja

Strojno učenje se može izvršavati u više metoda koje su širokog opsega. Metode se klasificiraju u tri kategorije koje ovise o tome koji su ulazni signali kod učenja, odnosno koje su povratne informacije na sustav koji uči (Russell & Norvig, 2021). Detaljnije će biti razjašnjeno u nastavku.

3.1.1. Nadzirano učenje

Kod nadziranog učenja *(eng. supervised learning)* sustav dobiva parove ulaznih podataka i pripadajućih rezultata, nazovimo ih (x,y), iz kojih mora naučiti funkciju koja je sposobna pridružiti izlaznu vrijednost za određeni ulazni skup podataka odgovarajućem rezultatu, odnosno potrebno je pronaći preslikavanje y = f(x). Važno je naglasiti da je ulazni skup podataka označen na način da svaki podatak pripada određenoj kategoriji. Konkretan primjer ulaznog skupa podataka bi bile slike životinja od kojih su samo dostupne slike psa ili mačke. Uz svaku sliku dolazi i pripadajuća oznaka *(eng. label)*, odnosno dodatna informacija koja govori sustavu nalazi li se na slici pas ili mačka. Sustav nakon faze učenja mora biti sposoban sam kategorizirati životinju koja mu se postavlja kao ulazni skup podataka i odrediti radi li se o mački ili psu na navedenoj slici, odnosno odrediti pripadajuću oznaku životinje na slici (Russell & Norvig, 2021). U nastavku je prikazan drugi primjer u obliku slike gdje se radi o prepoznavanju spam poruka. Ulazni skup podataka predstavljaju email poruke od kojih su neke označene kao spam. Nakon perioda učenja očekujemo da će sustav biti u stanju prepoznati spam poruke u novom skupu ulaznih podataka.



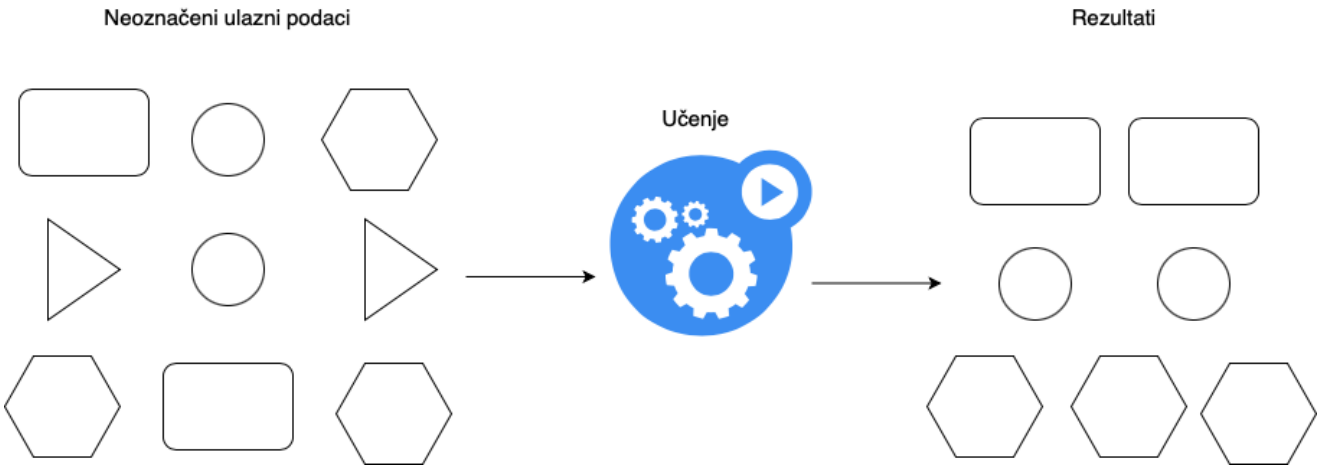
Slika 3. Primjer nadziranog učenja u prepoznavanju spam poruka (Pant, 2019)

Nadzirano učenje se može podijeliti na dva tipa problema kod rudarenja podataka, a to su klasifikacija i regresija. Klasifikacija se koristi kako bi se prepoznali određeni entiteti unutar skupa ulaznih podataka i kako bi se donio konačan zaključak kojoj oznaci pripada dani entitet odnosno kako bi se on definirao. S obzirom na prethodno navedeno preslikavanje y = f(x), kod klasifikacije y predstavlja pojedinu konkretnu kategoriju. Regresija s druge strane se koristi kako bi se razumjele veze između zavisnih i nezavisnih varijabli, odnosno u formuli preslikavanja y predstavlja skalabilnu vrijednost (Russell & Norvig, 2021).

3.1.2. Nenadzirano učenje

Za razliku od nadziranog učenja gdje imamo konkretne oznake koje su unaprijed definirane i kojima je potrebno pridružiti ulazni skup podataka, kod nenadziranog učenja *(eng. unsupervised learning)* sustav prima ulazni skup neoznačenih podataka. Prethodno navedene podatke sustav pokušava analizirati i grupirati na način da se otkrivaju određeni uzorci ili grupacije podataka, odnosno nekakve zajedničke karakteristike, bez uključivanja čovjeka u navedeni postupak(Russell & Norvig, 2021). S obzirom da nenadzirano učenje ima iznimno dobru sposobnost u otkrivanju sličnosti i razlika u neoznačenim podacima, ono se nameće kao odlično rješenje u područjima kao što su segmentacija korisnika, prepoznavanje slika i eksplorativna analiza podataka (IBM Cloud Education, 2020).

U nastavku je prikazan primjer nenadziranog učenja gdje možemo jasno vidjeti ulazni skup podataka koji nema pripadajuće oznake. Sam sustav nenadziranog učenja je zadužen za prepoznavanje sličnosti i razlika i generiranje pripadajućih kategorija u koje će svrstavati prisutne entitete.



Slika 4. Primjer nenadziranog učenja (Ravish, 2021)

3.1.3. Podržano učenje

Podržano učenje *(eng. reinforcement learning)* je područje strojnog učenja koje temelji učenje sustava na izvršavanju radnji unutar određenog okruženja na način da se maksimizira povratna kumulativna nagrada za izvršavanje prethodno navedenih radnji. Ovo znači da sustav pokušava izvršiti određene radnje koje su mu dostupne u okruženju u kojem se nalazi i za svaki pokušaj sustav dobiva niz povratnih signala koji mu ukazuju koliko je ispravno, odnosno neispravno njegovo izvršavanje. Sustav konstantno nastoji maksimizirati nagradu i minimizirati negativnu povratnu informaciju i na taj način se prilagođava i uči (Russell & Norvig, 2021).

Slika 5. Primjer podržanog učenja (Ravish, 2021)

3.2. Proces strojnog učenja u koracima

Kako bi kreirali sustav koji je spreman učiti i naposljetku pružati određene rezultate potrebno je kreirati takozvani model. Model je završni rezultat algoritma strojnog učenja koji je radio na danom skupu ulaznih podataka. Model predstavlja ono što je algoritam strojnog učenja naučio. S obzirom da su algoritam strojnog učenja i sam model međusobno povezani možemo zaključiti da je algoritam strojnog učenja procedura koja se pokreće na skupu ulaznih podataka koja rezultira modelom. Jednostavan prikaz navedenog u obliku formule je sljedeći, Model = Algoritam(Podatci) (Brownlee, 2016).

Proces strojnog učenja se sastoji od 7 koraka koji će biti objašnjeni u nastavku. Podjela koja se navodi u nastavku se u cijelosti temelji na članku dr. sc. Raul V. Rodrigueza koji se navodi u literaturi (V. Rodriguez, 2020).

3.2.1. Prikupljanje podataka

Prikupljanje podataka je prvi korak u procesu strojnog učenja. Ovaj korak je iznimno važan zato što kvaliteta krajnjeg model predviđanja *(eng. predictive model)* ovisi o kvaliteti i količini podataka koji se pružaju sustavu. Za manje projekte se često koristi proračunska tablica koja se kasnije može ispisati u obliku CSV datoteke *(eng. Comma separated values)*, dok se kod većih projekata koriste punokrvne baze podataka. Način prikupljanja podataka se uvelike razlikuje ovisno o prirodi projekta.

3.2.2. Pripremanje podataka

Nakon što su podatci prikupljeni potrebno je odraditi određene postupke pripreme istih kako bi minimizirali učinak pojedinih faktora na krajnji ishod. Jedan od koraka u pripremanju podataka je mijenjanje redoslijeda samih podataka kako bi on bio nasumičan. Ne želimo nikakav predodređeni redoslijed podataka koji bi mogao uzrokovati nekakve nepoželjne zaključke. Također ovo je korak u kojemu se treba odlučiti koji će se atributi koristiti u kreiranju modela. Odabrani atributi će direktno utjecati na vrijeme izvršavanja i same rezultate. Također je poželjno utvrditi veze između različitih atributa koje možemo iskoristiti kako bi unaprijedili sam postupak generiranja modela.

Uz veze je potrebno utvrditi postoji li kakva neuravnoteženost u podacima. Ovo znači da moramo osigurati određenu ravnotežu količine podataka za svaki od pojedinih željenih rezultata. Ukoliko imamo preveliku reprezentaciju podataka koji odgovaraju jednom tipu rezultata razvit će se svojevrsna naklonost unutar modela koja će naginjati onom rezultatu koji je značajnije zastupljen, što nikako ne želimo.

Iduće što je potrebno napraviti je razdvojiti podatke u dva dijela. Jedan dio podataka koji ćemo koristiti za treniranje našeg modela i to će ujedno biti većina podataka početnog skupa podataka, a drugi dio podataka će se koristiti za evaluaciju našeg modela i njegovih performansi. Okvirni omjer podataka za trening u odnosu na evaluaciju je 80 naprema 20, respektivno.

U ovom koraku je, ovisno o samom skupu podataka, ponekada potrebno napraviti dodatne oblike manipulacije i prilagođavanja samih podataka. Ovdje se misli na normalizaciju, ispravljanje grešaka, uklanjanje duplikata i slično.

3.2.3.Odabir modela

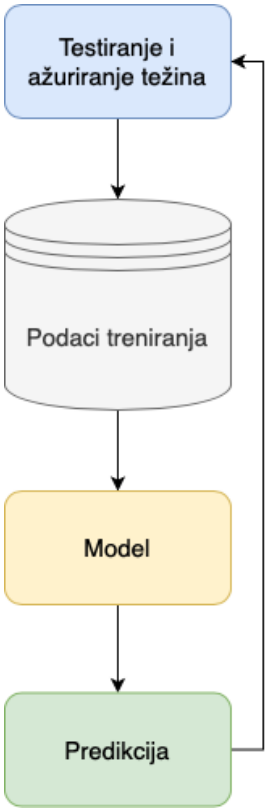
Postoji mnoštvo modela koji su kreirani tijekom godina. Određeni modeli odgovaraju određenim zahtjevima. U tablici u nastavku će biti prikazano nekoliko modela i njihova opća primjena.

Tablica 1. Modeli strojnog učenja i primjene

Model	Primjena
Logistička regresija *(eng. logistic regression)*	Predviđanje cijena
Potpuno spojene neuronske mreže *(eng. fully connected neural networks)*	Klasifikacija
Konvolucijske neuronske mreže *(eng. convolutional neural networks)*	Procesiranje slika
Ponavljajuće neuronske mreže *(eng. recurrent neural networks)*	Prepoznavanje glasa
Nasumična šuma *(eng. random forest)*	Otkrivanje prijevara
Učenje podržavanjem *(eng. reinforced learning)*	Učenje metodom pokušaja pogrešaka
Generativni modeli *(eng. generative models)*	Kreiranje slika
K - srednje vrijednosti *(eng. K-mean values)*	Segmentacija
K - najbližih susjeda *(eng. K-closest neighbours)*	Sustavi preporuke
Bayesovi klasifikatori *(eng. Bayes classification)*	Filtriranje šuma u podacima i spama

3.2.4.Treniranje modela

Korak treniranja modela se ujedno smatra glavnim dijelom strojnog učenja. Ovo je korak u kojemu se koristi prethodno navedeni skup podataka kako bi se inkrementalno poboljšavala mogućnost modela u njegovom predviđanju rezultata. Prije početka samog treniranja potrebno je inicijalizirati takozvane težine. Težinama je potrebno dodijeliti nasumične vrijednosti, a one se koriste kao vrijednosti koje utječu na veze između ulaza i izlaza. Algoritam koji radi na treniranju modela ih samostalno podešava u svakom prolazu treniranja. Iterativni dijagram je prikazan u nastavku.



Slika 6. Dijagram treniranja modela

3.2.5.Procjena

Nakon završetka treniranja modela moramo se uvjeriti daje li model zadovoljavajuće rezultate ili ne. U ovom koraku koristimo procjenu. U drugom koraku je navedeno kako je potrebno podijeliti ulazni skup podataka, a upravo u ovom koraku koristimo drugi dio podataka nakon podjele, a to je dio podataka namijenjen za evaluaciju, odnosno procjenu. Procjena nam omogućava provjeru našeg modela na skupu podataka s kojim do sada nije imao doticaja, odnosno s podacima koji se nisu koristili u postupku treniranja. Ovaj korak predstavlja provjeru performansi modela u stvarnom svijetu. Ukoliko je preciznost modela manja ili jednaka 50% znači da model neće biti koristan u primjenu u stvarnom svijetu, ali u slučaju da model može

predvidjeti rezultat u sigurnosti 90% i više možemo vjerovati da će model dobro funkcionirati u stvarnom svijetu.

3.2.6. Podešavanje parametara

U slučaju da u koraku evaluacije nismo zadovoljni s modelom ili smatramo da ga se može poboljšati, moguće je podesiti parametre treniranja modela i na taj način je moguće dobiti bolji model nakon koraka treniranja. Ukoliko model nije zadovoljavajuć moguće je da se javio problem prekomjernog prilagođavanja *(eng. overfitting)* ili premalog prilagođavanja *(eng. underfitting).* Prethodno navedeni problemi će biti navedeni u nastavku. Moguće je napraviti nekoliko stvari kako bi smanjili negativne pojave u našem modelu. Prije svega je moguće povećati broj iteracija treniranja modela. Iteracije treniranja se još nazivaju epohama. Druga mogućnost je podešavanje parametra razine učenja. Razina učenja je vrijednost koja multiplicira gradijent kako bi ga postepeno približila globalnom ili lokalnom minimumu kako bi se minimizirao trošak funkcije. Promjena vrijednosti je iznimno osjetljiva i potrebno je pažljivo mijenjati iste. Minimalne razlike u prethodno navedenim vrijednostima mogu znatno utjecati na vrijeme izvršavanje modela. Podešavanje ovih vrijednosti je iznimno zahtjevno i zahtjeva mnogo pokušaja kako bi se odredile ispravne vrijednosti, s obzirom da se same vrijednosti veoma razlikuju od projekta do projekta.

3.2.7. Predikcija ili zaključivanje

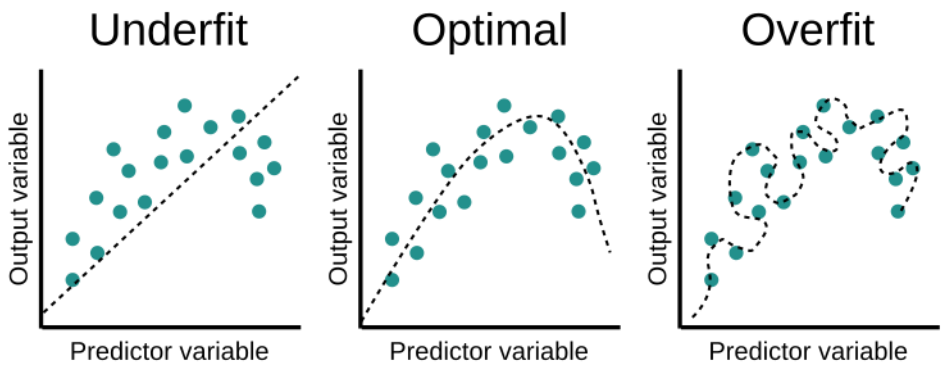
Zadnji korak strojnog učenja je zapravo dobivanje traženih rezultata. Model je spreman za korištenje, njegova vrijednost strojnog učenja je ostvarena i koristi se za dobivanje rezultata.

3.3. Pogreške modela prekomjernog prilagođavanja i premalog prilagođavanja

Pojam prekomjernog prilagođavanja *(eng. Overfitting)* se koristi kada funkcija obraća previše pažnje na određeni skup podataka što uzrokuje loše performanse na neviđenim podacima. Odnosno model modelira podatke treniranja predobro, previše se prilagođava konkretnim podacima za treniranje. Ovo se događa kada model nauči detalje i šum u podacima za treniranje do te razine da negativno utječe na performanse s podacima iz stvarnog svijeta. Nasumične fluktuacije ili buka su primijećeni od strane modela i naučeni kao svojstvo od strane modela. Problem nastaje u tome što dana svojstva ne odgovaraju novim podacima u tu nastaju problemi kod dobivanja rezultata. Prekomjerno prilagođavanje je učestalije kod funkcija koje su s bezparametarskim i nelinearnim modelima i imaju veću fleksibilnost kod učenja ciljane funkcije (Brownlee, 2016).

Premalo prilagođavanje *(eng. Underfitting)* se najjasnije objašnjava na način da funkcija nije sposobna pronaći nikakav uzorak u podacima. Radi se o modelu koji nije sposoban modelirati podatke za treniranje niti ih generalizirati u odnosu na nove podatke. Ovakav model će imati loše performanse već na podacima za treniranje. Premalo prilagođavanje je jednostavno otkriti već na evaluaciji samog modela i najčešće se rješava na način da se odabere drugi, pogodniji algoritam strojnog učenja za dani problem.

U nastavku su vizualno prikazani prethodno navedeni problemu u obliku aproksimacijske funkcije na dvodimenzionalnom grafu.



Slika 7. Preveliko/premalo prilagođavanje modela primjer na grafu (Educative, 2021)

4. Osnovni dijelovi dubokog učenja

Ovo je poglavlje u kojemu se duboko učenje razlaže na njegove osnovne dijelove. Počevši od neuronske mreže i njezinih dijelova, umjetnih neurona, nastoji se objasniti temeljna struktura dubokog učenja. Nakon toga se objašnjava samo treniranje neuronske mreže i elementi o kojima ono ovisi, kao što su aktivacijske funkcije, funkcije gubitka, algoritam širenja unatrag i na kraju se definiraju takozvani hiperparametri čija su podešavanja zadužena za performanse neuronskih mreža. Svaki od dijelova će biti pobliže objašnjen i smješten u kontekst samog dubokog učenja kao nadležne cjeline.

4.1. Neuronske mreže

Neuronske mreže su srž samog dubokog učenja i ovo potpoglavlje će navesti i objasniti pojedine dijelove iste. Neuronske se mreže temelje na strukturi mozga životinja, odnosno čovjeka. Glavna inspiracija dolazi od biološkog neurona koji predstavlja jednu ćeliju unutar mozga koja se veže s ostalim neuronima kako bi komunicirali jedni s drugima. Komunikacija se vrši na način da se prosljeđuju elektro-kemijski impulsi preko sinapsi dok god su ti impulsi dovoljno jaki da pokrenu lančanu reakciju prenošenja od jednog neurona do drugoga.

Neuronska mreža nastoji imitirati ovo ponašanje i to izvršava na sličan način uz pomoć umjetnih neurona. Umjetni neuroni su, baš poput stvarnih, povezani jedni s drugima i nalaze se u više slojeva. Između pojedinih umjetnih neurona se nalaze takozvane težine koje promjenama svojih vrijednosti imitiraju učenje novih informacija i ujedno su glavno sredstvo za dugotrajnu pohranu informacija unutar neuronskih mreža (Patterson & Gibson, 2019). O umjetnim neuronima i samim neuronskim mrežama će biti više riječi u nastavku poglavlja.

Prije nego što se krene na same dijelove neuronske mreže potrebno je razjasniti nekoliko pojmova koje se koriste pri definiranju samog perceptrona, a kasnije umjetnog neurona i na kraju i same neuronske mreže (Patterson & Gibson, 2019).

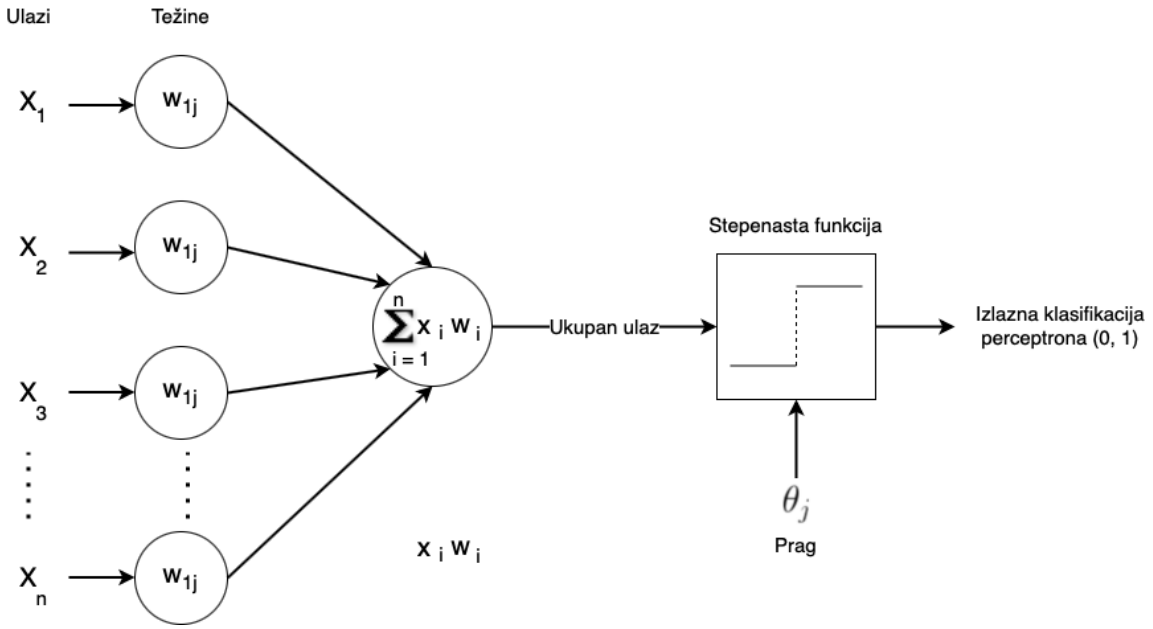
Prvi pojam koji definiramo je **težina veza** *(eng. connection weight)*. Radi se o koeficijentu koji se najčešće prvotno poprima nasumičnu vrijednost koja se kasnije u procesu učenja povećava ili smanjuje, a uloga same težine je pojačavati, odnosno smanjivati ulazni signal pripadajućem neuronu. Težine se nalaze između ulaznih podataka i neurona, ali i između međusobno povezanih neurona.

Drugi pojam su **sklonosti** *(eng. biases)*. Sklonosti su skalarne vrijednosti čija je glavna uloga omogućiti neuronskoj mreži nove interpretacije odnosno ponašanja. Prethodno navedena uloga se izvršava na način da se navedene sklonosti dodaju ulaznom skupu podataka s prethodno postavljenom vrijednosti koja će osigurati da se određeni neuroni aktiviraju na svakom od slojeva bez obzira na ulazne signale.

Treći pojam su **aktivacijske funkcije** *(eng. activation functions)* o kojima će biti više riječi u nastavku, a ovdje ćemo ih samo površno definirati kako bi se lakše mogao pratiti tekst o umjetnim neuronima i mrežama koje su sačinjene od istih. Aktivacijske funkcije su funkcije koje određuju ponašanje umjetnih neurona. Aktivacijske funkcije transformiraju kombinaciju ulaza, težina i sklonosti. Rezultat transformacije prethodno navedenih vrijednosti se prosljeđuje dalje sljedećem sloju neurona ili krajnjem izlazu. Pojam aktiviranog neurona se odnosi na događaj u kojemu umjetni neuron prosljeđuje drugom umjetnom neuronu vrijednost različitu od nula.

4.1.1. Perceptron

Perceptron je temeljni element na kojemu se bazira duboko učenje. Radi se o linearnom modelu koji se koristi za binarnu klasifikaciju i njegova struktura je prikazana na slici u nastavku.



Slika 8. Struktura perceptora (Patterson & Gibson, 2019)

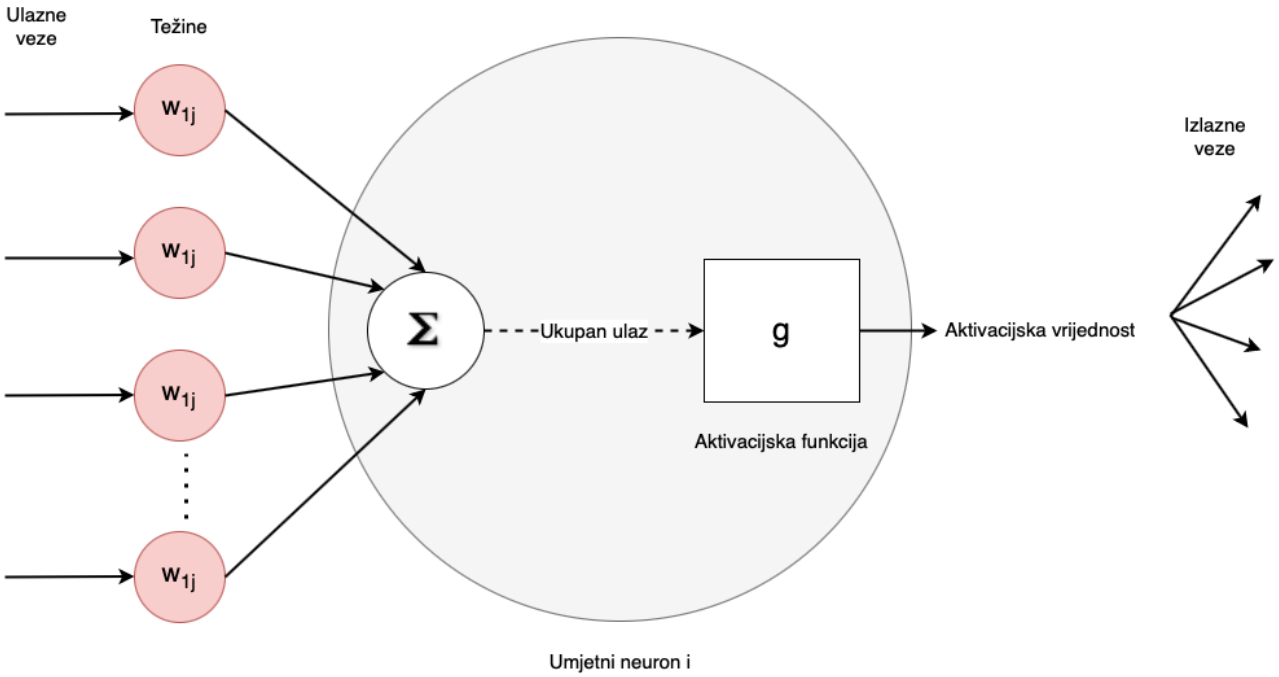
Ideja iz pojedinog perceptora je relativno jednostavna. Svaki perceptor se sastoji od skupa ulaznih podataka koji se množe s odgovarajućom težinom. Nakon toga se radi suma umnožaka i ta suma se prosljeđuje do stepenaste funkcije koja ima definirani prag. Uobičajeno se koristi Heaviside stepenasta funkcija koja je dobila naziv po Oliveru Heavisideu, a rezultira nulom za negativne argumente, odnosno jedinicom za pozitivne (Calvert, 2002). Ukoliko je prethodno navedena suma umnožaka težina i ulaznih vrijednosti veća od definiranog praga, perceptron se aktivira odnosno rezultira jedinicom, u suprotnom slučaju rezultira nulom.

Učenje perceptora se izvršava na način da se inicijaliziraju težine na svakom od pripadajućih ulaza na nasumične male vrijednosti. Nakon toga perceptor uzima svaki zapis podataka koji je predodređen u skupu za treniranje perceptora i računa njegovu klasifikaciju i nakon toga uspoređuje izračunatu klasifikaciju sa stvarnom pridruženom klasifikacijom. Ukoliko se poklapaju ne rade se nikakve promjene, u suprotnom slučaju se prilagođavaju vrijednosti težina kako bi se dobio očekivani rezultat. Svaki ulaz pretstavlja jedan od atributa koji se promatra za dani zapis. Ovaj postupak računanja klasifikacije i prilagođavanja težina se ponavlja sve dok svi primjeri nisu ispravno klasificirani i dok god nije moguće sve primjere linearno dvojiti njihove vrijednosti u hiperravnini (Patterson & Gibson, 2019).

4.1.2.Umjetni neuron

Umjetni neuron je nastao na temelju prethodno navedenog perceptrona i sličnosti su iznimno velike, ali postoji značajna razlika. Ukupni ulaz i dalje ostaje isti kao kod perceptrona, a to je da je ukupni ulaz jednak sumi umnožaka pojedinih ulaza s odgovarajućim težinama,

međutim razlika se javlja u dijelu aktivacijske funkcije koja, u slučaju umjetnog neurona, može poprimiti više oblika. Aktivacijska funkcija na ovaj način može davati znatno kompleksnije aktivacijske izlaze. Detalji umjetnog neurona su prikazani na slici u nastavku.



Slika 9. Umjetni neuron s ulazima i izlazima (Patterson & Gibson, 2019)

Uočavaju se manje razlike umjetnog neurona za razliku od perceptrona. Ulazni umjetni neuroni imaju iste ulaze kao perceptroni s obzirom da se nalaze u prvom sloju neuronske mreže, ali s obzirom da se umjetni neuroni mogu nalaziti između ostalih umjetnih neurona njihovi ulazi su u tom slučaju aktivacijske vrijednosti prethodnih umjetnih neurona, odnosno umjetnih neurona u prethodnom sloju. Ovo znači da će ukupan ulaz biti suma umnožaka aktivacijskih vrijednosti i odgovarajućih težina. Važno je naglasiti da u određenim slučajevima poneki umnožak ulaza i težine može iznositi nula, odnosno to znači da on može biti ignoriran, ako je težina jednaka 0.0 na navedenom ulazu (Patterson & Gibson, 2019).

U nastavku su prikazane matematičke formule za dobivanje izlazne vrijednosti umjetnog neurona koja je objašnjena u prethodnom ulomku.

$$ulazna\_suma_i = W_i * A_i + b$$

Prethodna formula prikazuje dobivanje ulazne sume u i-tom neuronu. W označava vektor koji sadrži sve vrijednosti težina koje se nalaze na ulazima u neuron i, dok A označava vektor koji sadrži sve aktivacijske vrijednosti koje dolaze na ulazima u neuron i. Na kraju imamo oznaku b koja označava dodavanje sklonosti *(eng. bias)*, koja je objašnjena na početku poglavlja.

$$a_i = g(ulazna\_suma_i)$$

Iduća formula koja je prikazana je formula za dobivanje izlaza iz neurona i. Kao što se vidi iz prikazanog prethodno izračunata ulazna suma se stavlja kao ulaz u funkciju g, koja predstavlja aktivacijsku funkciju neurona i. Kao krajnji rezultat dobijemo a_i koji predstavlja izlaz iz neurona. Taj isti izlaz se prenosi dalje drugim neuronima koji se nalaze na sljedećem sloju neurona unutar mreže. Pozicije i postupak prenošenja signala će biti objašnjeni u nastavku ovog rada. Što se tiče izmjena vrijednosti težina, umjetni neuroni se ovdje ponašaju jednako kao i perceptroni i oni prilagođavaju vlastite težine i sklonosti na ulazima kako bi simulirali učenje i propuštali signale na način da se dobivaju traženi rezultati tijekom treniranja.

4.1.3. Višeslojne mreže prosljeđivanja

Višeslojna mreža prosljeđivanja *(eng. Multilayer feed-forward network)* kao što joj samo ime nalaže je mreža koja se sastoji od umjetnih neurona posloženih u više slojeva, gdje se podaci u obliku signala prenose prema naprijed, odnosno možemo reći da se podaci *prosljeđuju* od jednog umjetnog neurona prema sljedećim umjetnim neuronima koji se nalaze u neuronskoj mreži. Važno je naglasiti da je jedan neuron u određenom sloju povezanim sa svim neuronima u sljedećem sloju. Ovaj tip višeslojnih mreža se može trenirati na veći broj načina u kategoriji širenja unatrag *(eng. backpropagation)* (Heaton, 2015). O ovome će biti više riječi u nastavku ovog rada. Prema Pattersonu i Gibsonu razlikuju se tri kategorije slojeva (Patterson & Gibson, 2019).

Prva kategorija je **ulazni sloj** *(eng. input layer)*. Ulazni sloj je sloj neurona koji prvi prihvaćaju ulazni skup podataka. Broj ulaznih neurona je u pravilu uvijek jednak broju ulaznih atributa koje želimo provući kroz neuronsku mrežu. Nakon ulaznih slojeva slijedi jedan ili više takozvanih skrivenih slojeva, s kojima su umjetni neuroni ulaznog sloja u potpunosti povezani, što znači da svaki umjetni neuron koji se nalazi na ulaznom sloju svoj rezultat prosljeđuje svim umjetnim neuronima koji se nalaze na sljedećem sloju. Ova povezanost će biti jasno vidljiva na slici u nastavku ovog potpoglavlja.

Druga kategorija je već navedeni **skriveni sloj** *(eng. hidden layer)*. Skriveni sloj se naziva tako zato što se nalazi između dva sloja koja imaju doticaj s vanjskom okolinom i iz tog razloga nemam izravan pristup njemu. Skriveni sloj je ključan dio svake neuronske mreže zato što su težine koje se prilagođavaju unutar navedenog sloja glavni način na koji neuronske mreže uče.

Treća kategorija je **izlazni sloj** *(eng. output layer)*. Ovdje se radi o sloju koji izbacuje krajnje rezultate iz neuronske mreže. Najčešće su to konkretni odgovori ili određena predviđanja iz modela kojeg smo kreirali. Broj umjetnih neurona koji se nalaze u izlaznom sloju

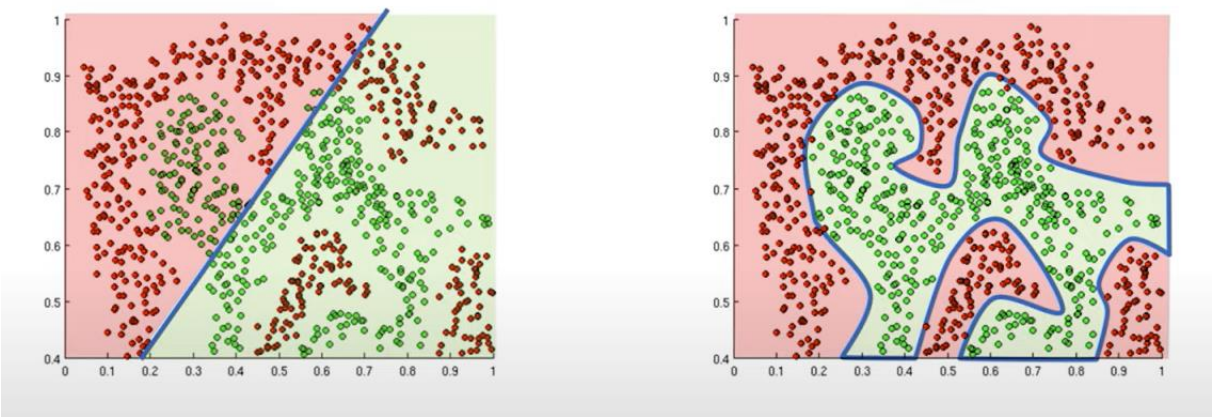
ovisi o ulaznom sloju, odnosno ulaznim podacima. Izlaz može poprimiti više oblika, a on ovisi o tome kako je neuronska mreža postavljena. Izlaz može biti realan broj, u slučaju regresije, ili skup vjerojatnosti pripadanja klasama u slučaju da se radi o klasifikaciji.

4.2. Treniranje neuronskih mreža

Kao što je već više puta navedeno u ovom radu, sam proces učenja u neuronskim mrežama se svodi na prilagođavanje vrijednosti težina i sklonosti. Na ovaj način naš model može efikasno procjenjivati koji dio informacije utječe u kojem udjelu na to koji će biti krajnji ishod same predikcije. U velikom broj skupova podataka se često nalazi snažna korelacija između jednog atributa i određene oznake, odnosno kategorije. Konkretan primjer iz stvarnog svijeta bi bila korelacija između cijene automobila i zapremine motora u slučaju predviđanja cijene automobila na tržištu polovnih vozila. Atribut cijene vozila ima veliku korelaciju sa zapreminom motora na način da veća zapremina povlači za sobom veću cijenu vozila. Neuronske su mreže u postupku treniranja same zadužene da u određenom broju pokušaja dođu do zaključka koji dijelovi informacija utječu više, a koji dijelovi manje u postupku dobivanja krajnje predikcije. Nakon što se težine i sklonosti prilagode dovoljan broj puta, želimo dobiti neuronsku mrežu koja pojačava korisne ulaze, a smanjuje loše. Veća težina na određenoj vezi označava veću važnost, odnosno veću korelaciju danog signala i krajnjeg rezultata same mreže (Patterson & Gibson, 2019). Postavljanje vrijednosti težina i sklonosti bit će objašnjeno u nastavku s algoritmom širenja unatrag *(eng. backpropagation)* koji je jedan od algoritama koje usko vežemo sa samim dubokim učenjem.

4.3. Aktivacijske funkcije

Kod umjetnih neurona su se spominjale aktivacijske funkcije, a u ovom dijelu rada će se one razraditi detaljnije. Aktivacijske funkcije *(eng. activation function)* su funkcije koje se nalaze unutar pojedinih umjetnih neurona i koje su zadužene za propagaciju signala dalje u neuronskoj mreži. Možemo reći da one definiraju ograničenja unutar kojih zadana vrijednost mora biti kako bi se pojedini neuron 'aktivirao' i poslao signal dalje u mreži. Glavna uloga aktivacijskih funkcija je uvođenje nelinearnosti u neuronsku mrežu, zato što se radi o nelinearnim funkcijama. Ovo omogućava neuronskim mrežama da se koriste na nelinearnim podacima, što je iznimno važno s obzirom da su gotovo svi podaci iz stvarnog svijeta nelinearni.



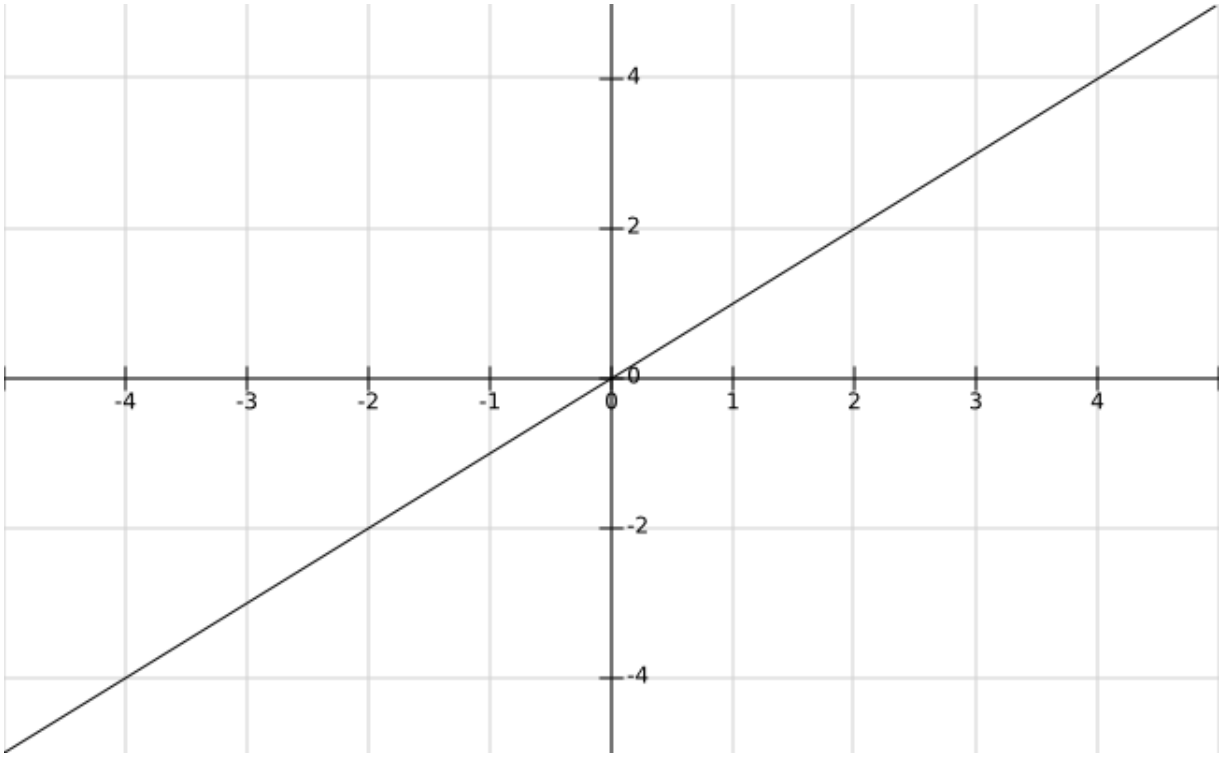
Slika 10. Razlika između linearnog pristupa odvajanja skupa podataka i nelinearnog (Amini, 2021)

U objašnjenu perceptrona stoji da se sastoji od isključivo jedne funkcije, dok kod umjetnih neurona postoji mnoštvo mogućih aktivacijskih funkcija. U nastavku ovog poglavlja će se navesti neke od najčešćih funkcija koje se koriste kao aktivacijske funkcije u neuronskim mrežama. Funkcije koje se navode su predložene u radu *Artificial Inteligence for Humans*, autora Jeff Heatona (Heaton, 2015).

4.3.1. Linearna aktivacijska funkcija

Linearna aktivacijska funkcija je jedna od najjednostavnijih mogućih funkcija koje se koriste u neuronskim mrežama. Radi se o funkciji koja ne mijenja izlaz neurona s obzirom na njegov ulaz, što u praksi znači da neuron samo prosljeđuje vrijednost koja mu dana.

$$f(x) = x$$



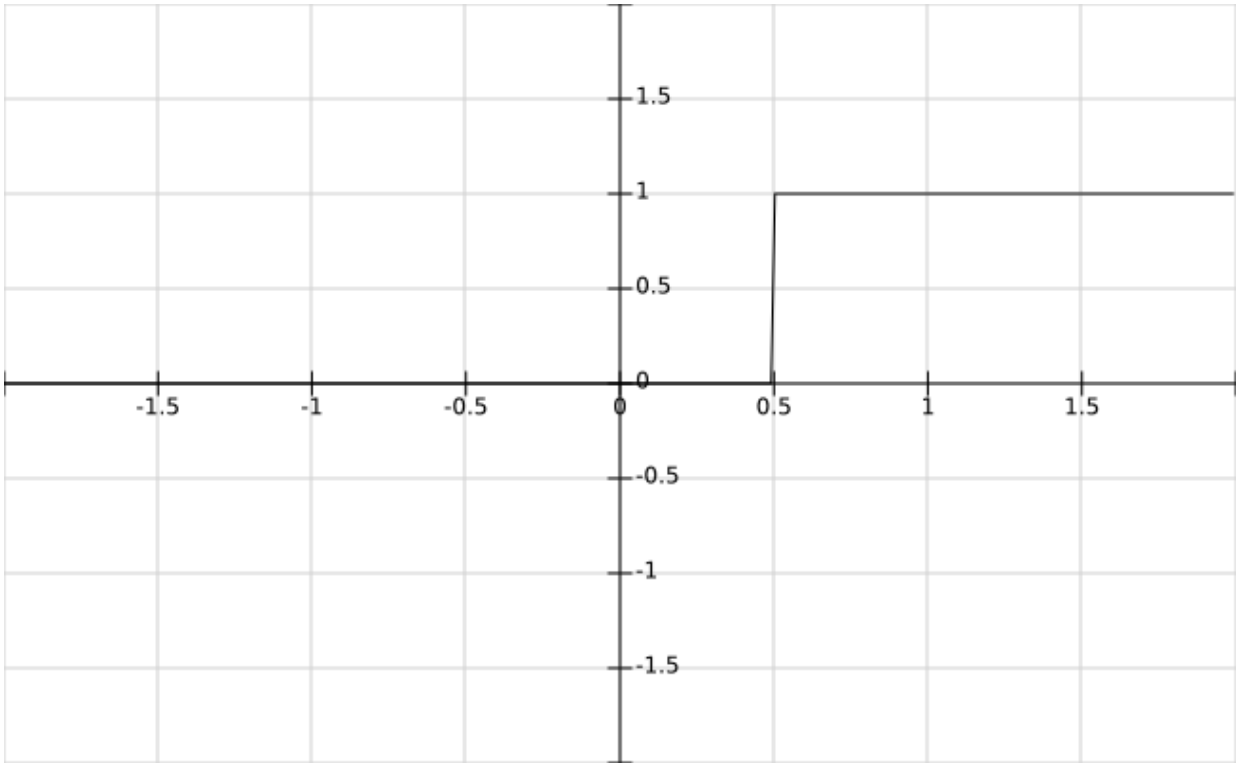
Slika 11. Linearna aktivacijska funkcija

Na prethodnoj slici je prikazana linearna funkcija na kojoj je jasno vidljivo da se ulazni skup podataka samo preslikava kao izlaz. Ovaj tip funkcije se koristi u posljednjem, odnosno izlaznom sloju neuronskih mreža koje su zadužene za izdavanje brojčanih vrijednosti. Takve neuronske mreže se nazivaju regresijske neuronske mreže.

4.3.2.Stepenasta aktivacijska funkcija

Funkcija koja se koristi kod perceptrona, naziva se stepenasta aktivacijska funkcija. Radi se o još jednoj trivijalnoj funkciji koja ima predefiniran određeni prag, ukoliko ulazna vrijednost prelazi navedeni prag izlazna vrijednost je jedan, u suprotnom ona je nula.

$$f(x) = \begin{cases} 1, ako\ je\ x \geq 0.5 \\ 0, ako\ je\ x\ < 0.5 \end{cases}$$

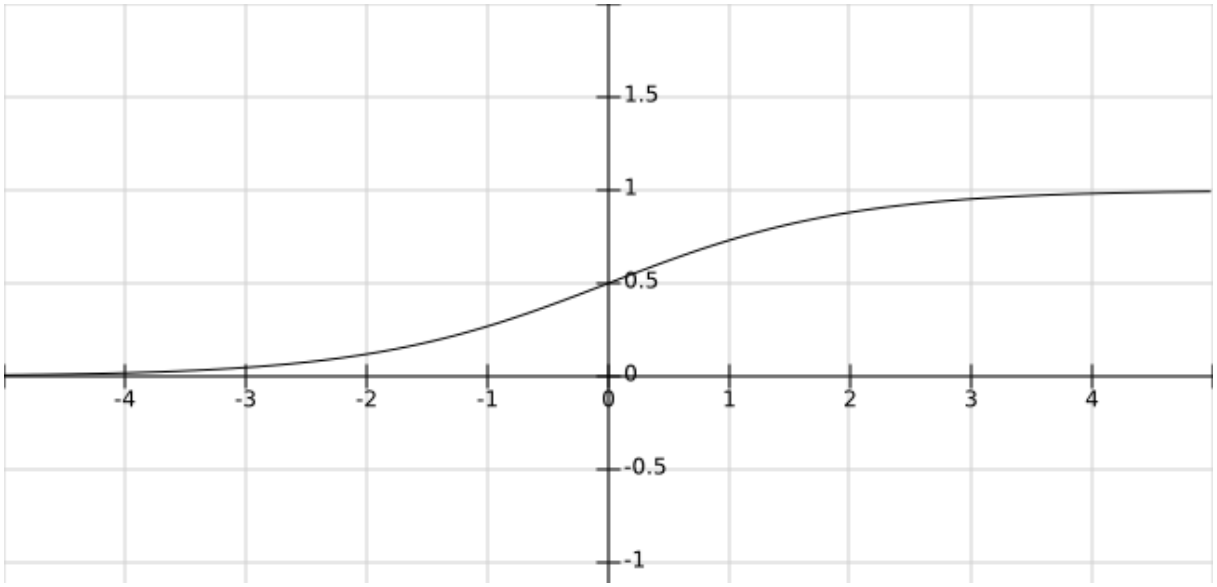


Slika 12. Stepenasta aktivacijska funkcija

4.3.3.Sigmoidna aktivacijska funkcija

Jedna od najčešćih aktivacijskih funkcija u neuronskim mrežama prosljeđivanja je sigmoidna aktivacijska funkcija. Često se koristi u slučaju kada traženi izlaz treba biti pozitivna vrijednost. Također se koristi u slučaju kada traženi izlaz treba predstavljati vjerojatnost nečega, s obzirom da je vrijednost izlaza sigmoidne funkcija uvijek između 0 i 1, što je vidljivo u slici same funkcije u nastavku.

$$f(x) = \frac{1}{1 + e^{-x}}$$

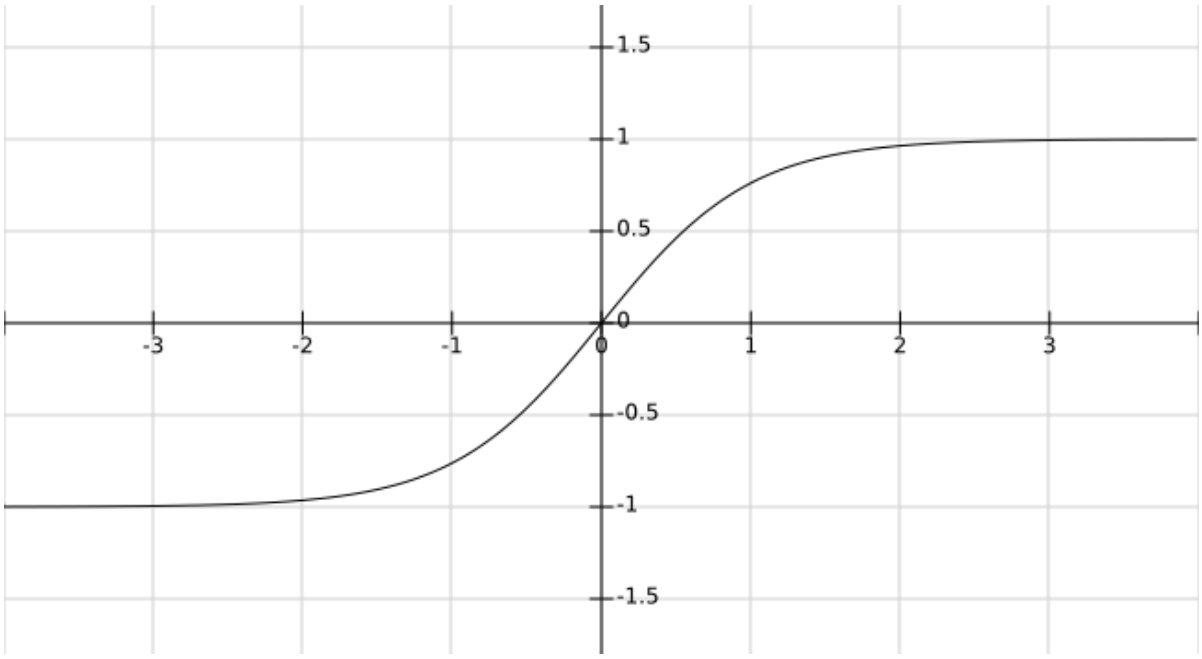


Slika 13. Sigmoidna aktivacijska funkcija

4.3.4.Hiperbolička tangentna aktivacijska funkcija

Hiperbolička tangenta aktivacijska funkcija se koristi u slučaju kada neuronska mreža treba pružati rezultate u rasponu od -1 do 1. Ova aktivacijska funkcija ima više prednosti u odnosu na prethodne, a jedna od glavnih prednosti je ta što je znatno pogodnija za korištenje kada se radi s negativnim brojevima.

$$f(x) = \tanh(x)$$

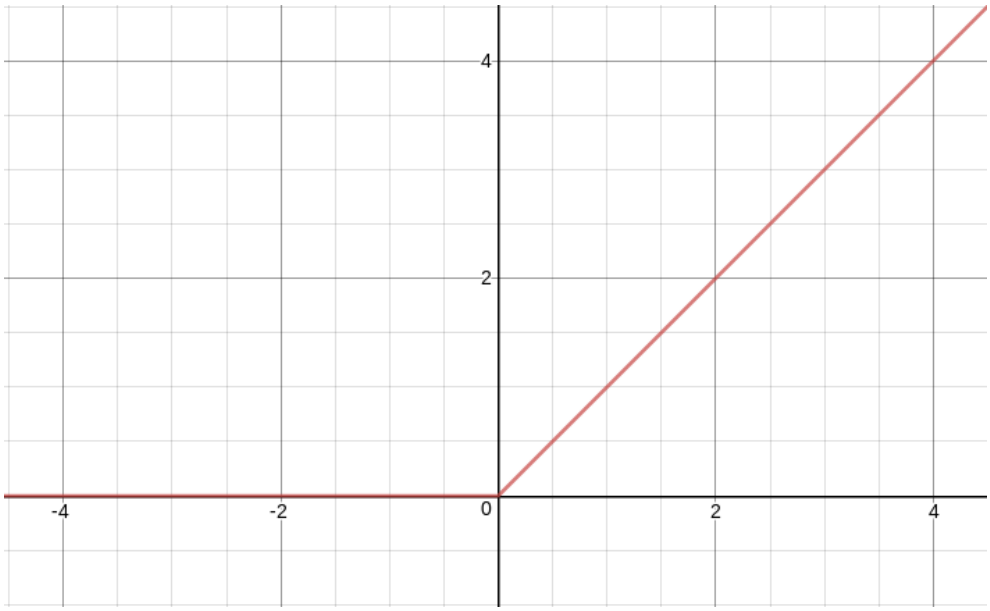


Slika 14. Hiperbolička tangentna aktivacijska funkcija

4.3.5.Ispravljena linearna jedinica

Ispravljena linearna jedinica *(eng. rectified linear units, ReLU)*, je funkcija koja se trenutno nameće kao najbolja aktivacijska funkcija u neuronskim mrežama. Prethodno je to bila hiperbolička tangentna aktivacijska funkcija, ali jednostavnost same funkcije i njeni rezultati treniranja su pokazali da ReLU funkcija daje znatno bolje rezultate. ReLU aktivacijska funkcija se koristi na skrivenim slojevima neuronske mreže u kombinaciji s linearnim aktivacijskim funkcijama na izlaznom sloju (Heaton, 2015).

$$f(x) = \max(0, x)$$



Slika 15. Ispravljena linearna jedinica

Kao što je vidljivo na prethodnoj slici izlazna vrijednost funkcije je nula do određene vrijednosti, u ovom slučaju je to nula, nakon koje izlaz ima linearan odnos s varijablom o kojoj ovisi, što je ovom primjeru x. U nastavku rada će biti navedeno zašto je ova aktivacijska funkcija bolja u odnosu na prethodno naveden.

4.4. Funkcija gubitka

Funkcija gubitka *(eng. loss function)* kvantificira razliku, odnosno gubitak između rezultata predviđanja neuronske mreže i očekivanog rezultata. Uz pomoć funkcije gubitka se dobiva uvid u to koliko je dani model sposoban davati idealne rezultate. Što je manja vrijednost gubitka to su bolje performanse samog modela.

Prilagođavanjem vrijednosti težina i sklonosti se nastoji smanjiti gubitak u neuronskoj mreži. Nije moguće jednostavno izračunati vrijednosti težina u neuronskoj mreži zbog prevelike zavisnosti i prevelikog broja nepoznanica. Iz tog razloga se problem prilagođavanja težina, odnosno problem učenja svodi na problem optimizacije. Koristi se algoritam uz pomoću kojega se ciljana vrijednost pojedine težine kreće po prostoru mogućih vrijednosti u pokušaju da se nađe najbolja moguća vrijednost. Za ove potrebe optimizacije se najčešće koristi stohastički optimiziacijski algoritam gradijenta silaska *(eng. stohastic gradient descent optimization algorithm)* i težine se ažuriraju uz pomoć algoritma širenja unatrag koji će biti razjašnjen u nastavku. Gradijent se odnosi na gradijent greške, a cilj algoritma gradijenta silaska je kretati se niz gradijent greške (Brownlee, 2019b).

U nastavku odjeljka se navode dvije najčešće korištene funkcije gubitka od kojih se jedna koristi u slučaju regresije dok se druga koristi u slučaju klasifikacije.

Funkcija gubitka koja se koristi kod regresije, odnosno u slučaju kada kao izlaz neuronske mreže očekujemo realni broj se naziva gubitak kvadrirane srednje vrijednosti greške *(eng. mean squared error loss)*. Ovo je funkcija koja mjeri kvadriranu razliku između očekivane vrijednosti i dobivene predikcije na prosjeku rezultata na temelju cijelog ulaznog skupa podataka neuronske mreže (Amini, 2021).

$$J(W) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - f(x^i; W))^2$$

U prethodnoj formuli n predstavlja broj slučajeva, a unutar zagrada $y^{(i)}$ predstavlja očekivanu stvarnu vrijednost, dok $f(x^i; W)$ predstavlja predviđenu vrijednost.

Funkcija gubitka koja se koristi kod klasifikacije se naziv gubitak unakrsne entropije *(eng. cross entropy loss).* Ova funkcija se još naziva i logaritamski gubitak, a koristi se u slučaju kada su izlazne vrijednosti 0 ili 1. Uspoređuje se dobivena predikcija sa stvarnim rezultatom i gubitak je, shodno nazivu, logaritamske prirode što znači da pokazuje veće gubitke bliže jedinic i manje gubitke koje teže nuli (Koech, 2020).

$$J(W) = -\sum_{i=1}^{c} y^{(i)} \log\left(f(x^i; W)\right)$$

U prethodnoj formuli c predstavlja broj mogućih klasa, a $y^{(i)}$ predstavlja očekivanu stvarnu vrijednost, dok $f(x^i; W)$ predstavlja predviđenu vrijednost.

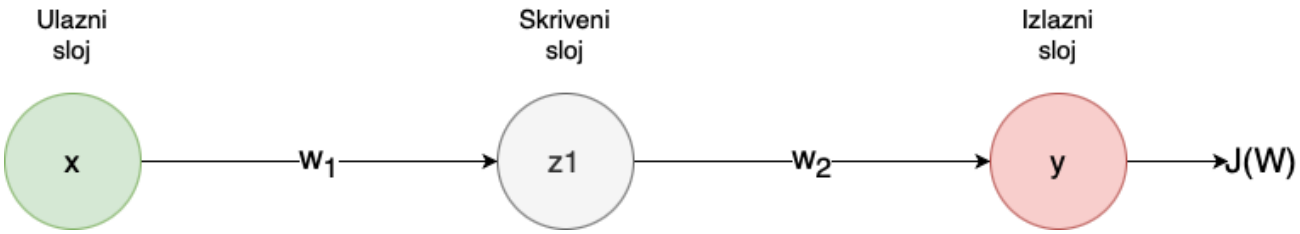
4.5. Algoritam širenja unatrag

Glavna svrha algoritma širenja unatrag *(eng. backpropagation algorithm)* je ukazati, odnosno odrediti u kojem smjeru treba mijenjati određene težine, treba li ih povećati ili smanjiti kako bi se smanjila krajnja vrijednost funkcije gubitka. Kako mu samo ime nalaže, algoritam širenja unatrag se izvodi na način da informacija ide od kraja prema početku neuronske mreže i na taj način omogućava računanje gradijenta čija će vrijednost pokazati koji je idući korak u mijenjanju pripadajućih težina (Goodfellow & Bengio, 2016).

Algoritam širenja unatrag koristi pravilo lanca *(eng. chain rule)* koje se veže uz derivacije i integrale, a izgleda ovako:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Kako bi se lakše razumio sam algoritam bit će prikazana trivijalna neuronska mreža na kojoj će se prikazati formule za računanje gradijenta i ideje iza njih.



Slika 16. Trivijalna neuronska mreža

Na prethodnoj slici se nalazi trivijalna neuronska mreža s funkcijom gubitka J(W), ulazom x, izlazom y i aktivacijskom funkcijom z1. Recimo da želimo izračunati gradijent za w_2.

$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial y} * \frac{\partial y}{\partial w_2}$$

U slučaju da želimo izračunati vrijednost za w_1 ponovno koristimo prethodno navedeno pravilo lanca, ali u ovom slučaju rekurzivno raspisujemo dio jednadžbe.

$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial y} * \frac{\partial y}{\partial w_1}$$

Rekurzivno se primjenjuje pravilo lanca na dio $\frac{\partial y}{\partial w_1}$ nakon čega jednadžba na način kako je prikazano u nastavku.

$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial y} * \frac{\partial y}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Na zadnjoj jednadžbi se lako uočava kako se od kraja neuronske mreže putem rekurzije i proširivanja naše jednadžbe dolazi do težina na prethodnim slojevima neuronske mreže, konkretno kako smo od kraja neuronske mreže preko težine w_2 došli do težine w_1.

4.6. Hiperparametri

Hiperparametri su parametri u strojnom učenju čijim se podešavanjem nastoje poboljšati performanse neuronskih mreža. Razlika između parametara i hiperparametara je u tome što su parametri sami procijenjeni iz skupa podataka, dok su hiperparametri ručno definirani. Odabirom hiperparametara se nastoji osigurati da model nema problem prevelikog, odnosno premalog prilagođavanja skupu podataka za treniranje kako bi se osigurala dobra predikcija modela. Prema Gibsonu glavni hiperparametri su sljedeći (Patterson & Gibson, 2019).

4.6.1. Stopa učenja

Stopa učenja *(eng. learning rate)* je hiperparametar koji je zadužen koliko će se model mijenjati u odnosu na procjenjenu grešku na svakom periodu u kojemu se mijenjaju težine. Vrijednost ovog hiperparametra može predstavljati potencijalan problem u neuronskim mrežama. Ukoliko je vrijednost premala, vremenski kao i hardverski trošak će biti znatno veći nego što je to optimalno, što u procesu kao što je duboko učenje, koje samo po sebi vremenski i hardverski iznimno zahtjevno, predstavlja problem. U suprotnom slučaju postavljanje prevelike vrijednosti može rezultirati u tome da su pojedinačne promjene težina prevelike što na kraju može rezultirati u prelasku preko minimuma ili nestabilnom procesu treniranja.

4.6.2. Regularizacija

Glavna uloga regularizacije *(eng. regularization)* je kontrolirati razinu prevelikog prilagođavanja. Model s prevelikim težinama je često znak da se isti previše prilagodio skupu podataka za treniranje, što će rezultirati u nepouzdanim predikcijama u slučaju podataka koji nisu bili dio skupa za treniranje. Regularizacija težina potiče manje težine, a suzbija veće i na taj način snižava kompleksnost samog modela (Brownlee, 2019a). Važno je naglasiti da je najbolji način, u praktičnoj primjeni, za osiguravanje dobrih težina i bez velike regularizacije potreban što veći skup podatak za treniranje.

4.6.3. Momentum

Momentum je hiperparametar koji je zadužen za izbavljivanje algoritama iz pozicija u kojim bi inače zapeo na način da se definira svojevrsna količina gibanja danog algoritma, odnosno njegov momenutm. Momentum izvršava ispunjava svoju svrhu na način da on zapravo mijenja stopu učenja, na sličan način kako stopa učenja utječe na promjenu težina u određenom periodu. Promjenom stope učenja, mijenja se i brzina, odnosno momentum, kojom se kreće težina prema svojem optimumu. Težina se mijenja u većim inkrementima i na taj način se mogu izbjeći određene situacije u kojemu bi inkrementalna promjena težina bila premala i sustav bi 'zapeo' u beskonačnoj petlji promjeni vrijednosti koja ne bi težila zadovoljavajućoj vrijednosti.

4.6.4. Rijetkost

Rijetkost *(eng. sparsity)* je hiperparametar koji prepoznaje važnost značajki za određene ulazne podatke. Ukoliko dani hiperparametar može prepoznati, on prepoznaje da vrijednost određenih značajki nije relevantna u određenim skupovima podataka i kao takve ih ne uzima u obzir. Hiperparametar rijetkost se naziva rijetkost iz razloga što se radi o značajkama koje su iznimno rijetke i kao takve ih ne želimo da ometaju sposobnost neuronske mreže u učenju.

5. Visoka razina dubokog učenja

U prethodnom poglavlju su bili objašnjeni koncepti niske razine dubokog učenja, pojedinačni dijelovi od kojih se duboko učenje gradi. Ovo poglavlje, s druge strane, se bavi konceptima visoke razine. Riječ je o vrstama arhitekture dubokih mreža i na koji način se kreiraju iste.

5.1. Ponavljajuće neuronske mreže

Ponavljajuće neuronske mreže *(eng. recurrent neural networks)* su mreže koje se temelje na neuronskim mrežama prosljeđivanja *(eng. feedforward neural networks)*, a prvenstveno se koriste za modeliranje sekvenci. Za samu strukturu neuronske mreže to znači da sama neuronska mreža mora uzeti u obzir još jednu dodatnu komponentu, a to je komponenta vremena, odnosno određeno stanje u određenom vremenskom periodu. Ponavljajuće neuronske mreže su sposobne obrađivati sekvence neodređene veličine uz pomoć vlastitih unutarnjih stanja (Alom et al., 2019). Svaki zapis koji je dio sekvence predstavlja jedan trenutak, odnosno korak u vremenu.

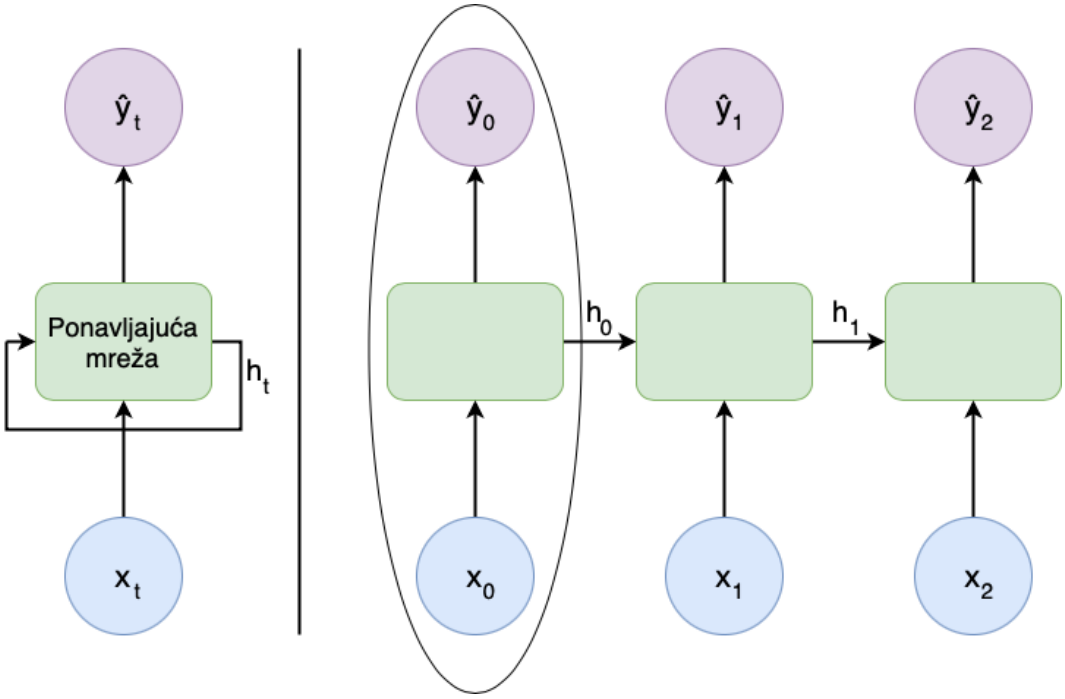
Primjene modeliranja sekvenci se dijele u tri dijela (Patterson & Gibson, 2019):

- Jedan-prema-više *(eng. one-to-many)*: izlaz je sekvenca, a ulaz je jedan. Radi se o slučajevima u kojima imamo jedan ulaz kao što je slika, a sekvencu kao izlaz, za čiji primjer možemo uzeti generiranje opisa slika. Ovo bi značilo da za jedan ulaz, koji je slika, kao rezultat želimo dobiti sekvencu riječi koja će sačinjavati rečenicu koja opisuje što se nalazi na slici.
- Više-prema-jedan *(eng. many-to-one)*: izlaz je jedan, a sekvenca je ulaz. Radi se o slučajevima u kojima imamo sekvencu kao ulaz, možemo uzeti za primjer niz riječi koje sačinjavaju rečenicu, a za izlaz želimo izbaciti rezultat koji će definirati koji osjećaj se nalazi iza same rečenica (npr. osjećaj sreće, ljutnje, zadovoljstva, nelagode itd.)
- Više-prema-više *(eng. many-to-many)*: izlaz i ulaz su sekvence. Jedan od primjera je strojno prevođenje, gdje se ulaz koji je sekvenca, recimo niz riječi na jednom jeziku, prevodi u drugu sekvencu riječi na drugom jezik.

U nastavku ovog potpoglavlja je razjašnjena arhitektura ponavljajuće neuronske mreže, kao i pojedinačni dijelovi od kojih se sastoji.

5.1.1. Neuroni s ponavljanjem

U već poznatoj neuronskoj mreži prosljeđivanja ne postoji način na koji možemo obrađivati podatke na način da se uzima u obzir njihova pozicija u vremenu, nego se dani ulaz obrađuje na način kao da se uzimaju vrijednosti iz jedne fiksne točke u vremenu.



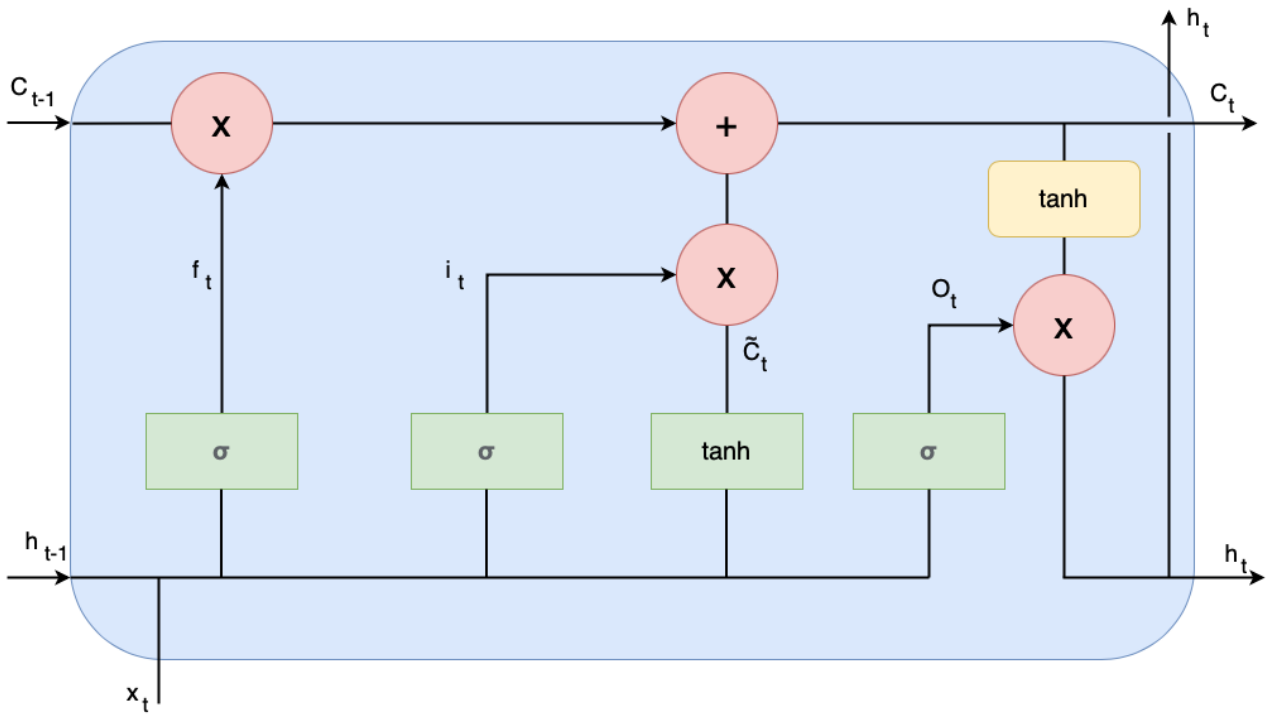
Slika 17. Ponavljajući neuroni razloženi u korake(Alom et al., 2019)

Na prethodnoj slici je prikazana ideja iza ponavljajućih neuronskih mreža. Oznaka **x** predstavlja ulazni vektor u neuronsku mrežu, zeleni oblici predstavljaju skriveni sloj neuronske mreže i na posljetku oznaka **ŷ** predstavlja izlazni vektor neuronske mreže. Okomita linija odvaja s lijeve strane kompaktni prikaz od razloženog prikaza po koracima s desne strane. Zaokruženi dio na slici možemo smatrati kao izolirani trenutak u vremenu. U slučaju da uzmemo samo zaokruženi dio, znamo da izlaz ovisi isključivo o vektoru ulaza, odnosno $\hat{y}_t = f(x_t)$. Uzevši u obzir da znamo da radimo sa sekvencijalnim podacima nužno je uzeti u obzir ulaze iz prethodnih stanja kada obrađujemo kasnija stanja. Ovo se postiže na način da se povezuje prethodni korak, odnosno stanje sa sljedećim. Povezivanje se izvršava na način da se uvodi unutarnja memorija ili pamćenje koje je zaduženo za pamćenje sekvence u određenom koraku i prosljeđivanju navedenog stanja idućem koraku obrade. Ovaj prijenos informacija je vidljiv na prethodnoj slici s oznakom **h**, gdje **h** predstavlja memoriju. Jednom nakon što je ovako postavljena neuronska mreža, izlaz više nije ovisan samo o ulazu već i o prethodnom stanju. $\hat{y}_t = f(x_t, h_{t-1})$, gdje je $\hat{y}_t$ izlaz, x_t ulaz, a h_{t-1} prethodna memorija koja sadrži informacije o stanju.

5.1.2. Dugotrajno kratkotrajna memorija

Glavni problem koji se pojavljuje kod ponavljajućih neuronskih mreža su nestajući ili eksplodirajući gradijenti. Problem eksplodirajućih gradijenata je kada je većina gradijenata prevelika i ne može se doći do optimuma. S druge strane problem nestajućih gradijenata je kada gradijenti postaju sve manji i manji do razine kada više nije moguće efikasno trenirati neuronsku mrežu. Oba navedena slučaja dolaze do izražaja tek u dugoročnijima ovisnostima odnosno u slučajevima gdje se između ovisnosti nalazi 10 ili više koraka unutar ponavljajuće neuronske mreže (Patterson & Gibson, 2019). Kao rješenje ovog problema se nameće kompleksnija ponavljajuća jedinica unutar ponavljajuće neuronske mreže, a to je dugotrajno kratkotrajna memorija *(eng. long short term memory)*.

Dugotrajno kratkotrajni memorijski modul u sebi sadrži mnoštvo takozvanih vrata koja su zadužena za propuštanje odnosno blokiranje pojedinih informacija. Na slici u nastavku se nalazi unutarnja struktura prethodno navedenog modula.



Slika 18. Struktura dugotrajno kratkotrajne memorije (Alom et al., 2019)

Na prethodnoj slici vidimo, između ostalog, aktivacijske funkcije u zelenom pravokutniku, a to su sigmoide. Izlazna vrijednost sigmoide je uvijek vrijednost između 1 i 0 i kao takva ona djeluje kao vrata koja propuštaju informaciju, ukoliko je izlazna vrijednost 0 ništa od informacija neće biti propušteno, dok u suprotnom slučaju ukoliko je izlazna vrijednost 1, informacija će biti propuštena u potpunosti.

Dugotrajno kratkotrajne memorije odrađuju 4 funkcije unutar sebe:

- Zaboravljanje
- Spremanje
- Ažuriranje
- Izlaz

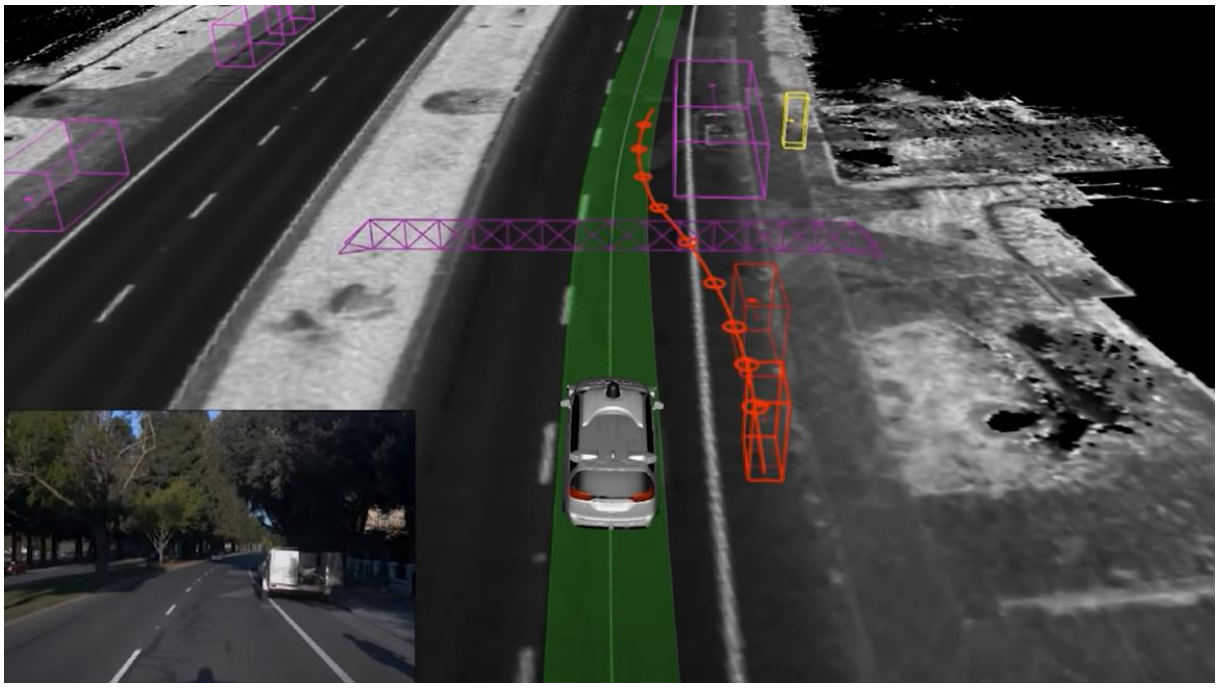
Prvi korak je zaboraviti nevažne informacije iz prethodnih stanja. Ovaj korak se izvršava na način da se uzima prethodno stanje i provlači kroz sigmoidna vrata, govorimo o liniji koja je prikazana na prethodnoj slici s oznakom f_t. Kao što je prethodno rečeno sigmoidna vrata određuju u kojoj će mjeri vrijednost biti propuštena. Drugi korak je utvrditi koje od prethodnih informacija su važne i koje je potrebno pohraniti. Ovaj dio je vidljiv na prethodnoj slici, a radi se o liniji s oznakom i_t. Izlazna vrijednost sigmoide i hiperboličkog tangensa se množi i kasnije prosljeđuje do stanja ćelije. Stanje ćelija je vrijednost koju dugotrajno kratkotrajne memorije spremaju odvojeno od prethodnih stanja i služi kao dodatna referentna vrijednost koja se ažurira u trećem koraku. Radi se o oznaci c_t kojoj pripada gornja horizontalna linija i ažurira se na temelju prethodno navedenih aktivacijskih funkcija koje su jasno naznačene na prethodnoj slici. Četvrti i posljednji korak je vraćanje izlaza. Na slici prikazana linija s oznakom O_t. Ovo je korak u kojemu izlazna vrata odlučuju što se propušta dalje u sljedeći korak, ovdje se prenosi izlazno stanje, kao i stanje ćelije (Alom et al., 2019).

5.1.3. Primjena ponavljajućih neuronskih mreža

Primjena ponavljajućih neuronskih mreža je iznimno široka. Ovaj odjeljak navodi neke od najčešćih primjena (Alom et al., 2019; Patterson & Gibson, 2019).

- Automatsko generiranje naslova slike *(eng. image captioning)*: postupak automatskog generiranja naslova slike na temelju onoga što se nalazi na samoj slici. Ovaj postupak se ponekada temelji na prepoznavanju lica i mjesta.
- Analiza osjećaja *(eng. sentiment analysis)*: postupak automatskog identificiranja i kategoriziranja osjećaja koje se prenosi putem teksta. Nastoji se utvrditi pozitivan, negativan ili neutralan osjećaj prema onome što je navedeno u tekstu.
- Modeliranje jezika *(eng. language modelling)*: postupak modeliranja teksta koji daje više mogućnost obrade teksta kao što je dovršavanje teksta i sažimanje teksta.
- Generiranje glazbe *(eng. music generation)*: automatsko generiranje glazbe na temelju ulaznog skupa podataka u fazi treniranja

- Strojno prevođenje *(eng. machine translation)*: automatsko prevođenje teksta s jedno jezika na drugi. Primjer strojnog prevođenja je najpoznatija aplikacija za prijevod u svijetu, a to je Google Translate.
- Predviđanje kretanja vozila *(eng. vehicle trajectory prediction)*: Predviđanje putanje vozila kod autonomnih automobila. Radi se o sustavu koji je sposoban predvidjeti putanju okolnih vozila s obzirom na okolinu u kojoj se nalaze.

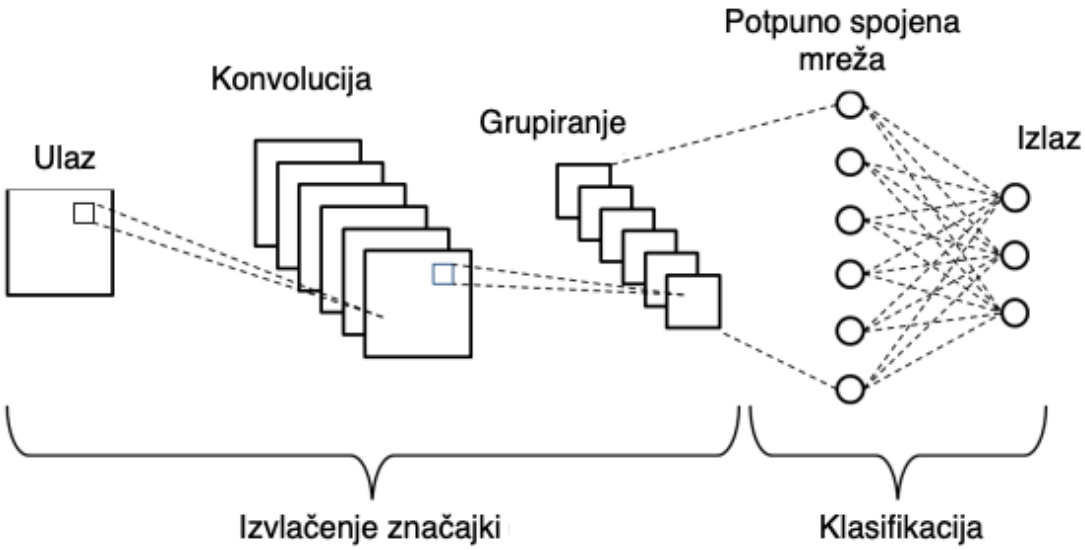


Slika 19. Predikcija putanje vozila za autonomna vozila (Amini & Soleimany, 2021)

5.2. Konvolucijske neuronske mreže

Konvolucijska neuronska mreža *(eng. convolutional neural network)* je neuronska mreža koja se koristi u području računalnog vida *(eng. computer vision)*. Radi se o mreži koja omogućava smisleno dodjeljivanje težina i sklonosti pojedinim značajkama slike kako bi se iste značajke mogle naknadno obrađivati, odnosno razlikovati.

Arhitektura konvolucijskih neuronskih mreža se temelji na istom principu na kojemu je organiziran vizualni korteks u ljudskom mozgu. Pojedini neuroni obrađuju samo određene dijelove vidnog polja, a ne cijelo vidno polje. Prethodno navedeni dijelovi vidnog polja sačinjavaj niz takozvanih pločica koje se u mozgu spajaju kako bi kreirali cijelu sliku (Patterson & Gibson, 2019).

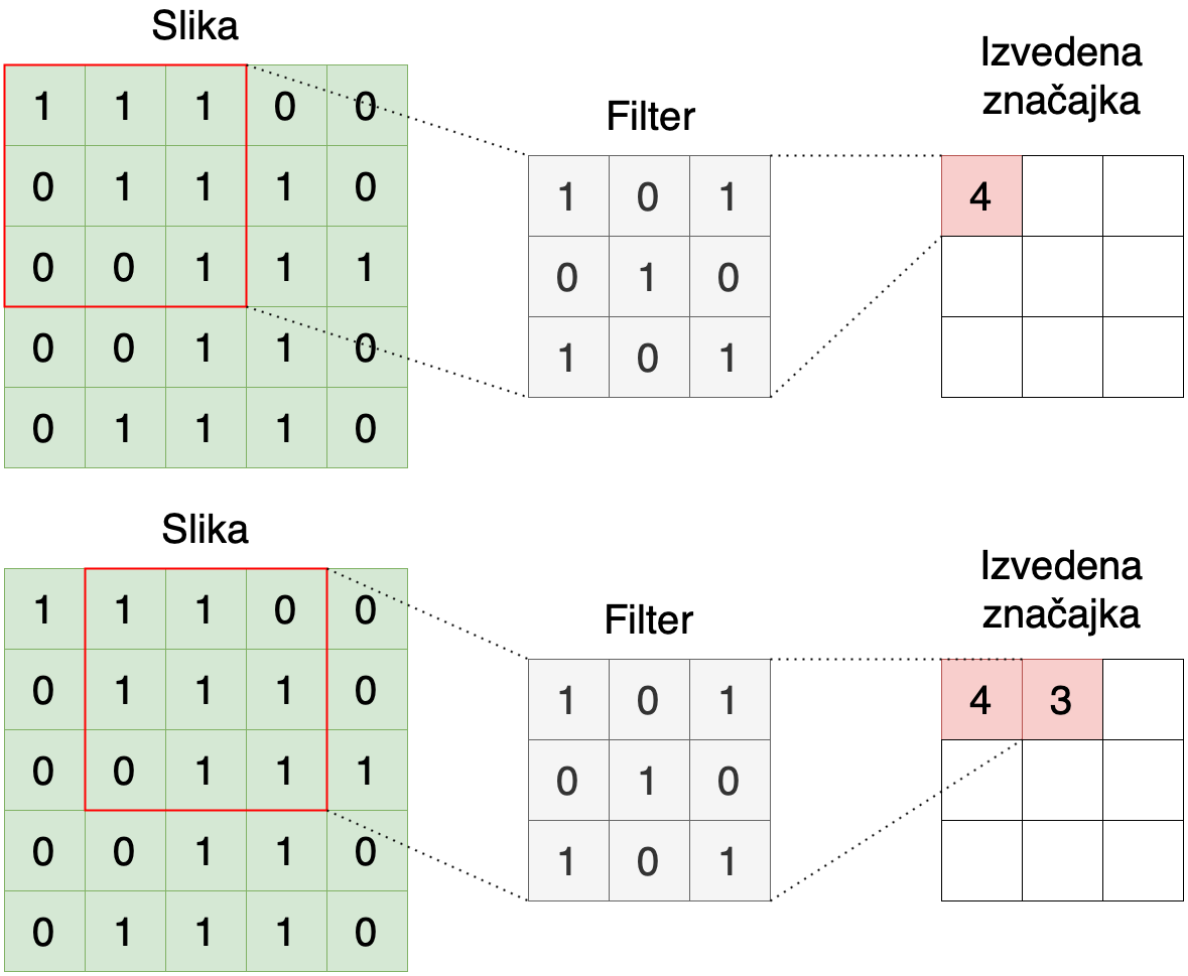


Slika 20. Jednostavni prikaz arhitekture konvolucijske neuronske mreže (Phung & Rhee, 2019)

Na prethodnoj slici je vidljivo kako se konvolucijska neuronska mreža sastoji od dva glavna dijela, a to su **izvlačenje značajki** i **klasifikacija**. Kao što je vidljivo na slici konvolucijska neuronska mreža u sebi sadrži tri moguća sloja: konvolucija *(eng. convolution)*, grupiranje *(eng. pooling)* i potpuno spojena mreža, koja je zadužena za klasifikaciju. U dijelu izvlačenja značajki slojevi konvolucije i grupiranja se pojavljuju naizmjenično više puta prije nego što se dođe do dijela klasifikacije. Dio izvlačenja značajki reducira slike u oblike s kojima je znatno lakše raditi, ali to radi na način bez da gubi glavne značajke same slike koje su ključne za dobre predikcije. Izlazni čvorovi pojedinog sloja u dijelu izvlačenja značajki su grupirani u dvodimenzionalne ravnine, čiji se postupak naziva mapiranje značajki (Alom et al., 2019).

5.2.1. Konvolucijski sloj

Konvolucija se definira kao matematička operacija koja opisuje način na koji će se spojiti dva skupa informacija. Kod konvolucijskih neuronskih mreža konvolucija se odnosi na otkrivanje značajki slike koja se obrađuje (Patterson & Gibson, 2019).

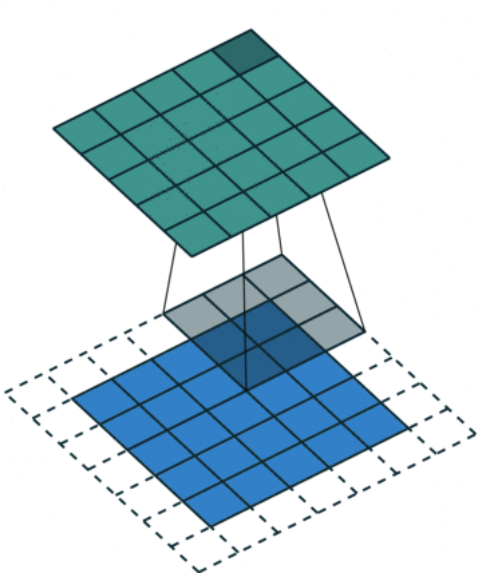


Slika 21. Izvršavanje konvulacije uz pomoć filtera (Patterson & Gibson, 2019)

Na prethodnoj slici zeleni pravokutnik predstavlja ulaz slike u obliku 5x5 tablice. Dio koji je zadužen za izvršavanje operacije konvulacije se naziva **filter**. Filter se primjenjuje na ulazni skup podataka na način da se ograničeni dio dimenzije filtera uzima na ulaznom skupu podataka i izvršava se matrično množenje. Veći rezultat množenja ukazuje to da se pronašla tražena značajka u određenom okviru. U ovom slučaju filter je dimenzije 3x3 i adekvatno tome se uzima dio iz ulaznog skupa podataka dimenzija 3x3. Nakon što se dobije vrijednost množenja, filter se pomiče desno za unaprijed definirani **korak** *(eng. stride)*. Filter se pomiče na ovaj način i izvršava se množenje sve dok se ne dođe do kraja s desne strane, nakon čega se filter vraća skroz lijevo i pomiče prema dolje, opet za definirani korak. Ovaj postupak se izvršava dok god nisu pokrivena sva polja u ulaznom skupu podataka. Uobičajeno su prvi slojevi zaduženi za otkrivanje osnovnih značajki kao što su rubovi i boje, dok su kasnije dodani slojevi zaduženi za otkrivanje kompleksnijih značajki.

Kod postupka konvulacije moguće je da rezultat konvulacije ima reducirane dimenzije u odnosu na ulaz, ali je isto tako moguće da se nastoji održati veličinu dimenzija ili čak povećati

istu. Ovo se izvršava uz pomoć dodavanja takozvanih **obloga** *(eng. padding)*. Obloge su naknadno dodana polja na rubovima tablice koja se obrađuje kako bi se povećale dimenzije same tablice kako bi nakon primjene filtera imali izlaz željenih dimenzija. Dodana polja se inicijaliziraju na vrijednost nula kako ne bi utjecali na pronalazak traženih značajki.

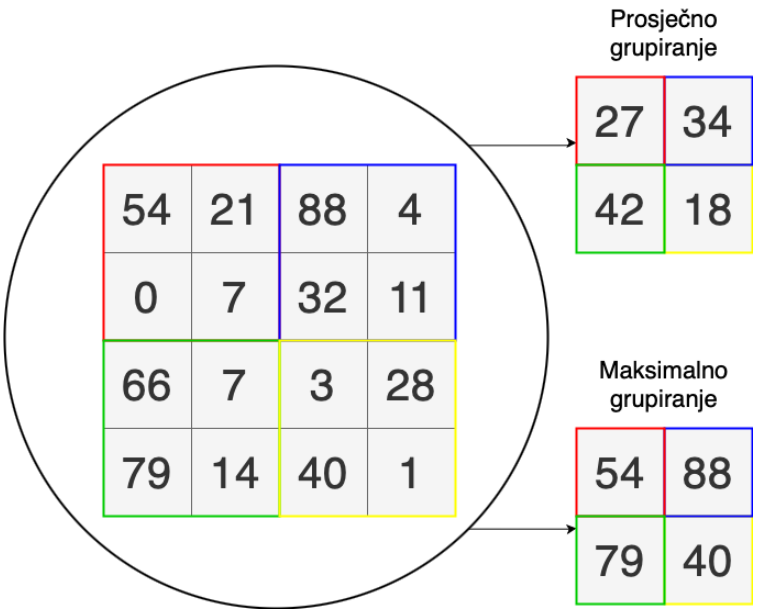


Slika 22. Vizualni prikaz dodavanja obloge na ulazni skup podataka (MathWorks, 2021)

5.2.2. Sloj grupiranja

Sloj grupiranja *(eng. pooling layer)* također obrađuje ulazni skup podataka na način da nakon obrade, podaci s kojima se nastavlja raditi imaju znatno manju zahtjevnost u pogledu hardvera i vremena koje je potrebno za obradu. Također omogućava izvlačenje dominantnih značajki (Alom et al., 2019).

Razlikuju se dvije vrste grupiranja. Češće grupiranje je maksimalno grupiranje *(eng. max pooling)*. Radi se o trivijalnom postupku gdje se ponovno primjenjuje filter određenih dimenzija, odnosno gleda se ograničeno područje ulaznog skupa podataka i uzima se njegova maksimalna vrijednost. Filter se pomiče na isti način kao i u konvulacijskom sloju. Drugi način grupiranja je prosječno grupiranje *(eng. average pooling)*. Jedina razlika u odnosu na maksimalno grupiranje je to što se ne uzima maksimalna vrijednost pokrivena filterom, već se uzima prosječna vrijednost svih polja (Patterson & Gibson, 2019). Maksimalno grupiranje daje bolje rezultate iz razloga što je maksimalno grupiranje također efektivno u suzbijanju raznih nepoželjnih šumova i kao takav se preferira u odnosu na prosječno grupiranje (Alom et al., 2019).



Slika 23. Tipovi grupiranja u konvolucijskim neuronskim mrežama

5.2.3. Sloj potpuno spojene mreže

Nakon što je ulazna slika pretvorena u tip podataka koji je prikladan za obradu u prethodnim slojevima, dolazi se do posljednjeg sloja, a to je sloj potpuno spojene mreže *(eng. fully connected layer)*. Ovdje se prethodno dobiveni rezultati, odnosno izdvojene značajke konvertiraju u jednodimenzionalan vektor koji se prosljeđuje kroz potpuno spojeni sloj. Ovaj potpuno spojeni sloj obrađuje prethodno navedeni vektor i sposoban je prepoznati određene dominatne značajke kao i značajke niskog reda. Na kraju sloj potpuno spojene mreže koristi softmax klasifikacijsku tehniku koja predstavlja kategoričku raspodjelu vjerojatnosti, čija je suma kategoričkih vjerojatnosti jednaka 1 (Alom et al., 2019).

5.2.4. Popularne arhitekture konvolucijskih neuronskih mreža

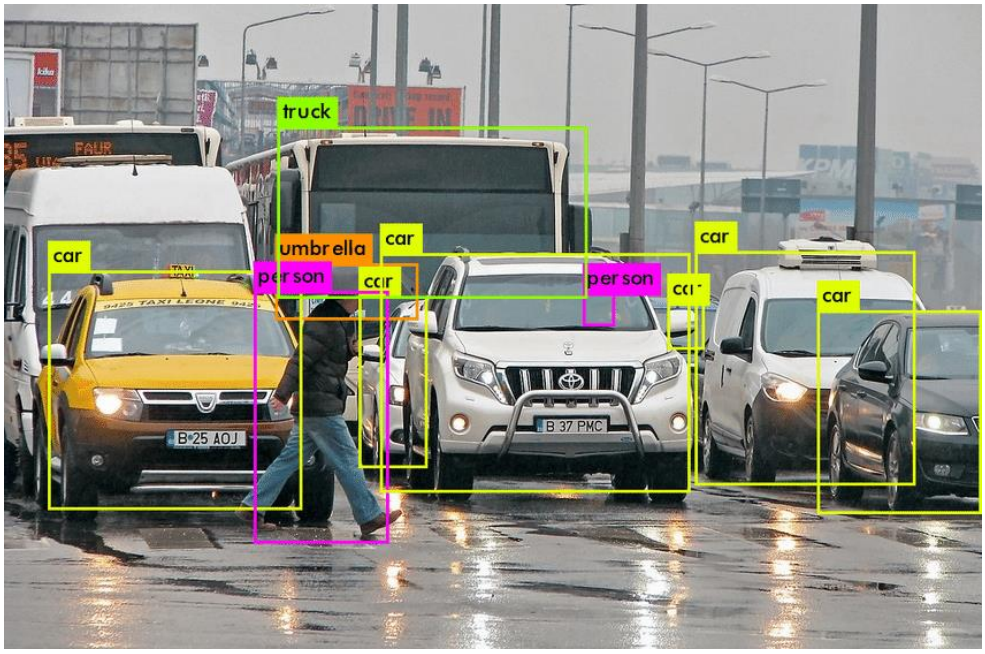
Postoji mnogo različitih arhitektura konvolucijskih neuronskih mreža. Neke od najznačajnijih su sljedeće:

- LeNet
- AlexNet
- ZF Net
- GoogLeNet
- VGGNet
- ResNet

5.2.5. Primjena konvolucijskih neuronskih mreža

Ovaj odjeljak sadrži neke od najčešćih primjena konvolucijskih neuronskih mreža (Alom et al., 2019; Patterson & Gibson, 2019).

- Rješavanje problema s grafovima: automatsko učenje podatkovnih struktura temeljenih na grafovima s raznim primjenama u području rudarenja podataka
- Procesiranje slika i računalni vid *(eng. computer vision)*: navedena primjena ima široku obradu u procesiranju silka. Jedna od primjena je klasifikacija. Klasifikacija se odnosi na predviđanje pripada li objekt određenoj klasi, odnosno oznaci ili ne pripada. Jedan od najvažnijih primjera klasifikacije u računalnom vidu je primjena u medicini, a to je postupak uočavanja stanica raka u ranoj fazi na temelju slika magnetske rezonance. Druga primjena je detekcija objekata. Detekcija objekata se razlikuje u odnosu na klasifikaciju na način da ne rezultira u postotku vjerojatnosti pripada li objekt određenoj klasi, nego rezultira pravokutnim okvirom koji označava traženi objekt na slici. Detekcija rezultira jednim ili većim brojem pravokutnika koji su definirani koordinatama i dimenzijama na način da dani pravokutnici okružuju traženi objekt ili više objekata. Sljedeći primjer je semantička segmentacija. Semantička segmentacija je korak dalje u odnosu na detekciju objekata. Zadatak semantičke segmentacije je sličan detekciji objekata, ali semantička segmentacija to radi znatno preciznije. Ne koriste se pravokutnici koji ukazuju na poziciju traženog objekta, nego se objekti odvajaju na razini pojedinih piksela. Svakoj od pojedine klase objekata se dodjeljuje distinktivna boja koja prekriva cijeli objekt na razini piksela (Szeliski, 2010).
- Procesiranje govora: procesiranje govora se koristi u svrhe poboljšanja govora u smislu uklanjanja šumova i isticanja govora.



Slika 24. Detekcija objekata (Potdar, 2021)

5.3. Duboko generativno modeliranje

Prethodno navedene ponavljajuće neuronske mreže i konvulacijske neuronske mreže pripadaju kategoriji nadziranog učenja, gdje modelu dubokog učenja dajemo podatke i odgovarajuće oznake za navedene podatke. Njihov cilj je mapirati odgovarajuću oznaku ulaznim podacima, s tim da oznaka može biti klasa ili kontinuirana vrijednost. S druge strane duboko generativno modeliranje pripada nenadziranom učenju, gdje model dubokog učenja dobiva samo podatke bez oznaka i sam je zadužen za otkrivanje uzoraka, odnosno struktura danih podataka.

Duboko generativno modeliranje ima za cilj naučiti distribucije skupa podataka za treniranje na način da uz pomoć naučenog postoji mogućnost generiranja novih podataka na temelju onih u fazi treniranja, ali s određenim varijacijama (Doersch, 2016).

Dva su pristupa najviše korištena u području dubokog generativnog modeliranja, a to su **varijacijski autoenkoderi** *(eng. variational autoencoders)* i **generativne kontradiktorne mreže** *(eng. generative adversial networks)*.

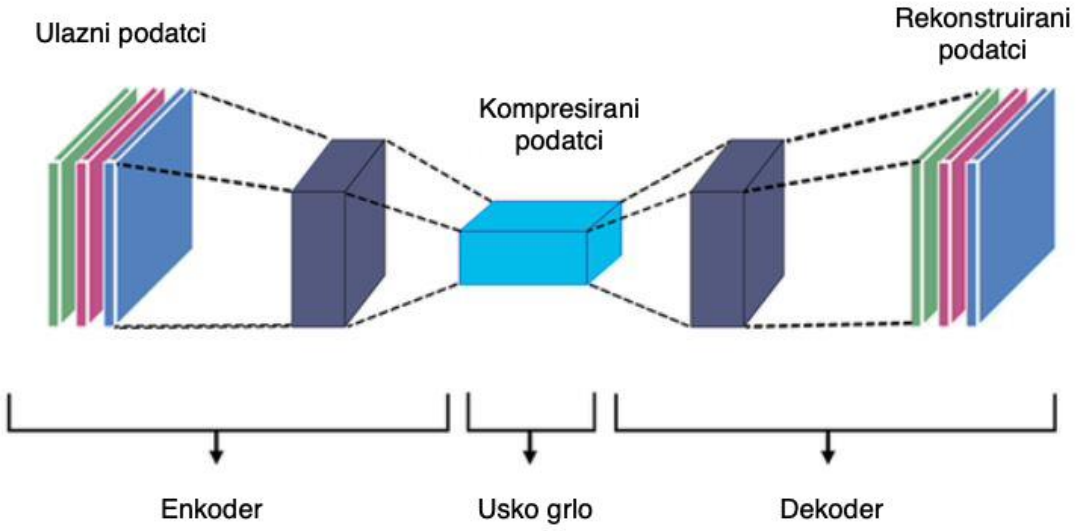
5.3.1. Autoenkoderi

Kako bi lakše razumjeli varijacijske autoenkodere potrebno je poznavati osnove rada običnih autoenkodera. Svrha autoenkodera je naučiti kako iterativnim optimizacijskim postupkom efikasno komprimirati podatke i iskoristiti navedene komprimirane podatke kako bi

se rekonstruiralo stanje podataka što je sličnije moguće početnom stanju. Cilj autoenkodera je smanjiti dimenzije podataka, ali smanjiti dimenzije podataka na način da se sačuva što veći dio informacija o samoj strukturi podataka u reduciranim reprezentacijama (Bank, Koenigstein, & Giryes, 2020).

Autoenkoderi se sastoje od četiri glavna dijela:

- Enkoder: prvi dio autenkodera u kojemu model uči na koji način reducirati dimenzije ulaznog skupa podataka i kako komprimirati podatke tako da ih kasnije bude moguće dekomprimirati.
- Komprimirani podaci: drugi dio autoenkodera su komprimirani podaci koji su rezultat prolaska kroz enkoder. Ovaj sloj se također naziva enkodirani sloj ili latentan sloj.
- Dekoder: posljednji dio u kojem model uči kako rekonstruirati komprimirane podatke tako da krajnji rezultat bude što bliži ulaznom skupu podataka.
- Gubitak rekonstrukcije: metoda koja daje uvid u to koliko je blizu rezultat dekodera u odnosu na originalan skup ulaznih podataka.



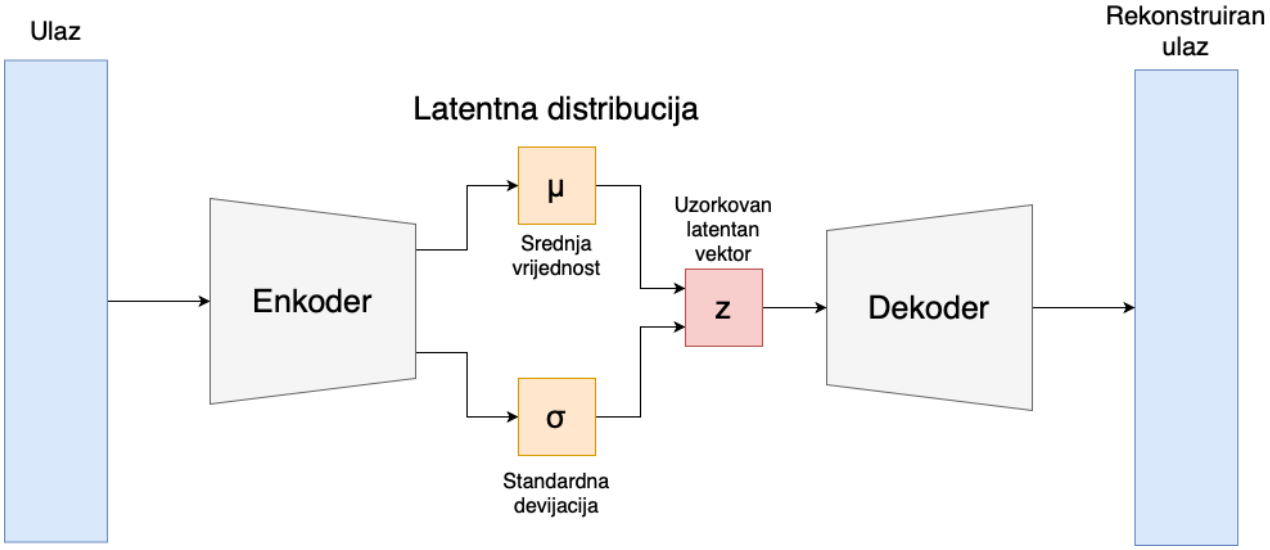
Slika 25. Osnovna struktura autoenkodera (Khosla, 2021)

5.3.2. Varijacijski autoenkoderi

Obični autoenkoderi imaju određena ograničenja. Jednom kad model autoenkodera uspješno završi fazu treniranja on je sposoban rekreirati ulazne podatke, ali i dalje nije moguće

kreirate nove podatke na temelju ulaznih. Ovdje dolaze varijacijski autoenkoderi koji omogućavaju upravo navedenu funkcionalnost.

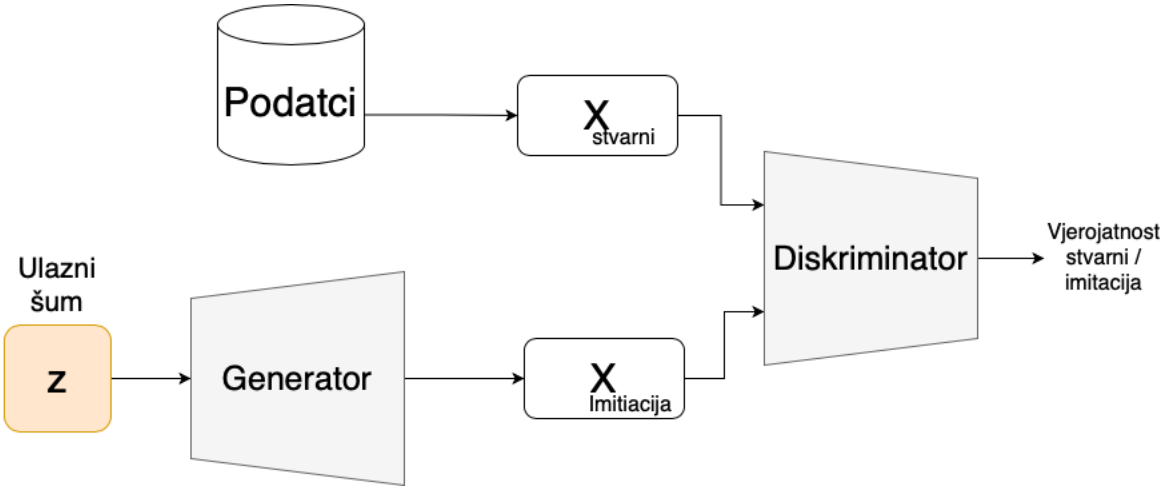
Glavna razlika između običnog autoenkodera i varijacijskog enkodera je u tome što je obični autenkoderi mapiraju ulazni skup podataka u fiksni vektor veličine n, dok varijacijski enkoder mapira ulazni skup podataka u distribuciju. Latentni sloj koji kod običnog autoenkodera sadržava jedan vektor je zamijenjen u slučaju varijacijskog autoenkodera s dva vektora. Jedan od vektora predstavlja srednju vrijednost distribucije (μ), a drugi predstavlja njenu standardnu devijaciju (σ). Ulazni vektor za dekoder se dobiva uzorkovanjem iz distribucije (Doersch, 2016).



Slika 26. Struktura varijacijskog autoenkodera (Ardi, 2020)

5.3.3. Generativne kontradiktorne mreže

Naziv generativnih kontradiktornih mreža *(eng. generative adversarial networks)* se temelji na njihovoj strukturi. Radi se o strukturi koja sadrži dvije neuronske mreže koje su međusobno kontradiktorne, odnosno međusobno su suprotstavljene. Jedna neuronska mreža ima ulogu generatora, a druga diskriminatora. Generator je zadužen za pretvaranje šuma u imitaciju konkretnih podataka. Uloga diskriminatora je zaprimiti imitaciju podataka koje je proizveo generator i zaprimiti stvarni skup podataka i nakon što je zaprimio ulaze iz oba izvora, zadužen je za razlikovanje imitiranih podataka od stvarnih. Generator i diskriminator se međusobno suprotstavljaju i ujedno ovise jedan o drugome. Tijekom treniranja modela diskriminator će postajati sve bolji i bolji u razlikovanju imitiranih od stvarnih podataka i kao takav će prisiliti generatora da generira sve bolje i bolje imitacije (Alom et al., 2019).



Slika 27. Struktura generativnih kontradiktornih mreža (Alom et al., 2019)

Na prethodnoj slici vidljiva je unutarnja struktura generativnih kontradiktornih mreža. U postupku treniranja ovakvog modela generator uzima šum koji koristi za generiranje imitacije podataka s kojima nastoji zavarati diskriminatora. Diskriminator, s druge strane, prima stvarnu instancu podataka kao i generiranu imitaciju i vraća vrijednost koja predstavlja kolika je vjerojatnost da je određeni ulaz imitacija. Za potrebe treniranja potrebno je utvrditi funkciju gubitka koja definira kontradiktorne ciljeve za generatora i diskriminatora, kao i globalni optimum.

$$\mathbb{E} = očekivana\ vrijednost$$

$$G = funkcija\ generatora$$

$$D = funkcija\ diskriminatora$$

$$D\big(G(z)\big) = diskriminatorova\ procjena\ vjerojatnosti\ je\ li\ generirana\ imitacija\ imitacija$$

$$D(x) = diskriminatorova\ procjena\ vjerojatnosti\ je\ li\ stvarna\ instanca\ imitacija$$

(Goodfellow et al., 2014)

Funkcija gubitka za diskriminator:

$$\arg max_d\, \mathbb{E}_{z,x}[\log D(x) + \log\,(1 - D\big(G(z)\big))]$$

Funkcija gubitka za generator:

$$\arg min_g\, \mathbb{E}_{z,x}[\log D(x) + \log\,(1 - D\big(G(z)\big))]$$

5.3.4. Primjena dubokog generativnog modeliranja

Duboko generativno modeliranje se prvenstveno koristi za generiranje raznog sadržaja temeljenog na stvarnim pojavama. Neke od najčešćih primjena su prikazane u nastavku (Alom et al., 2019).

- Procesiranje i generiranje slika i video zapisa: generativne kontradiktorne mreže se koriste za generiranje fotorealističnih slika i videozapisa. Također se koriste za generiranje slike na temelju ulaznog teksta.
- Procesiranje i generiranje zvuka i govora: generativne kontradiktorne mreže se koriste za generiranje zvuka, ali se koriste i za poboljšanje kvalitete govora.
- Medicinske svrhe: generativne kontradiktorne mreže se koriste za poboljšanje medicinskih slika putem uklanjanja šumova i nepravilnosti. Generativne kontradiktorne mreže se također koriste za segmentiranje moždanih tumora.

5.4. Podržano učenje

U prethodnim poglavljima je bilo govora o nadziranom i nenadziranom učenju, ali podržano učenje *(eng. reinforced learning)* se razlikuje od prethodno navedenih vrsta učenja i čini vlastitu kategoriju. Podržano učenje je tip učenja u kojemu postoji agent koji je smješten u okolinu određene kompleksnosti. Agent svojim akcijama djeluje u okolini od koje prima povratnu informaciju vezanu za njegovu akciju. Agent prepoznaje situacije koje se pojavljuju u okolini i nastoji pridružiti situaciji odgovarajuću akciju. Agent također mora imati određene ciljeve vezane za samu situaciju u okolini. Agent kao posljedicu za izvršene akcije od sustava prima povratnu informaciju s obzirom na njegov cilj. Povratna informacija okoline može biti pozitivna ili negativna, a agent nastoji maksimizirati pozitivne povratne informacije i prema tome prilagođava svoje akcije, odnosno svoje djelovanje (Sutton & Barto, 2015).

Slika 28. Sustav podržanog učenja (Alom et al., 2019)

Povratne informacije koje agent dobiva od okoline se još nazivaju nagradom. Agent dobiva nagrade tijekom vremenskog perioda i agent je podešen na način da vrednuje više nagrade koje su dobivene trenutak nakon akcije u odnosu na nagrade koje su dobivene nakon dužeg vremenskog perioda.

5.4.1. Q učenje

Kod podržanog učenja važno je navesti takozvano Q učenje, odnosno Q funkciju. Radi se o funkciji koja računa kvalitetu kombinacije akcije i stanja, odnosno možemo reći da nam daje uvid u to koliko će biti uspješna pojedina akcija za određeno stanje. Q učenje se definira kao pristup koji pronalazi optimalan smjer odabira akcija za svaki Markovljev proces odlučivanja. Markovljev proces odlučivanja je matematički okvir koji se koristi za modeliranje sustava koji rade sa stanjima, akcijama i nagradama. Temeljna jednadžba za Q-učenje se naziva Bellmanova jednadžba i prikazana je u nastavku (Alom et al., 2019).

$$Q(s,a) = r(s,a) + \gamma\, max_{a'}(Q(s',a'))$$

$$s = stanje$$

$$s' = stanje\ nakon\ akcije\ a$$

$$a = akcija\ u\ stanju\ s$$

$$a' = akcija\ u\ stanju\ s'$$

$$Q = kvaliteta\ primjenjene\ akcije\ na\ dano\ stanje$$

$$r(s,a) = trenutna\ nagrada\ s\ obzirom\ na\ stanje\ s\ i\ akciju\ a$$

$$\gamma = relativna\ vrijednost\ vremenskog\ odmaka$$

Odabir akcije se temelji na sljedećoj jednadžbi:

$$\pi(s) = argmax_a Q(s,a)$$

Svako stanje ima pridruženu takozvanu Q vrijednost. Nakon izvršavanja akcije u određenom stanju dobiva se povratna informacija ili nagrada koja se kasnije koristi kako bi se ažurirala Q vrijednost za navedeno stanje. Ovo je funkcija koja na temelju ulaznog stanja uzima u obzir sve moguće akcije i vraća onu koja rezultira najvećom Q vrijednosti, odnosno onu koja daje najbolji rezultat za dano stanje. Radi se o pronalasku akcije koja maksimizira Q vrijednost.

5.4.2. Primjena podržanog učenja

Podržano učenje ima iznimno veliki potencijal u primjeni u stvarnom svijetu, iako je trenutno stanje daleko od punog potencijala, primjena je raznolika, a u nastavku su navedene neke od najčešćih primjena na temelju rada Yuxi Lia (Li, 2019).

- Sustavi preporuke *(eng. recommender systems)*: sustavi preporuke su sposobni preporučiti proizvode, usluge, informacije i slično s obzirom na osobne preference korisnika. Konkretno se koristi za preporuku filmova, glazbe, vijesti i restorana od strane velikih svijetskih kompanija kao što su Netflix, Spotify i slično.
- Umjetna inteligencija u igrama: podržano učenje se koristi u računalnim igrama kako bi se ostvarile ljudske i nadljudske sposobnosti u navedenoj igri. Cilj je omogućiti stvarnim igračima osjećaj tijekom igranja igre kao da igraju protiv stvarne osobe. Već postoje iznimno snažni agenti za igre kao što su šah i go. Najpoznatiji agenti nose nazive AlphaGo i AlphaGo Zero i sposobni su igrati igre šaha i go na nadljudskim razinama. Najnapredniji agent ovakve prirode se naziva AlphaStar, a igra u kojoj djeluje se naziva StarCraft 2 i znatno je kompleksnija od prethodno navedenih igara kao što su šah i go (AlphaStarTeam, 2019).
- Robotika: podržano učenje se koristi za više namjenu u području robotike. Neki od najčešćih primjera korištenja podržanog učenja su pokretanje pomičnih dijelova, pozicioniranje i stabiliziranje robota. Još jedan način primjene su postupci imitacije u kojima robot nastoji imitirati ponašanje ciljanog subjekta. Udaljeno upravljanje robotima je također jedan od načina primjene podržanog učenja u robotici, a temelji se na detaljnom prenošenju pokreta od strane korisnika do robota (Kormushev, Calinon, & Caldwell, 2013).
- Medicina: postoji više primjena podržanog učenja u medicini, ali najvažnija se naziva dinamičan režim tretiranja *(eng. Dynamic treatment regimes)*, također poznat kao prilagodljiva strategija tretiranja. Riječ je o problemima u sekvencijalnom donošenju odluka kod tretiranja pacijenata. Sustav je zadužen uzeti u obzir sve poznate parametre slučaja pojedinog pacijenta i predložiti odgovarajući sekvencijalni tretman istog.

6. Primjeri dubokog učenja u Pythonu

U ovom poglavlju se nalazi više manjih primjera primjene dubokog učenja. Primjeri su napravljeni na način da se što jasnije prenese konkretna primjena duboka učenja u znanstveno-istraživalačkom pogledu, ali isto tako da se zadrži konkretna primjena u stvarnom svijetu. U nastavku se nalaze prethodno navedeni konkretni primjeri u zasebnim odjeljcima. Za potrebe ovih primjera primarno se koristila dostupna dokumentacija za Keras, koja se nalazi na službenim stranicama (Keras, 2021b).

6.1. Osnovni moduli

Implementiranje dubokog učenja je iznimno kompleksno, ali uz pomoć raznih dostupnih modula znatno je jednostavnije nego što je to prije bio slučaj. Za potrebe ovog rada koristit će se Googleov TensorFlow i Keras. Python kod ćemo pokretati na Google Colab platformi.

TensorFlow je Googleova biblioteka otvorenog koda *(eng. open source)* koja se koristi za numeričke izračune i strojno učenje velikih razmjera. TensorFlow sadrži mnoštvo modela i algoritama koji su prilagođeni strojnom učenju kao i dubokom učenju, a Python je programski jezik koji služi kao sučelje aplikacijskog programiranja za korištenje navedenih algoritama i modela koji se kasnije izvršavaju u C++ programskom jeziku (Google, 2021).

Keras je sučelje aplikacijskog programiranja, namijenjeno za duboko učenje, koje je napisano u programskom jeziku Python, a pokreće se na prethodno navedenoj platformi TensorFlow. Keras je napravljen s namjerom bržeg i lakšeg implementiranja i izvršavanja aplikacija koje se baziraju na dubokom učenju (Keras, 2021a).

6.2. Implementiranje Umjetne neuronske mreže

Prvi primjer primjene dubokog učenja se temelji na umjetnim neuronskim mrežama *(eng. artificial neural networks)*. Radi se o primjeru koji koristi potpuno povezanu neuronsku mrežu kako bi na temelju dostupnih podataka o korisnicima određene banke predvidio mogućnost odlaska danog korisnika.

6.2.1. Uvoz skupa podataka i potrebnih biblioteka

Prije svega želimo omogućiti pristup Google disku s obzirom da smo na njemu smjestili sve potrebne skupove podataka koje ćemo koristiti u ovom radu. Google Colab omogućava ovo pritiskom na jednu tipku koja automatski generira potreban kod za spajanje, nakon kojeg

je kasnije samo potrebno priložiti pristupni kod koji je također generiran od strane Googlea kako bi se omogućio pristup prethodno navedenom disku.

```
[22] from google.colab import drive
     drive.mount('/content/drive')
```

Slika 29. Uvoz Google diska

Sljedeći korak je uvoz potrebnih biblioteka. Za potrebe ovog primjera koristit ćemo biblioteke prikazane na slici u nastavku.

```
[23] import numpy as np
     import pandas as pd
     import tensorflow as tf
```

Slika 30. Uvoz potrebnih biblioteka

Ovime smo kompletirali postupak uvoza potrebnog skupa podataka i potrebnih biblioteka.

6.2.2. Pripremanje podataka

Kako bi pripremili podatke za daljnju obradu prije svega je potrebno učitati iste u memoriju. Ovo izvršavamo pomoću biblioteke pandas koja nam uvelike olakšava rad s datotekama čiji su podaci odvojeni zarezima *(eng. comma separated values file)*. Na sljedećim slikama se nalaze prikazi navedene datoteke s podacima koji se koriste u ovom primjeru.

A	B	C	D	E	F	G
Number	CustomerId	Surname	CreditScore	Geography	Gender	Age
1	15634602	Hargrave	619	France	Female	42
2	15647311	Hill	608	Spain	Female	41
3	15619304	Onio	502	France	Female	42
4	15701354	Boni	699	France	Female	39
5	15737888	Mitchell	850	Spain	Female	43
6	15574012	Chu	645	Spain	Male	44
7	15592531	Bartlett	822	France	Male	50
8	15656148	Obinna	376	Germany	Female	29
9	15792365	He	501	France	Male	44
10	15592389	H?	684	France	Male	27
11	15767821	Bearce	528	France	Male	31
12	15737173	Andrews	497	Spain	Male	24
13	15632264	Kay	476	France	Female	34

Slika 31. Skup podataka korisnika banke dio 1.

H	I	J	K	L	M	N
Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
2	0	1	1	1	101348.88	
1	83807.86	1	0	1	112542.58	0
8	159660.8	3	1	0	113931.57	1
1	0	2	0	0	93826.63	0
2	125510.82	1	1	1	79084.1	0
8	113755.78	2	1	0	149756.71	1
7	0	2	1	1	10062.8	0
4	115046.74	4	1	0	119346.88	1
4	142051.07	2	0	1	74940.5	0
2	134603.88	1	1	1	71725.73	0
6	102016.72	2	0	0	80181.12	0
3	0	2	1	0	76390.01	0
10	0	2	1	0	26260.98	0

Slika 32. Skup podataka korisnika banke dio 2.

U idućem koraku je potrebno učitati skup podataka u njegovoj cijelosti. Prije nego krenemo s učitavanjem podataka potrebno je razmisliti o svim atributima koji se nalaze u prethodnom skupu podataka i potrebno je procijeniti jesu li svi navedeni atributi korisni u postupku određivanja šanse odlaska iz banke ili ne. Možemo lako zaključiti kako prva tri stupca navedenog skupa podataka ne predstavljaju nikakav značaj u krajnjoj predikciji šanse ostanka i s obzirom na navedeno tako i učitavamo podatke.

```
[25] dataset = pd.read_csv('/content/drive/MyDrive/FOI/Diplomski/ANN/korisnici_banke.csv')
     X = dataset.iloc[:, 3:-1].values
     y = dataset.iloc[:, -1].values
```

Slika 33. Učitavanje seta podataka s izostavljanjem vrijednosti nebitnih atributa

Vrijednost x predstavlja skup vrijednosti koji će se koristiti za dobivanje predikcije, a vrijednost y predstavlja traženu predikciju, odnosno varijablu koju nastojimo predvidjeti.

Nakon uspješnog učitavanja podataka potrebno je još dodatno obraditi tekstualne podatke kako bi model dubokog učenja mogao uspješno raditi s istima. Kako je i prije navedeno u ovom radu tekstualni podaci moraju biti pretvoreni u numeričke kako bi mogli uspješno biti korišteni u modelu dubokog učenja. U našem primjeru imamo dva atributa navedena u tekstualnom obliku, a to su spol i država stanovanja. Prvo ćemo enkodirati značajke vezane za spol na način da ćemo s 0 označavati ženski spol a s 1 muški.

```
[28] from sklearn.preprocessing import LabelEncoder
     le = LabelEncoder()
     X[:, 2] = le.fit_transform(X[:, 2])
```

Slika 34. Enkodiranje spolova u brojčane vrijednosti

Nakon toga enkodiramo državu stanovanja u matricu. Ovo izvršavamo uz pomoć takozvanog *OneHot* enkodiranja. Prethodno navedeni način enkodiranja uzima sve jedinstvene vrijednosti iz pojedine kategorije i dodjeljuje im jedinstvenu matricu brojeva, u slučaju korištenja Python programskog jezika dodjeljuje im se jedinstveno polje brojeva.

```
[30] from sklearn.compose import ColumnTransformer
     from sklearn.preprocessing import OneHotEncoder
     ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
     X = np.array(ct.fit_transform(X))
```

Slika 35. Enkodiranje država u brojčane vrijednosti

Sljedeće što je potrebno napraviti je podijeliti ukupni skup podataka na dva dijela, a to je skup za treniranje i skup za testiranje. Skup za testiranje će nam biti 20 posto ukupnog skupa, a ostalih 80 posto će biti namijenjeno treniranju modela.

```
[32] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Slika 36. Podijela ukupnog skupa podataka na dio za treniranje i dio za testiranje

Zadnji korak kod pripreme podataka je skaliranje značajki skupa podataka, koje se još naziva standardizacija. Radi se o postupku normalizacije podataka unutar određenog raspona. Postupak također rezultira u bržem izvođenju algoritama dubokog učenja.

```
[33] from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

Slika 37. Standardizacija podataka

6.2.3.Izgradnja umjetne neuronske mreže

Ovo je dio u kojemu kreiramo umjetnu neuronsku mrežu koja će biti zadužena za rješavanje našeg problema. Prvi korak je inicijalizirati umjetnu neuronsku mrežu kao niz povezani slojeva. Za ove potrebe koristimo prethodno navedeni Keras.

```
[34] ann = tf.keras.models.Sequential()
```

Slika 38. Inicijalizacija umjetne neuronske mreže

Nakon ovoga je potrebno dodati ulazni sloj mreže i prvi skriveni sloj. Kako bi dodali prvi skriveni sloj ponovno koristimo Keras koji u sebi sadrži klasu s nazivom Dense koja je sposobna kreirati skrivene slojeve koje možemo proizvoljno dodavati našem prethodno kreiranom objektu umjetne neuronske mreže. Kod poziva konstruktora skrivenog sloja potrebno je definirati broj neurona u pojedinom sloju i pripadajuće aktivacijske funkcije. Ulazni sloj će sadržavati broj neurona jednak broju atributa pripremljenog skupa podataka, ali kod definiranja broja neurona nažalost ne postoji konkretan način određivanja optimalnog broja neurona, radi se o postupku pokušaja i pogreške kojim se nastoji doći do što boljeg rezultata. Za potrebe ovog primjera odlučili smo se za 6 neurona. Kao aktivacijsku funkciju koristimo funkciju ispravljene linearne jedinice *(eng. rectified linear function)*, takozvanu relu funkciju.

```
[35] ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Slika 39. Dodavanje skrivenog sloja objektu umjetne neuronske mreže

Nakon ovoga dodajemo na isti način još jedan, drugi po redu, skriveni sloj i na kraju dodajemo izlazni sloj, koji u našem slučaju ima samo jedan izlaz s obzirom da se radi o predikciji ostaje li korisnik u banci ili ne. Kao aktivacijsku funkciju koristimo sigmoidnu funkciju koja nam omogućava dobivanje vrijednosti između 0 i 1 koja će nam odmah dati vjerojatnost odlaska pojedinog korisnika iz banke.

```
[37] ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Slika 40. Dodavanje izlaznog sloja objektu umjetne neuronske mreže

Ovom naredbom smo završili dio izgradnje umjetne neuronske mreže i možemo prijeći na sljedeći dio, a to je treniranje upravo izgrađene umjetne neuronske mreže

6.2.4. Treniranje umjetne neuronske mreže

Prvi korak u treniranju umjetne neuronske mreže je kompiliranje do sada izgrađene umjetne neuronske mreže. Kod kompiliranja moramo odrediti tri parametra koja su važna za sam postupak kompiliranja. Tri parametra su optimizator, funkcija gubitka i željene izlazne metrike. Za optimizator odabiremo optimizator adam, za funkciju gubitka odabiremo binarnu

unakrsnu entropiju i za izlaznu metriku želimo samo preciznost koja će ukazivati, kako joj samo ime nalaže, preciznost predikcije našeg modela.

```
[38] ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Slika 41. Kompilacija umjetne neuronske mreže

Nakon što uspješno kompiliramo objekt umjetne neuronske mreže potrebno je pokrenuti treniranje modela na skupu podataka za treniranje. Kod pokretanja treniranja potrebno je odrediti 4 ulazna parametra. Prva dva parametra se odnose na atribute odnosno značajke na temelju kojih se dolazi do predikcije i na samo varijablu koja je zavisna, odnosno na samu varijablu koja se predviđa. Nakon toga određujemo veličinu serije *(eng. batch size)* koja definira koliko ćemo predikcija odjednom uspoređivati s stvarnim rezultatima, a kao standardna vrijednost veličine serije se uzima broj 32. Posljednji parametar je broj epoha. Epohe su broj ponavljanja koje će trening izvršiti. Što je veći broj epoha to će biti bolje krajnje predikcije.

```
[39] ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

Slika 42. Treniranje umjetne neuronske mreže na skupu podataka za trening

```
Epoch 1/100
250/250 [==============================] - 1s 1ms/step - loss: 0.5788 - accuracy: 0.7707
Epoch 2/100
250/250 [==============================] - 0s 1ms/step - loss: 0.5012 - accuracy: 0.7960
Epoch 3/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4710 - accuracy: 0.7960
Epoch 4/100
250/250 [==============================] - 0s 1ms/step - loss: 0.4518 - accuracy: 0.7960
```

Slika 43. Prve četiri epohe treniranja umjetne neuronske mreže

6.2.5. Evaluacija modela i dobivanje predikcija

Nakon što je model uspješno završio period treniranja naša umjetna neuronska mreža je spremna za korištenje. Kako bi dobili predikciju koristimo metodu samog objekta umjetne neuronske mreže, a to je metoda 'predict()'. Potrebno je definirati sve atribute za koje smo prethodno odredili da igraju ulogu u krajnjoj predikciji ostaje li korisnik u banci ili ne. Atribute je potrebno definirati u obliku ulaznih parametara za metodu 'predict()', na način da se tekstualni ulazni podaci unesu u obliku njihovih brojčanih vrijednosti koje smo dobili njihovim enkodiranjem. Također je važno naglasiti da je potrebno primijeniti identično skaliranje na ulazne parametre kao što je bilo primijenjeno u dijelu pripremanja podataka. Kako bi dobili krajnji odgovor, odnosno informaciju odlazi li korisnik ili ne, uspoređujemo naš rezultat

predikcije s 0.5. U slučaju da je veći od 0.5 znači da je vjerojatnije da će napustiti banku i suprotno.

```
print(ann.predict(sc.transform([[1, 0, 0, 700, 1, 45, 8, 100000, 2, 1, 0, 150000]])) > 0.5)
```

Slika 44. Predikcija pojedinačnog rezultata

Za prethodno navedenu predikciju dobili smo rezultat manji od 0.5 što znači da korisnik s navedenim parametrima neće napustiti banku.

Nakon predikcije još jednu stvar koju ćemo napraviti je to da ćemo predvidjeti rezultat testnog skupa. A to ćemo napraviti na način kako je prikazano na sljedećoj slici. Ovdje želimo rezultate prikazati u obliku nula i jedinica, što ćemo napraviti na način da dobivenu predikciju, kao i u prethodnoj liniji, usporedimo s 0.5 i zabilježimo taj rezultat umjesto prezicne vjerojatnosti predikcije. U posljednjoj liniji koristimo metode iz numpy biblioteke kako bi prikazali rezultate.

```
[41] y_pred = ann.predict(X_test)
     y_pred = (y_pred > 0.5)
     print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

     [[0 0]
      [0 1]
      [0 0]
      ...
      [0 0]
      [0 0]
      [0 0]]
```

Slika 45. Predviđanje rezultata testnog seta

Na prethodnoj slici možemo vidjeti matricu brojeva od kojih broj s lijeve strane predstavlja predikciju modela, a broj s desne strane stvarnu vrijednost na temelju skupa testnih podataka. Ne mogu biti prikazani svi podaci zbog veličine skupa podataka, ali možemo vidjeti kako se sve vrijednosti poklapaju osim vrijednosti u drugom redu. Možemo zaključiti da smo dobili zadovoljavajuće rezultate.

6.3. Implementiranje konvolucijske neuronske mreže

Drugi primjer u ovom radu se temelji na konvolucijskim neuronskim mrežama. Radi se o primjeru koji koristi konvolucijske neuronske mreže kako bi se stvorio model koji sposoban razlikovati dvije vrste životinja jedne od drugih. U ovom konkretnom slučaju radi se o razlikovanju mačke i psa.

6.3.1. Uvoz skupa podataka i potrebnih biblioteka

Kao i u prethodnom primjeru potrebno je omogućiti pristup do Google diska iz razloga što je tamo pohranjen skup podataka koji će se koristiti. Isto tako se ponovno uvoze potrebne biblioteke za daljnji rad na primjeru. U ovom slučaju nam je potrebna biblioteka za rad sa slikama, a dio je Kerasa.

```
[3]  import tensorflow as tf
     from keras.preprocessing.image import ImageDataGenerator
```

Slika 46. Uvoz potrebnih biblioteka

Nakon uspješnog uvoza potrebnih biblioteka nastavljamo slično kao i u prethodnom primjeru, a to je rad na pripremi podataka.

6.3.2. Pripremanje podataka

Pripremanje podatak u ovom dijelu primjera ćemo podijeliti na dva dijela. Prvo ćemo pripremiti podatke za treniranje, a onda podatke za testiranje, s manjim međusobnim razlikama koje će biti objašnjene u nastavku. Podaci s kojima radimo su odvojene slika pasa i mačaka. Radimo s 4000 slika pasa i isto toliko slika mačaka u skupu podataka za trening i po 1000 slika za svaku od prethodnih životinja u skupu podataka za testiranje.

Prvo krećemo s pripremanjem podataka za trening. Kod pripremanja navedenog skupa podataka potrebno je napraviti određene transformacije kako bi izbjegli preveliko prilagođavanje *(eng. overfitting)* skupu podataka za trening. U slučaju prevelikog prilagođavanje ćemo imati preveliko preciznost, a na skupu za test premalu i zato moramo napraviti određene transformacije. Transformacije koje primjenjujemo nisu striktno određene, potrebno je samo primijeniti određene transformacije u dostatnim količinama kako ne bi došlo do prevelikog prilagođavanja. Transformacije koje su odabrane za potrebe ovog primjera su 'shear_range', 'zoom_range', 'horizontal_flip'.

```
[5]  train_datagen = ImageDataGenerator(rescale = 1./255,
                                        shear_range = 0.2,
                                        zoom_range = 0.2,
                                        horizontal_flip = True)
     training_set = train_datagen.flow_from_directory('/content/drive/MyDrive/FOI/Diplomski/CNN/skup_podataka_za_trening',
                                                      target_size = (64, 64),
                                                      batch_size = 32,
                                                      class_mode = 'binary')
```

Slika 47. Pripremanje skupa podataka za trening

Na prethodnoj slici se nalazi još jedan parametar koji nije naveden. Radi se o parametru 'rescale' koji je zadužen za skaliranje značajki, postupkom dijeljenja vrijednost svakog piksela, koja je inače između 0 i 255, spušta na vrijednosti između 0 i 1. Na prethodnoj slici možemo

vidjeti nakon što dobijemo objekt 'ImageDataGenerator' izvršavamo još jednu naredbu. Navedena naredba je zadužena za pridruživanje prethodno kreiranog 'ImageDataGenerator' objekta odgovarajućem skupu podataka. Kod učitavanja podataka također određujemo parametre kao što su veličina pojedine slike i veličina pojedinačne serije. U pravilu se uvijek reducira veličina slike kako bi se pogodio dobar omjer preciznosti i brzine treniranja, broj nije striktno definiran i u našem slučaju se radi o dimenziji slike 64x64. Za veličinu serije se uzima standardna preporučena vrijednost, a to je 32.

Za pripremanje skupa podataka za test radimo sve isto kao i kod pripremanja za skup podataka za trening samo bez navedenih transformacija. Želimo da slike za testiranje budu nepromijenjene kako bi dobili dobar uvid u stvarne performanse modela.

```
[6] test_datagen = ImageDataGenerator(rescale = 1./255)
    test_set = test_datagen.flow_from_directory('/content/drive/MyDrive/FOI/Diplomski/CNN/testni_skup_podatak',
                                                target_size = (64, 64),
                                                batch_size = 32,
                                                class_mode = 'binary')
```

Slika 48. Pripremanje skupa podataka za testiranje

6.3.3. Izgradnja konvolucijske neuronske mreže

Ovo je dio u kojem kreiramo konvolucijsku neuronsku mrežu. Ovaj dio će biti poprilično sličan izgradnji umjetne neuronske mreže s obzirom da se također sastoji od više slojeva, ali će u ovom slučaju slojevi biti nešto drugačiji.

Prvi korak je inicijaliziranje konvolucijske neuronske mreže. U ovom slučaju korak je u potpunosti jednak kreiranju umjetne neuronske mreže, zato što se u oba slučaja radi o sekvencijalnom modelu koji se sastoji od više slojeva.

```
[7] cnn = tf.keras.models.Sequential()
```

Slika 49. Inicijaliziranje konvolucijske neuronske mreže

Nakon što je konvolucijska neuronska mreža uspješno inicijalizirana potrebno je dodati sloj konvolucije. Unutar Keras biblioteke pronalazimo klasu 'Conv2D' koja predstavlja konvoluciju za dvodimenzionalne prikaze. Potrebno je također odraditi ulazne parametre. Prvi ulazni parametar određuje broj objekata, odnosno filtera koji će otkrivati značajke slike. Određene arhitekture predlažu vrijednost od 32, tako da ćemo koristiti istu. Drugi parametar je 'kernel_size' i odnosi se na veličinu, odnosno dimenzije prethodno navedenih filtera, u našem slučaju je vrijednost 3 što znači da će se raditi o filteru dimenzija 3x3. Nakon toga potrebno je odrediti aktivacijsku funkciju. Kao i kod prethodne umjetne neuronske mreže i ovdje se

preporuča korištenje takozvane 'relu' aktivacijske funkcije. I s obzirom da je ovo prvi konvolucijski sloj potrebno je definirati još jedan parametar a to je 'input_shape'. Prethodno smo kod učitavanja slika odredili da ćemo im smanjiti veličinu na 64x64 i sukladno tome su nam prve dvije vrijednosti kod posljednjeg parametra jednake 64, dok treća vrijednost predstavlja svojevrsnu dubinu slike koja se zapravo odnosi na RGB sustav prezentiranja boja u kojemu jedan piksel ima tri vrijednosti koje zajedno definiraju jednu boju i zato je posljednja vrijednost u navedenom parametru 3.

```
[8] cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[64, 64, 3]))
```

Slika 50. Dodavanje sloja konvolucije

Sljedeći korak je dodavanje sloja grupiranja *(eng. pooling layer)*. Ponovno koristimo Keras i dohvaćamo klasu 'MaxPool2D'. Kod konstruktora prethodno navedene klase potrebno je definirati dva parametra. Prvi se odnosi na veličinu samog okvira koji će izvršavati grupiranje, a drugi se odnosi na pomak prethodno navedenog okvira. U našem slučaju definiramo okvir dimenzija 2x2 čiji će pomak iznositi 2 mjesta. Ovaj sloj je zadužen za daljnje smanjivanje dimenzija mape izvučenih značajki na način da će se iz navedenog okvira uzimati najveća vrijednost i unositi u novu matricu za svaki pomak.

```
[9] cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Slika 51. Dodavanje sloja za grupiranje

Nakon što smo odradili prethodni dio, dodajemo ponovno konvolucijski sloj i sloj grupiranja. Na identičan način kao i prije, samo što kod dodavanja konvolucijskog sloja ne definiramo parametar 'input_shape' s obzirom da se ne radi više o prvom sloju.

```
[10] cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
     cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Slika 52. Dodavanje drugog konvolucijskog sloja i sloja za grupiranje

Idući korak je iz prethodno definiranih slojeva dobiti jednodimenzionalni vektor vrijednosti koje ćemo moći provući kroz našu potpuno spojenu neuronsku mrežu. Kako bi postigli navedeno samo je potrebno izvršiti naredbu prikazanu u nastavku.

```
[11] cnn.add(tf.keras.layers.Flatten())
```

Slika 53. Dobivanje jednodimenzionalnog vektora na temelju prethodnih slojeva

Sada nakon što imamo jednodimenzionalan vektor, potrebna nam je potpuno spojena neuronska mreža kroz koju ćemo pustiti prethodno navedeni vektor. Ponovno definiramo dva parametra, od kojih se prvi odnosi na broj neurona u pojedinom sloju, a drugi se odnosi na aktivacijsku funkciju. Za broj neurona uzimamo broj 128, prateći prijedloge određenih prethodno definiranih arhitektura, a za aktivacijsku funkciju ponovo odabiremo 'relu'.

```
[12] cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

Slika 54. Dodavanje potpuno spojenog sloja neuronske mreže

Sve što nam na kraju preostaje za dovršavanje izgradnje konvolucijske neuronske mreže je dodati izlazni sloj, koji se sastoji od jednog neurona s obzirom da imamo samo pitanje radi li se o psu ili mački i za aktivacijsku funkciju je odabrana sigmoidalna funkcija, kako bi ponovno dobili konkretnu vrijednost vjerojatnosti s kojom sigurnošću možemo tvrditi da se određena životinja nalazi na slici.

```
[13] cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Slika 55. Dodavanje izlaznog sloja

6.3.4. Treniranje konvolucijske neuronske mreže

Kako bi mogli trenirati neuronsku mrežu potrebno ju je prije sve kompilirati. Kompiliranje konvolucijske neuronske mreže, jednako je kompiliranju umjetne neuronske mreže u prethodnom primjeru, tako da nećemo ponovno ulaziti u detalje.

```
[14] cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Slika 56. Kompiliranje konvolucijske neuronske mreže

Jednom kad smo kompilirali konvolucijsku neuronsku mrežu, spremni smo za idući korak, a to je samo treniranje. Ponovno koristimo metodu 'fit()', koja izvršava treniranje. Kao i za prethodni primjer moram definirati ulazne parametre. Prvi parametar se odnosi na ulazni skup podataka za treniranje, drugi parametar se odnosi na podatke za validaciju, odnosno evaluaciju naših rezultata, a to je u našem slučaju skup podataka za test. Posljednji parametar je broj epoha koji će se izvršavati i u našem slučaju će biti 30 epoha. 30 epoha se pokazalo kao dobar omjer preciznosti i brzine treniranja u mojim pokušajima treniranja.

```
[15] cnn.fit(x = training_set, validation_data = test_set, epochs = 30)
```

Slika 57. Treniranje konvolucijske neuronske mreže

Postupak treniranja će trajati nešto duže u odnosu na treniranje u prethodnom primjeru, ali bez obzira na to, s ovom posljednjom naredbom smo zaključili dio s treniranjem konvolucijske neuronske mreže.

6.3.5. Dobivanje konačne predikcije

Kao posljednji korak u izvršavanju dijela o implementiranju konvolucijskih neuronskih mreža, preostalo nam je samo definirati dobivanje konačne predikcije. Za ove potrebe moramo uvesti dio s nazivom 'image' iz Kerasa, koji ćemo koristiti za učitavanje naše slike za koju želimo dobiti predikciju. Osim što ćemo koristiti za jednostavno učitavanje koristit ćemo i za prilagođavanje same slike kako bi ulazni oblik podataka bio odgovarajući metodi za predviđanje. Uz prethodno navedeni dio iz biblioteke Keras, također će nam biti potrebna biblioteka numpy. Kod učitavanja slike važno je promijeniti veličinu same sliku u istu veličinu koju smo definirali tijekom pripremanja podataka, u našem slučaju to su dimenzije 64x64.

```
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('/content/drive/MyDrive/FOI/Diplomski/CNN/slike_za_evaluaciju/pas_ili_macka_1.jpg',
                            target_size = (64, 64))
```

Slika 58. Učitavanje slike za koju izvršavamo predikciju

Nakon što smo učitali sliku i dodijelili joj odgovarajuće dimenzije, potrebno je pretvoriti sliku u numpy polje, na način kako je to prikazano na slici u nastavku. Ovaj format slike je točno format koji prihvaća metoda za predikciju. Nakon toga je potrebno staviti sliku u seriju. Iako se radi o samo jednoj slici, ovaj postupak je potreban, zato što smo tijekom pripremanja slika definirali serije slika koje će se obrađivati i model koji je kreiran kao rezultat navedenih podataka, kao takav zahtjeva da slike budu u seriji. Ovdje koristimo metodu iz numpy biblioteke s nazivom 'np.exapnd_dims'. Ovo je metoda, kako joj i samo ime nalaže koja proširuje dimenzije određenog numpy polja. U našem slučaju kao parametre definiramo ciljanu sliku obliku polja, kao prvi parametar, a kao drugi parametar definiramo poziciju na koju želimo dodati ciljanu dimenziju. Ponovno, u našem slučaju radi se o prvoj poziciji, s obzirom da svaka slika prvo ima kao prvu dimenziju definiranu seriju.

```
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
```

Slika 59. Pretvaranje slike u numpy polje i dodavanje dimenzije za seriju

Sve što je sada potrebno je pozvati metodu za predikciju, na način da tijekom poziva normaliziramo sliku za koju tražimo predikciju na isti način kao što smo to radili u postupku pripremanja podataka, a to je reduciranje vrijednosti piksela u raspon između 0 i 1. Nakon toga ćemo dobiti rezultat unutar kojega će se nalazit serija koja će sadržavati samo jednu vrijednost, a to će biti naš traženi rezultat. Kako bi znali koja vrijednost predstavlja koju životinju između 1 i 0, pozivamo atribut skupa za treniranje pod nazivom 'class_indices'. Nakon uvida u navedeni atribut možemo zaključiti da sve vrijednosti koje su ispod 0.5, predstavljaju mačku, a sve iznad predstavljaju psa. Naravno što je vrijednost bliža pojedinom ekstremu to je veća pouzdanost u samu predikciju.

```
result = cnn.predict(test_image/255.0)
training_set.class_indices
if result[0][0] < 0.5:
  prediction = 'mačka'
else:
  prediction = 'pas'
```

Slika 60. Određivanje koja se životinja nalazi na slici

Ovim isječkom koda smo završili s dijelom vezanim uz implementiranje konvolucijske neuronske mreže.

6.4. Implementiranje ponavljajućih neuronskih mreža

Posljednji primjer ovoga rada se temelji na ponavljajućim neuronskim mrežama. Cilj ovog primjera je kreirati model koji je sposoban predvidjeti trendove kretanja vrijednosti dionica.

6.4.1. Uvoz potrebnih biblioteka i skupa podataka

Kao i u prethodnim primjerima prije svega je potrebno uvesti sve potrebne biblioteke i uključiti mogućnost pristupa Google disku na kojemu se nalazi potreban skup podataka.

```
[181] import tensorflow as tf
      import numpy as np
      import matplotlib.pyplot as plt
      import pandas as pd
```

Slika 61. Uvoz potrebnih biblioteka

Nakon što smo uspješno uvezli sve potrebne biblioteke, sljedeći korak je učitavanje podataka o dionicama za trening. Kod učitavanja skupa podataka za trening želimo učitati sve redove, ali nas zanima samo vrijednost stupca s naslovom 'open', to postižemo adresiranjem traženog stupca s vrijednostima unutar uglatih zagrada.

```
[76] dataset_train = pd.read_csv('/content/drive/MyDrive/FOI/Diplomski/RNN/AMZNtrain.csv')
     training_set = dataset_train.iloc[:, 1:2].values
```

Slika 62. Učitavanje skupa podataka za trening

Date	Open	High	Low	Close	Adj Close	Volume
2019-01-02	1465.199951	1553.359985	1460.930054	1539.130005	1539.130005	7983100
2019-01-03	1520.01001	1538	1497.109985	1500.280029	1500.280029	6975600
2019-01-04	1530	1594	1518.310059	1575.390015	1575.390015	9182600
2019-01-07	1602.310059	1634.560059	1589.189941	1629.51001	1629.51001	7993200
2019-01-08	1664.689941	1676.609985	1616.609985	1656.579956	1656.579956	8881400
2019-01-09	1652.97998	1667.800049	1641.400024	1659.420044	1659.420044	6348800
2019-01-10	1641.01001	1663.25	1621.619995	1656.219971	1656.219971	6507700
2019-01-11	1640.550049	1660.290039	1636.219971	1640.560059	1640.560059	4686200
2019-01-14	1615	1648.199951	1595.150024	1617.209961	1617.209961	6005900
2019-01-15	1632	1675.160034	1626.01001	1674.560059	1674.560059	5998500
2019-01-16	1684.219971	1705	1675.880005	1683.780029	1683.780029	6366900
2019-01-17	1680	1700.170044	1677.5	1693.219971	1693.219971	4208900
2019-01-18	1712	1716.199951	1691.540039	1696.199951	1696.199951	6020500
2019-01-22	1681	1681.869995	1610.199951	1632.170044	1632.170044	6416800
2019-01-23	1656	1657.430054	1612	1640.02002	1640.02002	5225200
2019-01-24	1641.069946	1657.26001	1631.780029	1654.930054	1654.930054	4089900
2019-01-25	1670.5	1683.47998	1661.609985	1670.569946	1670.569946	4945900
2019-01-28	1643.589966	1645	1614.089966	1637.890015	1637.890015	4837700
2019-01-29	1631.27002	1632.380005	1590.719971	1593.880005	1593.880005	4632800
2019-01-30	1623	1676.949951	1619.680054	1670.430054	1670.430054	5783800
2019-01-31	1692.849976	1736.410034	1679.079956	1718.72998	1718.72998	10910300

Slika 63. Primjer skupa podataka Amazon cijena dionica

6.4.2.Skaliranje značajki

Idući korak je skaliranje značajki. U ovom slučaju koristimo postupak normalizacije. Za ove potrebe uvozimo paket 'MinMaxScaler' iz sklearn biblioteke. Nakon toga koristimo istu metodu kako bi dobili objekt koji je namijenjen za skaliranje i pozivamo njegovu metodu 'fit()' koja nam izvršava postupak normalizacije nad našim skupom podataka za trening. Nakon izvršavanja ovih naredbi sve vrijednosti dionica će biti u rasponu između 0 i 1.

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
```

Slika 64. Skaliranje značajki, postupak normalizacije

6.4.3. Kreiranje potrebne podatkovne strukture za ponavljajuću neuronsku mrežu

Sljedeći korak je kreiranja posebne strukture podatak s određenim brojem takozvanih vremenskih koraka i s jednim pripadajućim izlazom. Ovo znači da će ponavljajuće neuronska mreža u svakom danom trenutku **t** gledati određeni broj prethodnih zapisa i na temelju njih, odnosno trendova koje zaključuje iz njih, rezultirati jednom vrijednosti za **t+1** vremenski trenutak. Kako bi spremali prethodno naveden vrijednosti inicijalizirat ćemo dvije liste od kojih će jedna od njih spremati određen broj vrijednosti prethodnih dana, dok će druga spremati pripadajuću vrijednost za idući dan. Lista 'X_train' će spremati vrijednost određenih broja prethodnih dana, dok će lista 'y_train' spremati pripadajuću vrijednost za sljedeći dan. Nakon toga obje liste pretvarano u numpy polja.

```
[137] X_train = []
      y_train = []
      for i in range(60, 1258):
          X_train.append(training_set_scaled[i-60:i, 0])
          y_train.append(training_set_scaled[i, 0])
      X_train = np.array(X_train)
      y_train = np.array(y_train)
```

Slika 65. Kreiranje podatkovne strukture potrebne za ponavljajuću neuronsku mrežu

Nakon što smo uspješno kreirali potrebnu strukturu, iduće što moramo napraviti je preoblikovati listu koja sadrži prethodno određeni broj podataka iz kojih se iščitava trend. Ovo moramo napraviti kako bi navedena lista odgovarala ulazu ponavljajuće neuronske mreže. Kako bi preoblikovali listu na željeni način koristimo numpy metodu 'reshape()'. Prvi parametar prethodno navedene metode se odnosi na ono što želimo preoblikovati, a to je naravno prethodno navedena lista. Drugi parametar predstavlja novi oblik koji želimo formirati. Ono što mi želimo je dodati još jednu dimenziju našoj listi. Željenu strukturu kreiramo na temelju službene dokumentacije Keras biblioteke. Unutar strukture koja sačinjava drugi parametar, prva vrijednost predstavlja vrijednost serije, odnosno u našem slučaju broj opažanja vrijednosti, druga vrijednost predstavlja broj vremenskih koraka, a to je točno broj stupaca u našoj listi, posljednja vrijednost je broj ulaznih dimenzija, odnosno broj indikatora koji utječu na krajnju vrijednost predikcije, a to je u našem slučaju 1. Ukoliko radimo s više indikatora ovdje je potrebno definirati veći broj, sukladno broju indikatora koji se koriste.

```
[157] X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

Slika 66. Preoblikovanje ulazne liste kako bi je prilagodili ulazu ponavljajuće neuronske mreže

6.4.4.Izgradnja ponavljajuće neuronske mreže

Kao i u prethodnim primjerima, izgradnju ponavljajuće neuronske mreže započinjemo s njenom inicijalizacijom. Koristimo Keras biblioteku kako bi generirali sekvencijalni model koji za koji znamo da se sastoji od više slojeva.

```
[188] rnn = tf.keras.models.Sequential()
```

Slika 67. Inicijalizacija ponavljajuće neuronske mreže

Nakon što smo inicijalizirali ponavljajuću neuronsku mrežu potrebno je dodati joj odgovarajuće slojeve kako bi ista bila funkcionalna. Prvo što dodajemo je prvi sloj dugotrajno kratkotrajne memorije. Prvi parametar je broj neurona i za ove potrebe prema opće prihvaćenim prijedlozima biramo broj od 50 neurona. Drugi parametar je povratni niz i ovdje stavljamo vrijednost 'True' zato što pravimo složenu mrežu i potrebno je definirati vrijednost kao true sve dok namjeravamo dodavati dodatne slojeve dugotrajno kratkotrajne memorije. U slučaju da se radi o zadnjem sloju dugotrajno kratkotrajne memorije ovaj parametar možemo izostaviti s obzirom da je njegova zadana vrijednost 'False', što označava kraj niza. Posljednji parametar koji se unosi je 'input_shape', a odnosi na oblik ulaznog skupa podataka. U našem slučaju možemo se sjetiti da imamo listu s 3 dimenzije, ali u ovom slučaju nije potrebno navesti tri dimenzije, već samo dvije koje se odnose na vremenski korak i broj indikatora. Ovo se radi zato što će se prva dimenzija, koja se odnosi na observacije, automatski uključiti. Nakon što smo dodali sloj dugotrajno kratkotrajne memorije, dodajemo sloj regularizacije ispadanja *(eng. dropout regularization)*. Sloj regularizacije ispadanja prima samo jedan parametar a to je stopa ispadanja neurona. Ovaj parametar se izražava u postotku. U našem slučaju 20% neurona će se ignorirati tijekom svake iteracije treniranja.

```
[189] rnn.add(tf.keras.layers.LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
      rnn.add(tf.keras.layers.Dropout(0.2))
```

Slika 68. Dodavanje prvog sloja dugotrajno kratkotrajne memorije i sloja regularizacije ispadanja

Nakon što smo dodali prvi sloj dugotrajno kratkotrajne memorije i pripadajući sloj regularizacije, potrebno je dodati ista dva sloja još tri puta kako bi ukupno imali 4 para navedenih slojeva. Dodavanje unutarnjih slojeva dugotrajno kratkotrajne memorije razlikuje se od prvog sloja na način što se kod unutarnjih slojeva ne mora definirati parametar 'input_shape'. Također u posljednjem sloju dugotrajno kratkotrajne memorije, kako je prethodno već navedeno, potrebno je postaviti parametar 'return_sequences' na 'False' ili ga ne navoditi s obzirom da je njegova zadana vrijednost 'False'.

```
[190] rnn.add(tf.keras.layers.LSTM(units = 50, return_sequences = True))
      rnn.add(tf.keras.layers.Dropout(0.2))
```

Slika 69. Dodavanje unutarnjih slojeva ponavljajuće neuronske mreže

Posljednji sloj koji je potrebno dodati je izlazni sloj. Izlazni sloj u ovom primjeru je potpuno povezani sloj neuronske mreže s jednim neuronom, čiji broj odgovara broju izlaznih dimenzija.

```
[193] rnn.add(tf.keras.layers.Dense(units = 1))
```

Slika 70. Dodavanje posljednjeg sloja ponavljajuće neuronske mreže

6.4.5. Treniranje ponavljajuće neuronske mreže

Kako bi mogli trenirati ponavljajuću neuronsku mrežu, kao i u ostalim primjerima, prvo je potrebno kompilirati istu. Kompilacija se izvršava na sličan način kao i u prethodnim primjerima. Definiraju se dva parametra, a to su optimizator i funkcija gubitka. Koristimo preporučene postavke, prema Keras dokumentaciji, za ponavljajuću neuronsku mrežu, a to su vrijednosti 'adam' za optimizator i 'mean_squared_error' za funkciju gubitka, funkcija.

```
[194] rnn.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

Slika 71. Kompiliranje ponavljajuće neuronske mreže

Nakon što smo uspješno kompilirali ponavljajuću neuronsku mrežu, potrebno je pokrenuti postupak treniranja. Koristimo, do sada već više puta navedenu, metodu 'fit()'. Metoda prima 4 parametra, od kojih je prvi ulazni skup značajki, drugi predstavlja baznu istinu, odnosno vrijednost s kojom ćemo uspoređivati naše predikcije na temelju ulaznog skupa značajki, treći parametar predstavlja broj epoha koje će se izvršavati i posljednji parametar predstavlja veličinu serije. Broj epoha za potrebe ovog primjera je postavljen na 150, kao što

je prije navedeno, broj epoha nije striktno definiran i podložan je promjeni, ali s obzirom na provedena testiranja pokazao se kao dobar omjer preciznosti predikcije i brzine obrade. Kao i u prethodnim primjerima uzima se standardna preporučena vrijednost veličine serije, a to je 32.

```
[249] rnn.fit(X_train, y_train, epochs = 150, batch_size = 32)
```

Slika 72. Treniranje ponavljajuće neuronske mreže

6.4.6. Dobivanje predikcija i prikazivanje rezultata

Posljednji dio ovog primjera odnosi se na dobivanje predikcija i prikazivanja istih u obliku grafa, kako bi mogli lako uočiti preciznost predikcije u odnosu na stvarne vrijednosti.

Kako bi započeli prethodno navedeni postupak, prije svega učitavamo stvarne vrijednosti dionica Amazona za 1. mjesec 2019. godine. I učitavamo stupac koji nam je potreban za iscrtavanje grafa.

```
[250] dataset_test = pd.read_csv('/content/drive/MyDrive/FOI/Diplomski/RNN/AMZNtest.csv')
      real_stock_price = dataset_test.iloc[:, 1:2].values
```

Slika 73. Učitavanje stvarnih vrijednost Amazonovih dionica

Nakon što smo uspješno učitali podatke potrebno je dohvatiti predikciju. S obzirom da se predikcija vrši za vremenski period **t+1** na temelju prethodnih 60 vremenskih perioda, potrebno je spojiti skup podataka za trening zajedno sa skupom podataka za testiranje, kako bi mogli ispravno pokrenuti dohvaćanje predikcije. Kod spajanja prethodno navedenih skupova podataka važno je obratiti pažnju na to koje ćemo točno skupove podataka spojiti, odnosno u kojoj njihovoj fazi. Ukoliko se odlučimo spojiti skup podataka za treniranje koji je prošao skaliranje, bit ćemo primorani skalirati i skup podataka za test, što će izmijeniti vrijednost samih podataka, a to je nešto što nikako ne želimo. S obzirom na prethodno navedeni razlog spojit ćemo početni skup podataka za treniranje koji nije prošao skaliranje značajki i spojit ćemo ga s skupom podataka za testiranje, na način da ćemo ih vertikalno spojiti uz pomoć ulaznog parametra 'concat' funkcije s nazivom 'axis', gdje vrijednost 0 predstavlja vertikalno spajanje, dok vrijednost 1 predstavlja horizontalno spajanje. Jednom ovako spojene vrijednosti pružit će nam ulazne vrijednosti svake predikcije, to jest 60 prethodnih cijena dionica za svako vrijeme **t** i to su podaci koje ćemo skalirati. Na ovaj način skup podataka za testiranje neće promijeniti svoje vrijednosti. Također je potrebno naglasiti da kod spajanja dva navedena skupa podataka,

potrebno je samo učitati vrijednosti u stupcima s oznakom 'Open', što je vidljivo u primjeru u nastavku.

```
[269] dataset_all = pd.concat((dataset_train['Open'], dataset_test['Open']), axis = 0)
```

Slika 74. Spajanje skupova podataka za treniranje i testiranje

Idući korak je izvući potrebne ulaze. Ono što mi trenutno želimo predvidjeti je cijenu dionica u 1. mjesecu 2019. godine. Za svaki dan nam je potrebno prethodnih 60 financijskih dana. Kako bi dobili prvi financijski dan u 2019. godini potrebno je od ukupno spojenog skupa podataka pod nazivom 'dataset_all' uzeti duljinu i oduzeti je s duljinom skupa za treniranje, s nazivom 'dataset_test'. Na ovaj način smo se pozicionirali na prvi financijski dan 2019. godine. A s obzirom da nam treba 60 prethodnih dana za predikciju, donju granicu željeni ulaznih podatak dobivamo naknadnim oduzimanjem sa 60 od vrijednosti koje smo prethodno dobili. Za gornju granicu uzimamo kraj ukupnog spojenog skupa podataka. Nakon što smo definirali granice pozivamo 'values' kako bi izvukli potrebne vrijednosti i oformili numpy polje.

```
inputs = dataset_all[len(dataset_all) - len(dataset_test) - 60:].values
```

Slika 75. Izvlačenje potrebnih ulaznih podataka i spojenog skupa podataka

Nakon što smo uspješno izdvojili potrebne ulazne podatke potrebno je iste preoblikovati, na sličan način kako smo to već ranije radili. Ponovno koristimo metodu 'reshape()' kako bi dobili prethodne podatke u dvodimenzionalnom polju, ali u jednom stupcu, odnosno u obliku linija koje su poredane u jednom stupcu. Nakon toga je potrebno pretvoriti ulazne podatke u trodimenzionalni oblik koji očekuje neuronska mreža koju smo izgradili. Trodimenzionalni oblik je potreban kod treniranja, ali je isto tako potreban kod predikcija. Skaliranje koje je primjenjujemo na naše ulazne podatke mora biti jednako skaliranju koje se primjenjivalo na skupu podataka za treniranje i zato koristimo metodu 'transform()' istog objekta za skaliranje kojeg smo prethodno koristili u skaliranju skupa podatak za treniranje.

```
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
```

Slika 76. Preoblikovanje i skaliranje ulaznih podataka za predikciju

Nakon što smo odradili prethodni korak potrebno je dovršiti postupak kreiranja trodimenzionalne strukture koju naša neuronska mreža očekuje kod izvršavanja predikcije. Prije svega inicijaliziramo varijablu koja će poprimiti traženu trodimenzionalnu strukturu s nazivom 'X_test'. Nakon toga radimo sličan postupak kao kada smo kreirali strukturu za proces

treniranja naše ponavljajuće neuronske mreže, samo je potrebno napraviti nekoliko izmjena. Definiramo petlju s rasponom od 60 do 80 zato što se moramo vratiti za svaki trenutak koji predviđamo, 60 vremenskih korak u natrag, tako da donja granica mora biti 60, kako ne bi probili okvire liste. S druge strane definiramo gornju granicu s 80 zato što testni skup podataka sadrži samo 20 financijskih dana i ako zbrojimo 60 kao početnu vrijednost s 20 prisutnih financijskih dana dobijemo vrijednost 80. Unutar petlje izvršavamo prikupljanje podataka iz prethodno generiranih ulaznih vrijednosti 'inputs'.

Nakon što su vrijednosti uspješno preuzete potrebno je definirati trodimenzionalni oblik, koji smo prethodno već naveli kako se radi u ovom radu. Prvo prebacujemo 'X_test' skup podataka u numpy polje i nakon toga pozivamo 'reshape()' metodu kako bi dobili željenu trodimenzionalnu strukturu.

```
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

Slika 77. Definiranje trodimenzionalne strukture potrebne za predikciju

Posljednje što moramo napraviti u ovom dijelu je pozvati metodu za predikcije nad našom trodimenzionalnom strukturom. Nakon što smo dobili predikciju potrebno je istu inverzno skalirati kako bi negirali skaliranja s kojim je naš model naučio raditi i ujedno vraćati iste rezultate.

```
predicted_stock_price = rnn.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

Slika 78. Predikcija vrijednosti dionica i inverzno skaliranje

Ovaj primjer ćemo zaključiti s vizualnim prikazom predikcije vrijednosti dionica u odnosu na stvarne vrijednosti. Vizualni prikaz grafa smo dobili uz pomoć matplot biblioteke, a navedeni graf jasno prikazuje sličnost u trendovima rasta i pada vrijednosti stvarnih dionica i predikcija vrijednosti dionica.

```
[288] plt.title('Predviđanje cijena Amazon dionica')
      plt.xlabel('Vrijeme')
      plt.ylabel('Cijena Amazon dionica')
      plt.plot(real_stock_price, color = 'black', label = 'Stvarna cijena Amazon dionica')
      plt.plot(predicted_stock_price, color = 'blue', label = 'Predviđena cijena Amazon dionica')
      plt.legend()
      plt.show()
```

Slika 79. Definiranje grafa za prikaz odnosa stvarnih cijena dionica i predikcija modela

Slika 80. Grafički prikaz odnosa stvarnih cijena dionica i njihovih predikcija

Na prethodnom grafičkom prikazu možemo jasno vidjeti kako naša predikcija uspješno prati trendove rasta i pada krivulje stvarnih cijena dionica. Ovime završavamo s posljednjim primjerom u ovom radu.

7. Zaključak

Cilj ovog rada je bio dati jedan sažet, ali istovremeno dovoljno informativan pregled, relativno novog područja, dubokog učenja. Duboko učenje kao podskup strojnog učenja i ujedno podskup umjetne inteligencije je bazirano na mreži umjetnih neurona koji su raspoređeni u višestrukim slojevima. Prije svega se dao uvid u samo strojno učenje i odnos samog strojnog učenja s dubokim učenjem. Nakon razjašnjenog dijela strojnog učenja, korištenjem metode analize duboko učenje je raščlanjeno na pojedine dijelove koji ga sačinjavaju i isti su bili objašnjeni kako bi se uspio shvatiti način funkcioniranja dubokog učenja. Idući korak je bio objasniti više koncepte dubokog učenja u kojima su predstavljene više arhitekture kao što su: ponavljajuća neuronska mreža, konvolucijska neuronska mreža, generativno adaptivna mreža i podržano učenje. Za svaku od prethodnih arhitektura navedena je i primjena u stvarno životu, gdje se moglo uočiti koliko je zapravo velik utjecaj dubokog učenja iako se nalazi u ranoj fazi razvoja i nije još ni blizu punog potencijala.

U praktičnom dijelu rada se koristio programski jezik Python, biblioteke TensorFlow i Keras i razvojna platforma Google Colab. Cilj praktičnog dijela je bio kroz tri različita primjera približiti konkretno implementiranje dubokog učenja na primjerima iz stvarnog svijeta. Koristeći navedene tehnologije implementirana su tri različita primjera od kojih svaki koristi po jedan viši koncept iz teorijskog dijela. Na ovaj način se pokrilo veće područje dubokog učenja i pružila prilika za jednostavan početak u kretanju s proučavanjem područja dubokog učenja. Postupak uspješnog implementiranja svakog konkretnog primjera dubokog učenja je raščlanjen na više manjih koraka kako bi se čitatelj što lakše snašao u samim primjerima i na kraju bio sam sposoban implementirati sličan oblik aplikacije.

Za kraj je važno još jednom navesti veliki potencijal dubokog učenja, neupitno je da se radi o tehnologiji budućnosti koja će rezultirati u znatnom napretku gotovo svih ljudskih djelatnosti. Već sada možemo vidjeti duboko učenje svugdje oko nas, od autonomnih vozila, sigurnosnih značajki, kao što su prepoznavanje lica pa sve do najvažnije primjene, a to je spašavanje života u području medicine. Sve veću primjenu dubokog učenja možemo povezati sa sve većom generacijom podataka od strane cijelog čovječanstva. Prethodno navedeni podaci su glavni pokretači sve sposobnijih modela dubokog učenja. Kako raste broj dostupnih podataka, tako će i rasti kvaliteta modela dubokog učenja i pripadajuće sposobnosti. Možemo očekivati sve značajnije napretke u bliskoj budućnosti koji će vrlo vjerojatno promijeniti način na koji živimo naše živote i kao takvo područje predstavlja iznimno izazovnu i uzbudljivu opciju u životnoj karijeri.

Popis literature

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., … Asari, V. K. (2019). A State-of-the-Art Survey on Deep Learning Theory and Architectures. *Electronics*, *8*(3), 292. https://doi.org/10.3390/electronics8030292

Alpaydin, E. (2016). *Machine Learning The New AI*. Massachusetts: The MIT Press.

Alpaydin, E. (2020). *Introduction to Machine Learning*. Massachusetts: The MIT Press.

AlphaStarTeam. (2019). AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. Preuzeto 25.08.2021. s Deepmind website: https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii

Amini, A. (2021). Introduction to Deep Learning. Preuzeto 28.08.2021. s MIT website: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Amini, A., & Soleimany, A. (2021). *Deep Sequence Modeling*.

Ardi, M. (2020). Using Variational Autoencoder (VAE) to Generate New Images. Preuzeto 27.08.2021. s https://becominghuman.ai/using-variational-autoencoder-vae-to-generate-new-images-14328877e88d

Bank, D., Koenigstein, N., & Giryes, R. (2020). Autoencoders. *Cornell University*. Preuzeto s https://arxiv.org/pdf/2003.05991.pdf

Brownlee, J. (2016). *Master Machine Learning Algorithms From Scratch: Discover How They Work and Implement Them From Scratch*. Melbourne.

Brownlee, J. (2019a). *Better Deep Learning*.

Brownlee, J. (2019b). Loss and Loss Functions for Training Deep Learning Neural Networks. *Deep Learning Performance*. Preuzeto s https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

Calvert, J. B. (2002). Heaviside, Laplace and the Inversion Integral. Preuzeto 22.07.2021. s http://mysite.du.edu/~jcalvert/math/laplace.htm

Doersch, C. (2016). Tutorial on Variational Autoencoders. *Carnegie Mellon / UC Berkeley*. Preuzeto s https://arxiv.org/pdf/1606.05908.pdf

Educative. (2021). Overfitting and underfitting. Preuzeto 10.08.2021. s https://www.educative.io/edpresso/overfitting-and-underfitting

Gavrilova, Y. (2020). Artificial Intelligence vs. Machine Learning vs. Deep Learning: Essentials. Preuzeto 18.07.2021. s https://serokell.io/blog/ai-ml-dl-difference

Goodfellow, I., & Bengio, Y. (2016). *Deep Learning*. Preuzeto s https://www.deeplearningbook.org

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Bengio, Y. (2014). Generative Adversarial Networks. *Universite de Montreal*. Preuzeto s https://arxiv.org/pdf/1406.2661.pdf

Google. (2021). TensorFlow. Preuzeto 24.07.2021. s https://www.tensorflow.org

Grimson, E. (2016). Introduction to Machine Learning Presentation. Preuzeto 20.07.2021. s https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0002-introduction-to-computational-thinking-and-data-science-fall-2016/lecture-slides-and-files/MIT6_0002F16_lec11.pdf

Grimson, E. (2017). *Introduction to Machine Learning*. Preuzeto s https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0002-introduction-to-computational-thinking-and-data-science-fall-2016/lecture-videos/lecture-11-introduction-to-machine-learning/

Heaton, J. (2015). *Artificial Inteligence for Humans: Volume 3: Deep Learning and Neural Networks*. Heaton Research.

IBM Cloud Education. (2020). Unsupervised Learning. Preuzeto 20.07.2021 s https://www.ibm.com/cloud/learn/unsupervised-learning

Keras. (2021a). About Keras. Preuzeto 24.08.2021. s https://keras.io/about/

Keras. (2021b). Keras API reference. Preuzeto 30.08.2021. s https://keras.io/api/

Khosla, R. (2021). Auto-Encoders for Computer Vision: An Endless world of Possibilities.

Koech, K. E. (2020). Cross-Entropy Loss Function. Preuzeto 24.08.2021. s Towards Data Science website: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

Kormushev, P., Calinon, S., & Caldwell, D. (2013). Reinforcement Learning in Robotics: Applications and Real-World Challenges. *Robotics*, *2*(3), 122–148. https://doi.org/10.3390/robotics2030122

Li, Y. (2019). *Reinforcement Learning Applications*. Preuzeto s http://arxiv.org/abs/1908.06973

MathWorks. (2021). Deep Learning with Images. Preuzeto 28.7.2021. s https://www.mathworks.com/help/deeplearning/deep-learning-with-images.html?s_tid=CRUX_topnav

Mccarthy, J. (2004). *WHAT IS ARTIFICIAL INTELLIGENCE?* Preuzeto s http://www-formal.stanford.edu/jmc/

Pant, A. (2019). Introduction to Machine Learning for Beginners. Preuzeto 19.07.2021. s https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08

Patterson, J., & Gibson, A. (2019). Deep learning. A Practitioner's Approach. In *O'Reilly Media, Inc.*

Phung, & Rhee. (2019). A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. Preuzeto 24.08.2021. s https://www.researchgate.net/publication/336805909_A_High-Accuracy_Model_Average_Ensemble_of_Convolutional_Neural_Networks_for_Classification_of_Cloud_Image_Patches_on_Small_Datasets

Potdar, K. (2021). Object detection in a dense scene. Preuzeto 26.08.2021. s https://www.researchgate.net/figure/Object-detection-in-a-dense-scene_fig4_329217107

Ravish, R. (2021). Supervised, Unsupervised, And Semi-Supervised Learning With Real-Life Usecase. Preuzeto 19.07.2021. s https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning

Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (Fourth Edi). Pearson Education.

Sutton, R., & Barto, A. (2015). *Reinforcement Learning: An Introduction* (Second edi). The MIT Press.

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Academia.

V. Rodriguez, R. (2020). The 7 Key Steps To Build Your Machine Learning Model. Preuzeto 19.07.2021. s Analyitics India Magazine website: https://analyticsindiamag.com/the-7-key-steps-to-build-your-machine-learning-model/

Popis slika

Popis tablica