Empirijsko ispitivanje performansi algoritama neuronskih mreža na skupovima podataka različitih karakteristika

Pilošta, Bruno

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Bruno Pilošta

EMPIRIJSKO ISPITIVANJE PERFORMANSI ALGORITAMA NEURONSKIH MREŽA NA SKUPOVIMA PODATAKA RAZLIČITIH KARAKTERISTIKA

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Bruno Pilošta

Matični broj: 0016129843

Studij: Baze podataka i baze znanja

EMPIRIJSKO ISPITIVANJE PERFORMANSI ALGORITAMA NEURONSKIH MREŽA NA SKUPOVIMA PODATAKA RAZLIČITIH KARAKTERISTIKA

DIPLOMSKI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Dijana Oreški

Varaždin, lipanj 2022.

Bruno Pilošta

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/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj rad bavi se istraživanjem rada algoritama neuronskih mreža nad skupovima podataka različitih karakteristika. U uvodnom dijelu rada dana je motivacija odabira teme ovog rada kao i sažeti opis problematike kojom će se baviti u ovom radu. U nastavku su opisane metode i tehnike rada zajedno sa osvrtom na prethodna, odnosno istraživanja na sličnu temu. U teorijskom dijelu rada opisuju se rad i struktura umjetnih neuronskih mreža iz domene strojnog učenja zajedno sa njihovim gradivnim jedinicama, neuronima. Nastavak na to jest opis algoritma povratne propagacije i optimizacijskih algoritama koji se koriste u praktičnom dijelu rada. Isto tako navedene su grupe karakteristika skupova podataka te je dan jednostavan opis istih. Praktični dio rada uključuje opis svakog skupa podataka, njegovu analizu i postupak pripreme za modeliranje te na posljetku analizu i usporedbu performansi svih korištenih algoritama nad različitim skupovima podataka. Na kraju je dan zaključak na performanse algoritama gdje se izdaju smjernice za korištenje algoritama.

Ključne riječi: neuronske mreže, meta-značajke, strojno učenje, optimizacijski algoritmi, problemi klasifikacije i regresije

Sadržaj

1. Uvod

Tema ovog rada jest empirijsko ispitivanje performansi algoritama neuronskih mreža nad skupovima podataka različitih karakteristika. Kao što se i iz imena rada može naslutiti, u ovom radu posebice će se istraživati neuronske mreže, od njihove teorijske podloge pa do ispitivanja različitih algoritama učenja mreža, odnosno optimizacijskih algoritama.

U današnje vrijeme neuronske mreže su jedno od najistraživanijih područja umjetne inteligencije, a njihove primjene su različite. Od korištenja u prediktivne svrhe kod jednostavnih problema klasifikacije i regresije, pa do danas vrlo popularnog „dubokog učenja“. Glavni element neuronskih mreža je njihova sposobnost učenja, to jest, poboljšavanja performansi kroz određeno vremensko razdoblje. Upravo algoritmi koji se nalaze u pozadini „učenja“ neuronskih mreža biti će glavni predmet istraživanja u ovom radu. Današnji standardi uključuju algoritme temeljene gradijentnom spustu kojim se nastoji minimizirati greška neuronskih mreža, a samim time i točnost u rješavanju raznih problema koje modeli neuronskih mreža rješavaju. U ovom istraživanju analizirat će se rad algoritama uključenih u razvojni okvir „*Tensorflow/Keras*“ koji, mogli bi reći, u trenutku pisanja ovog rada, predstavlja standard u području podatkovne znanosti kod gradnje umjetnih neuronskih mreža. Kako ovaj razvojni okvir uključuje nekoliko različitih algoritama, javlja se potreba za istraživanjem potencijalnih smjernica u vidu koji algoritam koristiti kod kakvih skupova podataka. Skupovi mogu biti različitih karakteristika te je za očekivati da ne rade svi algoritmi isto kod svih skupova podataka.

Cilj ovog rada jest pronaći skupove podataka različitih karakteristika u pogledu različitih veličina skupova podataka prema broju instanci, dimenzionalnosti, problema koji se rješava i količine šuma u podacima kao i izrada prediktivnih modela treniranih različitim algoritmima te usporedba njihovih performansi na temelju različitih metrika točnosti i brzine treniranja. Na kraju rada analizirat će se razlike između algoritama ovisno o skupovima podataka uz davanje preporuka koji algoritam koristiti ili ne koristiti kod kojih skupova podataka.

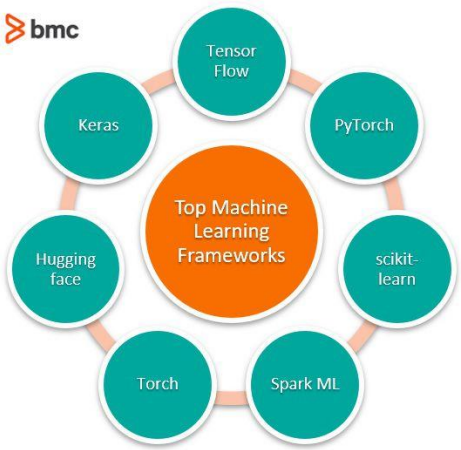
2. Metode i tehnike rada

Tema ovog rada uključuje, ne samo provođenje empirijskog testiranja, već i opis što su neuronske mreže, odnosno opis teorijske podloge na kojoj se one temelje kako bi se mogli donositi zaključci i potencijalne preporuke koji algoritam koristiti u kojem slučaju.

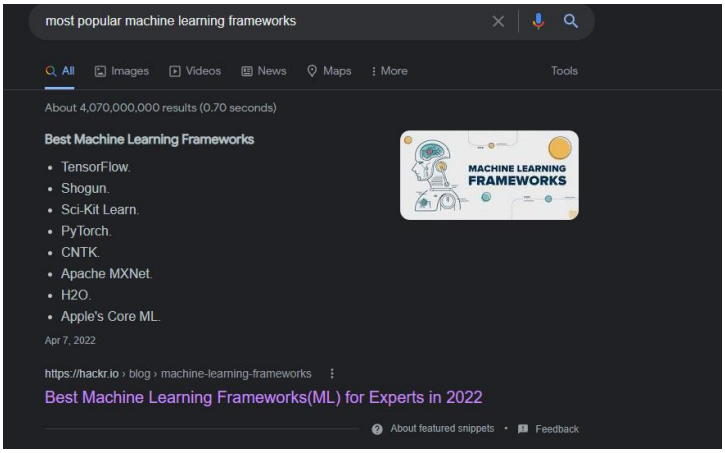
Testiranje će biti provedeno uz pomoć Jupyter Notebook-a i programskog jezika Python, odnosno njegove najpopularnije i najkorištenije biblioteke za strojno učenje i umjetnu inteligenciju Tensorflow. Upravo riječ „najpopularnija biblioteka“ bila je glavni razlog odabira ovog razvojnog okvira. Jednostavnom „Google“ pretragom za najpopularnijim okvirima strojnog učenja možemo vidjeti da je gotovo na svakoj listi Tensorflow prvi izbor.

Slika 1 Najpopularniji ML razvojni okvir prema ProjectProp[1]

[1] Poveznica: https://www.projectpro.io/article/machine-learning-frameworks/509



Slika 2 Najpopularniji ML razvojni okvir prema BMC[2]



Slika 3 Google pretraga za najpopularnijim ML razvojnim okvirima (datum: 26.4.2022.)

Upravo zbog njegove velike popularnosti odlučio sam testirati rad algoritama koji su implementirani u ovom razvojnom okviru. Kako većina ljudi koristi ovaj razvojni okvir kod izrade prediktivnih modela pomoću neuronskih mreža svakako bi bilo potrebno odraditi testiranja algoritama koji su implementirani u ovom okviru.

Optimizacijski algoritmi neuronskih mreža koji su implementirani u „Tensorflow-u“ i koji će se testirati u ovom istraživanju su „***Adam***“, „***Adamax***“, „***Adagrad***“, „***NAdam***“, „***SGD***“, „***RMSprop***“. U nastavku rada detaljnije će se opisati svaki od navedenih algoritama u vidu načina rada, razlika, prednosti i nedostataka. Teorijska analiza ovih algoritama jedan je od najvažnijih zadataka tijekom teorijske obrade teme te će veći dio obrade ove teme rada imati naglasak na opisu ovih algoritama. Osim toga u teorijskom dijelu rada opisat će se potpuni rad neuronskih mreža zajedno sa njihovim gradivnim jedinicama, neuronima.

[2] Poveznica: https://www.bmc.com/blogs/machine-learning-ai-frameworks/

U praktičnom djelu rada opisat će se svaki korišteni skup podataka u pogledu analize originalnog skupa podataka, procesa pripreme pa sve do modeliranja algoritmima neuronskih mreža. Nakon modeliranja provest će se vrednovanje modela gdje će biti opisane razlike među modelima na temelju različitih metrika. Isto tako pokušat će se analizirati teorijska povezanost rada algoritama i njihov utjecaj na same rezultate. Na kraju rada pokušat će se izvesti određene smjernice, odnosno preporuke koje algoritme koristiti u kojim slučajevima.

Za učitavanje i obradu skupova podataka koristi će se biblioteke „Pandas“ i „Numpy“ te biblioteka za vizualizaciju „Matplotlib“.

Kako je treniranje neuronskih mreža izrazito računalno zahtjevan proces potrebno je dovoljno snažno računalo kako bi izrada modela bila u prihvatljivom vremenskom roku. Prednost Tensorflow-a jest mogućnost korištenja GPU-a za treniranje neuronskih mreža što može uvelike ubrzati sam proces. No, u trenutku pisanja ovog rada korištenje GPU-a za treniranje modela neuronskih mreža u potpunosti je podržano kod novijih kartica proizvođača Nvidia zbog čega će se treniranje svih modela izvoditi na šesterojezgrenom procesoru. Kompletne specifikacije na kojima će se izvoditi treniranja modela su sljedeće:

- AMD šesterojezgreni/dvanaestdretveni Ryzen 5 3600 3.6/4.2GHz procesor
- 16 GB radne memorije
- AMD Radeon RX 480 grafička kartica
- KINGSTON SSD A2000 512GB NVMe

3. Prijašnja istraživanja

Kako bi se lakše ušlo u problematiku analize algoritama neuronskih mreže nad različitim skupovima podataka, za početak će se analizirati neka od prethodnih istraživanja na tu temu.

3.1. „Effects of data set features on the performances of classification algorithms“

U istraživanju naziva „*Effects of data set features on the performances of classification algorithms*“ autori su nastojali proučiti utjecaj različitih karakteristika skupova podataka na performanse nekoliko klasifikacijskih algoritama. Glavni zadatak bio je istražiti na koji način razne karakteristike skupova podataka utječu na ukupnu točnost kao i vrijeme potrebno za treniranje prediktivnih modela. U istraživanju je korišteno 106 skupova podataka različitih karakteristika. Karakteristike skupova podataka su postojanost varijabli kategorijskog tipa i numeričkog tipa, veličina skupa podatak (broj instanci), dimenzija skupa, postojanost nedostajućih podataka u skupu podataka, domena skupa podataka, nebalansiranost skupa podataka i dimenzija klasa.

U istraživanju je korišteno osam klasifikacijskih algoritama koji uključuju Bayesove mreže, logistički klasifikator, RBF mreže, Sequential Minimal Optimization (SMO), stabla odlučivanja, KNN,…. Evaluacija performansi izvedena je pomoću Cohen-ove Kappa statistike, RMSE, MAE i proteklo vrijeme. Na kraju je uočeno da su najbolje ukupne performanse imale Bayes-ove mreže i stabla odlučivanja.

Što se tiče karakteristika koje utječu na točnost predviđanja uočeno je da ako su „input“ varijable kategorijskog tipa, točnost pada kod algoritama logističke regresije, RBF mreže i KNN algoritama. Tip input varijable ne utječe na točnost modela izrađenog algoritmom Bayesovih mreža. Osim navedenog uočeno je da se točnost poboljšava povećanjem broja instanci u skupu podataka. Na kraju, uočeno je da jedino kod Bayesovih mreža karakteristike skupova podataka nemaju nikakav utjecaj. (Kwon & Sim, 2013)

3.2. „Impact of Dataset Size on Classification Performance: An Empirical Evaluation in the Medical Domain“

Drugo istraživanje odnosi se na ispitivanje utjecaja veličine skupa podataka na performanse algoritama kod problema klasifikacije. U ovom istraživanju empirijski se ispituju performanse nekoliko različitih algoritama nad skupovima podataka iz medicinske domene. Istraživači su koristili dvadeset skupova podataka na temelju kojih su ispitivali performanse algoritama SVM (metoda potpornih vektora), neuronske mreže, Naive Bayes algoritam, stabla odlučivanja, slučajne šume i AdaBoost algoritam. Evaluacija performansi odrađena je pomoću nekoliko raznih metrika koje uključuju točnost, preciznost, odaziv, F mjeru, ROC AUC.

Od spomenutih dvadeset skupova podataka osamnaest ih je „manjih“, a dva „velika“. Kako bi ispitali performanse manjih skupova podataka u odnosu na velike napravljena je segmentacija velikih skupova podataka na skupove manjih veličina od 980, 490 i 98 instanci koji su zapravo predstavljali kategorije veličina podataka „veliki“, „srednji“ i „mali“. Performanse algoritama testirale su se nad segmentima u odnosu na preostale manje skupove podataka koji su bili veličina od 18 – 1030 instanci.

Najrobusniji algoritmi pokazali su se AdaBoost i Naive Bayes. Osim toga zaključeno je da veličina skupa podataka nema utjecaj na performanse, već koliko distribucija podskupova podataka (segmenata) predstavlja originalni skup podataka.(Althnian et al., 2021)

3.3. Komentar na prethodna istraživanja

Prilikom traženja prethodnih istraživanja na temu koja se veže na ispitivanje performansi raznih algoritama nad skupovima podataka različitih karakteristika uočeno je da postoji izrazito mali broj istraživanja koje se bave ovom problematikom. Također, što se tiče komparacije algoritama neuronskih mreža nisu pronađena istraživanja koja se bave analizom algoritama neuronskih mreža.

Dva opisana istraživanja bavila su se komparacijom performansi raznih algoritama klasifikacije, no jedino je drugo istraživanje uključilo neuronske mreže u ispitivanju performansi. Osim toga, opisana istraživanja uključivala su pretežito točnosti predviđanja bez uključivanja vremena potrebnog za treniranje/testiranje prediktivnih modela, pogotovo kod skupova podataka različitih veličina gdje je za pretpostaviti da će vrijeme potrebno za treniranje modela kod velikih skupova podataka biti veće nego kod manjih skupova.

4. Struktura i rad neuronskih mreža

Kako bi mogli lakše razumjeti što su to neuronske mreže u području umjetne inteligencije trebali bi utvrditi od kuda dolazi naziv „neuronske mreže“. Kao što i sam naziv govori, neuronske mreže bile bi određeni umreženi elementi, odnosno umreženi neuroni.

Riječ neuron, u domeni strojnog učenja, naziv vuče iz područja biologije, odnosno anatomije mozga gdje neuron predstavlja temeljnu jedinicu mozga i živčanog sustava. Neuron bi mogli definirati kao stanicu koja se na temelju podražaja (inputa) aktivira, odnosno, šalje naredbe drugim stanicama u tijelu. Kako svaka osoba ima oko 100 milijardi neurona koji su međusobno povezani, takva mreža zove se neuronska mreža. (Dr. Woodruff, n.d.)

Ova inspiracija iz područja biologije dovela je do jednog od najvećih pomaka u području umjetne inteligencije, ali i same tehnologije, gdje su, uz pomoć današnje vrlo velike računalne snage, mogućnosti gotovo beskonačne.

Kao i takozvana „biološka neuronska mreža“ (u nastavku BNN), umjetna neuronska mreža (u nastavku ANN) sastoji se od umreženih, u ovom slučaju, umjetnih neurona. Neuron u domeni ANN predstavlja matematički model. Kao i kod BNN-a neuron u ANN-u ima sposobnosti aktivacije, odnosno na izlazu se aktivira signal koji se šalje kroz ostatak mreže. U ANN-u signal se aktivira kada linearna kombinacija inputa prelazi postavljeni prag koji ovisi o aktivacijskoj funkciji.(Russell & Norvig, 2010)

Kao i BNN, ANN se sastoji od velikog broja ovakvih ukratko opisanih neurona. U nastavku će se detaljnije opisati struktura neuronske mreže i način rada. Neuronske mreže koje se spominju u nastavku rada odnose se na umjetne neuronske mreže iz domene strojnog učenja i umjetne inteligencije.

4.1. Struktura neuronskih mreža

Kao što je već spomenuto, neuronske mreže sastoje se od neurona, odnosno čvorova povezanih usmjerenim vezama. Neuronske mreže započinju takozvanim „input“ čvorovima, odnosno čvorovima koji predstavljaju „ulazak“ u neuronsku mrežu. „Input“ čvorovi, odnosno ulazni čvorovi, predstavljaju vidljivi sloj neuronske mreže i preko ovog sloja podaci našeg skupa podataka zapravo ulaze u neuronsku mreže i od njih se propagiraju kroz ostatak mreže. Iz ulaznih čvorova veze mogu ići direktno na izlazni sloj ili, češće, prema takozvanom skrivenom sloju iz kojeg veze idu prema izlaznom sloju. U prvom slučaju govorimo o

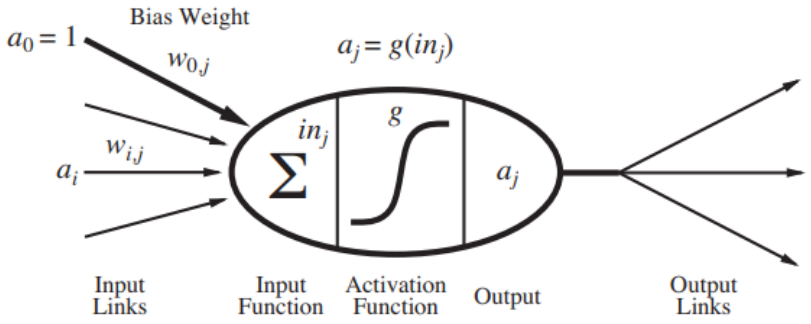
„jednoslojnim“ neuronskim mrežama.(Russell & Norvig, 2010). U nastavku rada naglasak će biti na neuronskim mrežama koje sadrže barem jedan skriveni sloj.

Kada imamo u neuronskoj mreži uključen veći broj skrivenih slojeva možemo govoriti o takozvanom „dubokom učenju“. Riječ „dubina“ odnosi se na broj skrivenih slojeva u neuronskoj mreži, odnosno što više skrivenih slojeva imamo, možemo reći da je neuronska mreža „dublja“. U literaturi se spominju različiti podaci, no o dubokom učenju, prema IBM-u, govorimo tek kada neuronska mreža sadrži više od tri skrivena sloja. (Kavlakoglu, 2020) Neuroni u ovom sloju, na temelju ulaznih vrijednosti od neurona ulaznog sloja i neurona prethodnih skrivenih slojeva, generiraju izlaz pomoću aktivacijske funkcije.(Techopedia, 2018)

Na kraju imamo izlazni sloj koji se može sastojati od različitog broja neurona ovisno o tipu problema koji se rješava. Ovaj sloj predstavlja izlaz iz neuronske mreže te ovdje čvorovi sadrže finalne vrijednosti koje odgovaraju rezultatu problema koji rješavamo. (Russell & Norvig, 2010) Primjerice, ako imamo problem n-višeklasne klasifikacije, imat ćemo n čvora u ovom sloju od kojeg svaki može predstavljati vjerojatnost klasifikacije za određenu klasu. Ako imamo problem regresije imat ćemo jedan čvor u izlaznom čvoru koji će predstavljati neku, predviđenu, numeričku vrijednost.

5. Struktura i rad neurona

Do sada je opisana struktura neuronskih mreža, dok će se u ovom poglavlju detaljnije opisati od čega se sastoje njezini gradivni elementi, neuroni. Na slici 4 možemo vidjeti strukturu jednog neurona.



Slika 4 Struktura neurona (Izvor: Russell & Norvig, 2010)

Kao što se u prethodnim poglavljima spominjalo, umjetni neuron sadrži ulazne veze i izlazne. Sa i označen je neuron koji prethodi promatranom neuronu j, a izlazne veze a_i neurona i predstavljaju ulazne veze promatranog neurona j. Neuron j može imati više ulaznih veza, a svaka veza predstavlja točno jedan izlaz iz nekog neurona iz prethodnog sloja. Sa $w_{i,j}$ označavamo težinu koja se pridodaje svakoj od ulaznih veza, a predstavlja važnost veze iz prethodnog neurona i u neuron j. Računanje izlazne vrijednost neurona započinje računanjem ponderirane sume svih ulaznih vrijednosti u neuron sljedećom formulom:(Russell & Norvig, 2010)

$$in_j = \sum_{i=0}^{n} w_{i,j} * a_i$$

U prikazanoj formuli in_j označava ponderiranu sumu ulaznih vrijednosti u neuron j. Ona se računa na način da se zbroje umnošci svih ulaznih veza i pripadajućih težine veza. Nakon toga nad ponderiranom sumom primjenjuje se aktivacijska funkcija. Aktivacijska funkcija najvažniji je element svakog neurona zbog čega će joj se u nastavku posvetiti jedno poglavlje gdje će se detaljnije opisati koja je njezina funkcija i koje aktivacijske funkcije je moguće koristiti i u kojim slučajevima.

Izlaz a_j iz neurona j zapravo predstavlja vrijednost koja se dobije primjenom aktivacijske funkcije nad dobivenom ponderiranom sumom in_j. Izlaz a_j možemo zapisati na sljedeći način:(Russell & Norvig, 2010)

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} * a_i\right)$$

gdje g predstavlja aktivacijsku funkciju koja se primjenjuje nad in_j. Ono što smo mogli primijetiti na prikazanoj strukturi neurona jest vrijednost $a_0 = 1$ za „nulti“ neuron. To nam označava takozvani „dummy“ ulaz odnosno „bias neuron“ ili „neuron pristranosti“ . Kao što vidimo izlazna vrijednost ovog neurona jednaka je 1 što zapravo znači da će in_j biti model linearne regresije jer će se vrijednost umnoška težine ove veze i izlazne vrijednosti neurona pristranosti $w_{0,j} * a_0$ biti $w_{0,j}$ pošto je a_0 jednak 1. Upravo ovo omogućava aktivacijskoj funkciji fleksibilnost prilikom traženja rješenja pošto ova konstanta omogućava pomak same funkcije ovisno o vrijednosti težine koja se dodjeli vezi sa neuronom pristranosti. (Gebel, 2020) Računanje vrijednosti težina za sve neurone mijenja se tijekom učenja neuronske mreže nekim od algoritama koje ćemo kasnije opisati i testirati u praktičnom djelu ovog rada.

5.1. Aktivacijske funkcije

Kao što je opisano u prethodnom poglavlju aktivacijske funkcije, ili funkcije prijenosa, primjenjuju se nad ponderiranom sumom inputa neurona, ali pitanje koje se postavlja koja je njezina funkcija, odnosno što bi se dogodilo kada se ne bi primijenila aktivacijska funkcija.

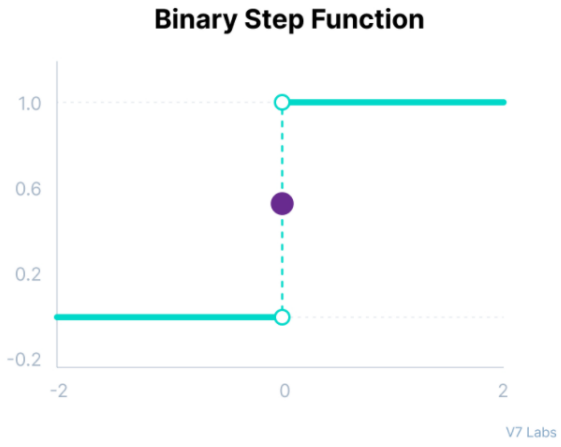
Vidjeli smo da se aktivacijska funkcija primjenjuje nad ponderiranom sumom, koja dodavanjem neurona pristranosti, ima oblik formule modela linearne regresije. Primjenom aktivacijske funkcije određujemo koja će biti izlazna vrijednost iz neurona. Ako ne bi primjenjivali aktivacijsku funkciju nad izračunatom ponderiranom sumom ulaza, svaki čvor bio bi jednostavno model linearne regresije što znači da bi cijeli prediktivni model izrađen pomoću neuronske mreže bio jedan model linearne regresije. Razlog je što kompozicija više linearnih funkcija (čvorova), jest zapravo sama linearna funkcija. To znači da naš model ne bi mogao uočiti nelinearnost u skupu podataka. (Baheti, 2022)

Važnost nelinearnosti kod neuronskih mreža očituje se u algoritmima učenja/optimizacije u „povratnoj propagaciji“ (engl. „backpropagation“) neuronskih mreža koji se, prema današnjim standardima, pretežito temelje na algoritmu „gradijentnog spusta“ (engl. „gradient descent“) u čemu će detaljnije biti riječ u sljedećem poglavlju.

Same aktivacijske funkcije mogli bi podijeliti u tri kategorije: „Binary Step Function“, Linearna aktivacijska funkcija i nelinearne aktivacijske funkcije.(Baheti, 2022)

5.1.1.Binary Step funkcija

Kao što i ime govori funkcija „Binary Step“ daje binarne izlaze 0 ili 1 ovisno o ulazima.



Slika 5 Binary Step funkcija - Izvor: (Baheti, 2022)

Ono što vidimo jest da ako je input funkcije (ponderirana suma ulaza u neuron) manja od 0 funkcija daje vrijednost 0, a u suprotnom, ako je input veći od 0, vraća jedan.(Baheti, 2022) Naravno, taj prag od nule ne mora biti fiksan, već se može odrediti proizvoljno pa bi zato mogli opisati ovu funkciju na način da ako je vrijednost inputa manja od praga, ona vraća nula, a u suprotnom jedan.

Iz opisanog vidimo da ovo nije najbolja funkcija za korištenje u skrivenim slojevima pošto je njezin izlaz binarna vrijednost, čime ne možemo dobro prikazati nelinearnost, što otežava proces povratne propagacije. (Baheti, 2022)

S druge strane ova funkcija mogla bi se koristiti na izlaznom čvoru kod problema binarne klasifikacije pošto je i njezin izlaz binarna vrijednost, no u nastavku će se prikazati bolja funkcija koja se koristi na izlaznim čvorovima kod problema klasifikacije.

5.1.2. Linearna funkcija

Linearna funkcija zapravo je najjednostavnija aktivacijska funkcija od opisanih te je vrlo intuitivna za shvatiti. Kao što je bilo opisano od prije na ulazu u čvor imamo linearnu kombinaciju inputa, na što primjenjujemo aktivacijsku funkciju. Ako za aktivacijsku funkciju uzmemo linearnu funkciju, zapravo ne radimo nikakve promjene nad ulazom. To možemo vidjeti iz njezine formule:

$$f(x) = x$$

gdje x predstavlja input, to jest u našem slučaju, ponderiranu sumu inputa. Iz toga dolazi zaključak da će izlaz iz neurona, primjenom aktivacijske funkcije, biti sama ponderirana suma input.(Brownlee, 2021e)

Iz tog razloga ovu funkciju ne možemo nikako primjenjivati kod neurona skrivenih slojeva pošto se narušava teza da bi neuroni skrivenih slojeva trebali pronalaziti nelinearnosti kod podataka. Ali linearna funkcija ima svoju namjenu, a to je kod problema regresije gdje želimo predvidjeti konkretnu numeričku vrijednost. U tom slučaju ovu aktivacijsku funkciju postavili bi na izlazni čvor te bi na taj način dobili konkretnu predviđenu numeričku vrijednost.

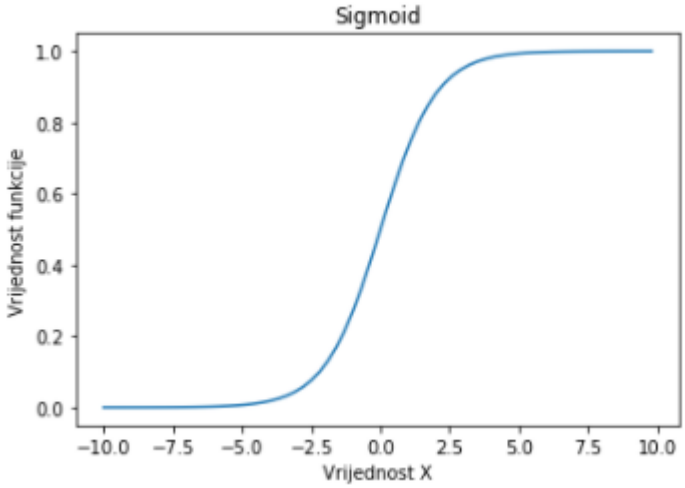
5.1.3. Nelinearne funkcije

U prethodna dva podpoglavlja opisane su jednostavne aktivacijske funkcije koje ipak ne možemo koristiti u skrivenom sloju upravo zbog njihove jednostavnosti. Rješenje problema uočavanja kompleksnih veza između ulaznih varijabli i izlazne varijable omogućavaju nelinearne aktivacijske funkcije.

Važna karakteristika aktivacijskih funkcija skrivenog sloja jest da ona mora biti diferencijabilna, odnosno, moguće je izračunati njezinu derivaciju na cijeloj njezinoj domeni. Razlog tome leži u algoritmu učenja mreže znanoj kao „backpropagation“ koji traži parcijalne derivacije na svakom čvoru mreže prilikom ažuriranja vrijednosti težina, no više u tome u poglavlju koji se bavi ovom temom.(Brownlee, 2021e)

5.1.4.Funkcija sigmoid

Prva vrlo popularna funkcija jest funkcija sigmoid/logistička funkcija. Funkcija za ulaz prima realne vrijednosti koje preslikava u raspon od nula do jedan. To možemo vidjeti iz njezinog grafa:



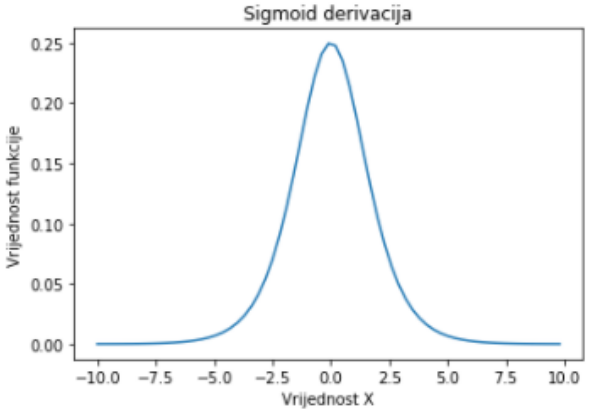
Slika 6 Graf funkcije sigmoid na domeni (-10,10) (Izrađeno u Python-u)

Matematička reprezentacija funkcije je sljedeća:

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

Mogućnosti korištenja ove funkcije su i u skrivenom sloju zbog njezine diferencijabilnosti i derivacije veće od nule na cijeloj domeni što omogućava algoritmu gradijentnog spusta poboljšavanje modela učenjem u svakoj iteraciji. Osim toga, ona poprima vrijednosti od nula do jedan što omogućava korištenje i na izlaznom sloju kod predviđanja u klasifikacijskim problemima i to u obliku vjerojatnosti izlaza. (Baheti, 2022)

Problem ove funkcije možemo vidjeti na grafu njezine derivacije:



Slika 7 Graf derivacije funkcije sigmoid na domeni (-10,10) (Izrađeno u Python-u)

Ovdje vidimo da veći gradijent postoji samo na domeni od oko -3 do 3. Problem koji se ovdje javlja jest takozvani „nestajući gradijent“ (engl. „Vanishing gradient“). Gradijent se koristi kod povratne propagacije gdje se pomoću njega ažuriraju težine u čvorovima te se prilikom povratka do početnog skrivenog sloja oni množe na svakom sloju. Kada imamo funkciju koja vraća male gradijente, ovo ulančano množenje rezultira jako malim gradijentom koji ažurira težine početnog skriveni sloja koji je jedan od najvažnijih slojeva u mreži te samim time treniranje postaje izrazito neefikasno i na kraju dovodi to slaboj predviđanja.(Wang, 2019)

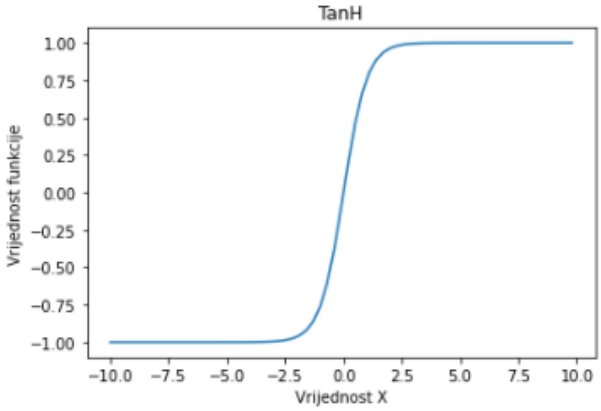
Još jedan problem koji se javlja kod ove funkcije jest eksponencijalna operacija koja se javlja kod njezinog računanja. Poznato je da su eksponencijalne operacije računalno skupe što jako šteti performansama treniranja modela. (Chen, 2021)

5.1.5.Funkcija tangens hiperbolni

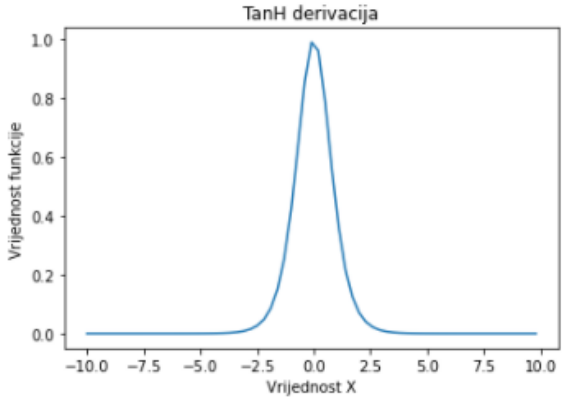
Hiperbolni tangens funkcija slična je sigmoidnoj, no razlika jest što su njezine izlazne vrijednosti u rasponu od -1 do 1. Isto tako, za razliku od sigmoide, izlazne vrijednosti centrirane su oko nule. Njezina matematička reprezentacija je sljedeća:(Chen, 2021)

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

Odmah prema formuli vidimo problem koji se javlja, a to su eksponencijalne operacije koje su računalno skupe. Možemo vidjeti i njezin graf i graf derivacije:



Slika 8 Graf funkcije TanH na domeni (-10,10) (Izrađeno u Python-u)

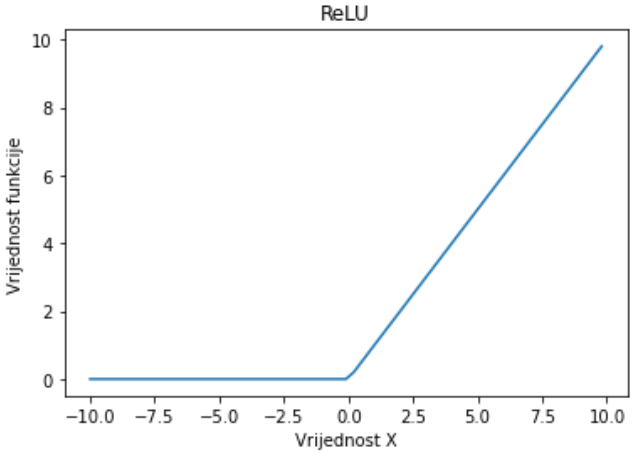


Slika 9 Graf derivacije funkcije TanH na domeni (-10,10) (Izrađeno u Python-u)

Iz grafa derivacije opet možemo vidjeti da postoji problem „nestajućeg gradijenta“ pošto vidimo da veći nagib postoji samo kada su vrijednosti u rasponu od oko -3 do 3, ali u ovom slučaju on je veći (vrijednost derivacije za X = 0 je 1, kod sigmoide 0.25).

5.1.6.ReLU

Aktivacijska funkcija koja rješava sve probleme koji su uočeni kod funkcija TanH i Sigmoid zove se ReLU ili „Rectified Linear Unit“, no dovodi određene nove probleme u obliku „mrtvih neurona“. Mogli bi gotovo reći da je ova funkcija, u trenutku pisanja ovog rada, standard u području neuronskih mreža zbog svoje velike jednostavnosti, ali i izrazito velike snage kod konvergencije gradijentnog spusta prema globalnom minimumu funkcije gubitka.(Baheti, 2022) Upravo njezina jednostavnost prikazana je u sljedećem grafu:

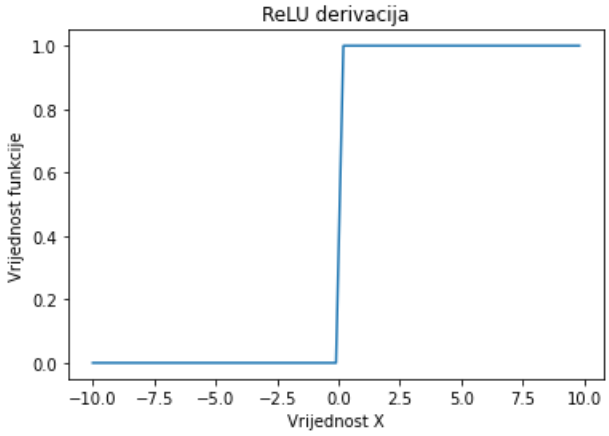


Slika 10 Graf funkcije ReLU na domeni (-10,10) (Izrađeno u Python-u)

Matematička reprezentacija ove funkcije je sljedeća:

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

Preko funkcije i grafa vidimo da se nelinearnost odražava kroz dvije linearne komponente koje se spajaju u nuli. Za vrijednosti manje od nula funkcija daje vrijednost 0, odnosno derivacija za vrijednosti manje od nula jest nula, odnosno na tom području nagib je nula, u suprotnom funkcija daje vrijednost x, odnosno nagib jest 1:



Slika 11 Graf derivacije funkcije ReLU na domeni (-10,10) (Izrađeno u Python-u)

Kao što vidimo jedino područje u kojem funkcija nije linearna jest oko nule, odnosno gdje se spajaju dvije linearne komponente.(Chen, 2021) Prednost ove funkcije je niska računalna složenost (funkcija vraća ili nula ili x) kao i njezina karakteristika da funkcija nema maksimalnu vrijednost za razliku od ostalih spomenutih funkcija što pomaže algoritmu gradijentnog spusta. (Chen, 2021) Problem ove funkcije javlja se kada su vrijednosti inputa u funkciju manje od nula pošto je u tom slučaju vrijednost derivacije jednaka nuli. Ono što se može dogoditi jest da vrijednosti ponderirane sume padnu ispod nule te se u tom slučaju neuron više neće aktivirati,

a prilikom povratne propagacije težine se neće ažurirati pošto je derivacija funkcije u tom slučaju nula. Na taj se način neki neuroni više ne mogu aktivirati, odnosno izlaz iz aktivacijske funkcije kod takvog neurona biti će nula kroz preostalo vrijeme treniranja.(Chen, 2021)(Géron, 2017)

5.1.6.1. Varijante ReLU funkcije

Problem „mrtvih neurona“ riješen je uz pomoć nekoliko varijanti ReLU funkcija. Prva jest tzv. „Leaky ReLU“ iz čije matematičke reprezentacije možemo iščitati rješenje spomenutog problema:

$$f(x) = \max(\alpha * x, x)$$

gdje je α parametar proizvoljne vrijednost koji određuje nagib funkcije.(Chen, 2021)

Vidimo da funkcija vraća veću vrijednost između $\alpha * x$ i x. To omogućava da za vrijednosti $x < 0$ manje od nećemo imati derivaciju manju od nule, a niti vrijednosti jednake nuli koju funkcija vraća. Kod ove funkcije, na cijelom njezinom području djelovanja, postoji određeni nagib. Na taj se način , kroz povratnu propagaciju, neće dogoditi da se težine neurona više ne mogu ažurirati, ali ni da se neuroni više neće aktivirati.

Također postoji i varijacija „Leaky ReLU“ funkcije znana i kao „Parametric ReLU“, a njezina karakteristika jest da se vrijednost α može mijenjati tijekom povratne propagacije baš kao i težine.

Još jedna varijanta ReLU funkcije jest i ELU odnosno „Exponential Rectified Unit“. Riječ „Exponential“ u njezinom nazivu govori da se djelomično razlikuje od prethodnih varijanti jer sadrži eksponencijalnu komponentu, a ona se odnosi na negativni dio krivulje što možemo vidjeti u njezinoj matematičkoj reprezentaciji. Funkcija je definirana na sljedeći način:(Chen, 2021)

$$\begin{cases} \alpha(e^x - 1) \ \ ako\ je\ x \leq 0 \\ \quad\quad x\ ako\ je\ x > 0 \end{cases}$$

Vidimo da funkcija koristi eksponencijalnu krivulju za reprezentaciju negativnih inputa. To također rješava problem „mrtvih neurona“ koji se pojavljuju u originalnoj verziji ReLU funkcije, no problem se pojavljuje u korištenju eksponencijalne funkcije za reprezentaciju negativnih inputa što, kao što je već spomenuto u prethodnim poglavljima, narušava brzinu učenja mreže. (Chen, 2021)

5.2. Zaključak na korištenje aktivacijskih funkcija

Kroz prethodna poglavlja opisano je nekoliko različitih aktivacijskih funkcija gdje je analiziran njihov rad te prednosti i nedostaci. Mogli smo uočiti da korištenje nelinearnih aktivacijskih funkcija nema smisla u skrivenim slojevima neuronske mreže.

U praktičnom djelu rada isprobat će se različite aktivacijske funkcije, ovisno o sloju mreže u kojima će se nalaziti. Kod klasifikacijskih problemima, na izlaznom sloju mreže, koristit će se Sigmoida, pošto ona daje vrijednosti između nula i jedan što nam zapravo daje dobru brojku koliko je model siguran u svoje predviđanje u obliku vjerojatnosti. U problemima regresije na izlaznom čvoru koristit će se linearna aktivacijska funkcija kako bi dobili točnu predviđenu numeričku vrijednost, što nam je u ovakvom problemu zapravo i cilj.

U skrivenim slojevima svakako će se koristiti ReLU funkcija, odnosno neka od njezinih varijanti. Ispitat će se performanse s različitim funkcijama te će se odabrati ona koja daje najbolju ukupnu točnost modela, ali uzet će se u obzir i vrijeme treniranja odnosno testiranja.

6. Backpropagation algoritam

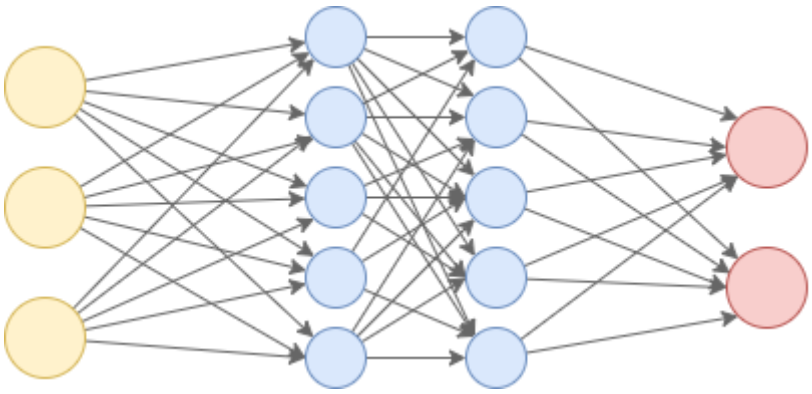
U ovom poglavlju opisat će se algoritam koji zapravo omogućava neuronskoj mreži prilagodbu, to jest, „učenje“. Spominjanjem pojma „učenje“ očekujemo proces koji se ne odvija u kratkom vremenskom razdoblju, već proces koji se odvija kroz više ponavljajućih iteracija. Upravo to se događa tijekom „učenja“ neuronske mreže.

Do sada su spomenute težine koje određuju važnost veze u neuronu te da se navedene težine tijekom učenja trebaju ažurirati, a način na koji se to odvija opisat će se u ovom poglavlju.

6.1. Intuitivni opis algoritma

Sam algoritam povratne propagacije intuitivno je izrazito jednostavan. Prilikom treniranja neuronske mreže imamo skup podataka na temelju kojeg želimo izraditi model, odnosno, čije izlazne vrijednosti želimo predvidjeti.

Za početak prikazat će se jednostavna arhitektura neuronske mreže sa tri neurona na ulaznom sloju, po pet neurona u dva skrivena sloja i dva izlazna neurona. Navedena tri ulazna neurona predstavljali bi neka tri atributa u skupu podataka na temelju kojih se instance u skupu podataka trebaju klasificirati u neke dvije kategorije koje predstavljaju dva izlazna neurona. Struktura svakog neurona identična je onoj opisanoj u poglavlju „Struktura i rad neurona“ to jest svaki neuron definiran je ulaznim vezama od čvora i sa pripadajućim vrijednostima a_i, težinama veze $w_{i,j}$ od čvora i prema čvoru j, ponderiranom sumom in_j umnožaka ulaznih vrijednosti i pripadajućih težina, aktivacijskom funkcijom g te izlazom čvora a_j. U arhitekturi prikazanoj na sljedećoj slici neuroni označeni žutom bojom predstavljaju ulazni sloj, plavom skriveni sloj, a crvenom izlazni sloj.

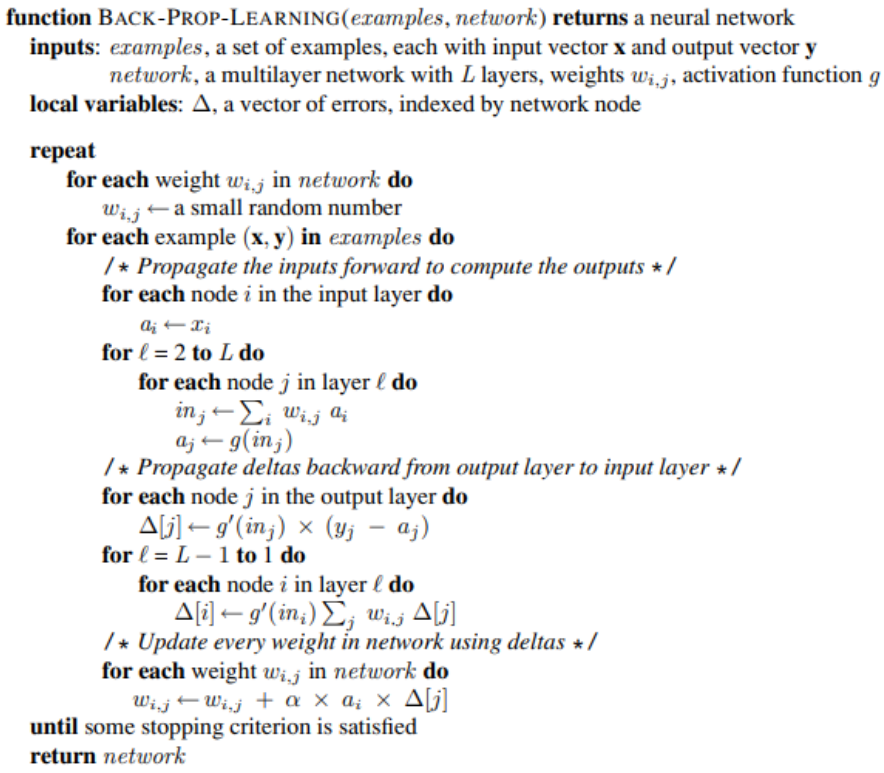


Slika 12 Jednostavna arhitektura neuronske mreže (Izrađeno u draw.io)

Nakon što imamo arhitekturu možemo na jednostavan način opisati algoritam povratne propagacije kroz mrežu. Kao što je već spomenuto algoritam povratne propagacije predstavlja iterativan proces. Za početak uzeli bi prvu instancu našeg skupa podataka koju bi propagirali kroz mrežu na način da bi se vrijednosti za svaki od naša tri atributa propagirala od ulaznog sloja do izlaznog pri čemu bi se računali izlazi iz svakog čvora tako sve do izlaza. Izlazni neuroni dali bi neku predviđenu vrijednost, koja bi na početku, naravno, bila daleko od točne. Nakon dobivanja predviđene vrijednosti računa se izlazna greška mreže koja predstavlja razliku između željene izlazne vrijednosti i dobivene izlazne vrijednosti. Sljedeći korak jest računanje koliko je koji neuron zaslužan za ukupnu grešku od zadnjeg sloja do početnog sloja. Na kraju provodi se optimizacija mreže na način da se težine svih neurona ažuriraju na temelju izračunate greške za koju je određeni neuron zaslužan. Cijeli ovaj proces provodili bi iterativno za svaku instancu skupa podataka za treniranje. (Géron, 2017)

6.2. Detaljni opis algoritma

Nakon što je na jednostavan način dan uvod u ovaj algoritam potrebno je detaljnije pojasniti na koji način sam algoritam funkcionira pomoću pseudokoda. Stuart Russell i Peter Norvig u svojoj knjizi *„Artificial Intelligence: A Modern Approach“* definirali su vrlo intuitivan pseudokod algoritma povratne propagacije koji ću u nastavku opisati.



```
function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with L layers, weights w_{i,j}, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight w_{i,j} in network do
      w_{i,j} ← a small random number
    for each example (x, y) in examples do
      /* Propagate the inputs forward to compute the outputs */
      for each node i in the input layer do
        a_i ← x_i
      for ℓ = 2 to L do
        for each node j in layer ℓ do
          in_j ← Σ_i w_{i,j} a_i
          a_j ← g(in_j)
      /* Propagate deltas backward from output layer to input layer */
      for each node j in the output layer do
        Δ[j] ← g'(in_j) × (y_j − a_j)
      for ℓ = L − 1 to 1 do
        for each node i in layer ℓ do
          Δ[i] ← g'(in_i) Σ_j w_{i,j} Δ[j]
      /* Update every weight in network using deltas */
      for each weight w_{i,j} in network do
        w_{i,j} ← w_{i,j} + α × a_i × Δ[j]
  until some stopping criterion is satisfied
  return network
```

Slika 13 Algoritam Backpropagation prema Russell S. i Norvig P.(Russell & Norvig, 2010), strana 734.

Kao što vidimo algoritam bi za parametre primao skup podataka za treniranje zajedno sa definiranom arhitekturom neuronske mreže, a vratio bi istreniranu neuronsku mrežu. Mreža se sastoji od L slojeva, a svaka instanca skupa podataka definirana je vektorom x, koji predstavlja vektor „prediktora“, i vektorom y koji predstavlja izlazni vektor (stvarne izlazne vrijednosti). Isto tako postoji lokalna varijabla vektora grešaka indeksirana prema čvorovima mreže koju ću u nastavku označavati kao ErrorVector.

Algoritam se izvodi tako dugo dok neki kriterij zaustavljanja nije zadovoljen. Na početku potrebno je odrediti početne vrijednosti svih težina veza među čvorovima kako bi se vrijednosti mogle propagirati prema naprijed. Početne vrijednosti težina najčešće su slučajni brojevi vrijednosti blizu nule. Nakon toga za svaku instancu iz skupa podataka izvodi se nekoliko koraka.

Prvi jest da se na izlazu iz neurona početnog sloja postave vrijednosti koje odgovaraju vrijednostima atributa instance, odnosno za svaki neuron i u početnom sloju mreže izlazna vrijednost a_i odgovara nekoj od vrijednosti iz ulaznog vektora x. Svaki neuron početnog sloja predstavlja neki od prediktora. Tu naravno broj neurona mora odgovarati broju prediktora, odnosno, atributa pomoću kojih želimo izvesti predviđanje.

U sljedećem koraku za svaki sloj od prvog skrivenog sloja do zadnjeg, to jest za svaki neuron j u tim slojevima, računamo ponderiranu sumu težina i vrijednosti izlaza iz prijašnjih neurona in_j i izlaznu vrijednost iz neurona j $a_j = g(in_{i,j})$.

Nakon što se provede prethodni korak za sve neurone u mreži računamo grešku na izlazu. Ovaj korak provodi se za svaki neuron u izlaznom sloju gdje dobivamo vektor greške $ErrorVector[j]$ gdje j predstavlja neuron u izlaznom čvoru. Računanje greške ovisi o funkciji greške/koštanja/gubitka koja se koristi. Funkcija koštanja često ovisi o problemu koji se rješava, a neke od njih su „Mean Squared Error“(MSE), „Mean Absolute Error“ (MAE), „Categorical Cross Entropy“, „Binary Cross Entropy“. (Mulla, 2020)

U ovom generalnom algoritmu imamo vektor ukupne greške pošto na izlazu može biti više neurona, no vrlo često na izlazu postoji samo jedan neuron, pa bi u tom slučaju imali jedinstvenu vrijednost ukupne greške na temelju koje se ostale vrijednost gradijenta računaju.

Nakon računanja ukupne greške mreže, odnosno vektora ukupne greške, sama greška povratno se propagira do početnog sloja mreže. Ovaj korak uključuje računanje gradijenta, odnosno vektora parcijalnih derivacija po svim težinama veza u mreži. Na kraju se provodi korak optimizacije, koji najčešće predstavlja algoritam gradijentnog spusta koji će se detaljnije opisati u nastavku.(Russell & Norvig, 2010)

7. Optimizacijski algoritmi

U prethodnim poglavljima opisivani su različiti elementi neuronskih mreža, od funkcije neurona, arhitekture neuronskih mreža sve do posljednjeg elementa, a to je algoritam povratne propagacije. Opisano je na koji način funkcionira algoritam povratne propagacije, a posljednji element neuronskih mreža koji je preostao jest opis optimizacijskih algoritama.

Posljednji korak u učenju neuronskih mreža, odnosno algoritmu povratne propagacije, jest korak optimizacije. Ono što smo do sada zaključili jest da se korak optimizacije provodi algoritmom gradijentnog spusta. Optimizacijski algoritmi koji se danas koriste u praksi jesu bazirani na gradijentnom spustu, ali sam algoritam bez dodatnih modifikacija, sam po sebi, rijetko se koristi, a razlozi tome leže u snazi današnjih računala. U nastavku će se opisati razne varijante gradijentnog spusta koje će se testirati u praktičnom djelu rada, a implementirane su u razvojnom okviru „Tensorflow“.

7.1. Gradient Descent (gradijentni spust)

Algoritam gradijentnog spusta (u nastavku GD) koristi se u različite svrhe te nije posebnost optimizacije neuronskih mreža. Zapravo, GD je optimizacijski algoritam za rješavanje širokog raspona različitih problema optimizacije, a njegova upotreba kod neuronskih mreža samo je jedan od njih. Kod neuronskih mreža ovaj se algoritam koristi kako bi se, na temelju dobivenih grešaka za koje su zaslužni pojedini neuroni, optimizirale težine njihovih ulaznih veza što na posljetku ima za posljedicu manju ukupnu grešku modela neuronske mreže, pa samim time i točnost modela.(Géron, 2017)

GD je algoritam koji je ovisan o prvoj derivaciji funkcije gubitka, to jest na temelju parcijalnih derivacija funkcije gubitka ažuriraju se težine neurona. Jednostavnim riječima, algoritam računa na koji je način potrebno smanjiti ili povećati vrijednost težina veza svakog neurona kako bi se funkcija gubitka minimizirala. Prije koraka optimizacije potrebno je izračunati gradijentni vektor, odnosno, vektor parcijalnih derivacija funkcije koštanja prema svakoj težini veza koje postoje u mreži. Gradijentni vektor možemo zapisati na sljedeći način:

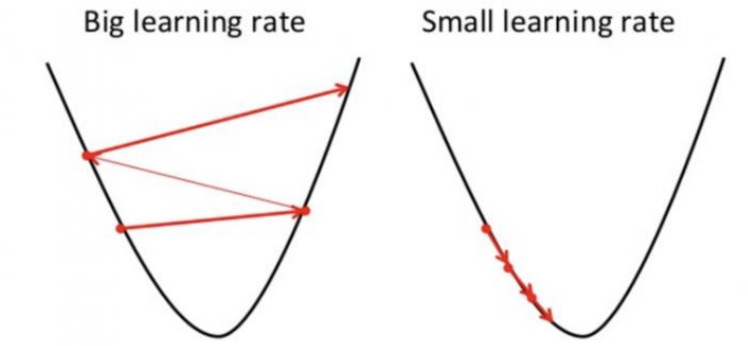
$$\nabla C = \begin{pmatrix} \frac{\partial C}{\partial w_0} \\ \frac{\partial C}{\partial w_1} \\ \cdots \\ \frac{\partial C}{\partial w_n} \end{pmatrix}$$

gdje ∇C označava gradijent funkcije koštanja, a w_n težinu n-te veze u mreži. Prema ovoj formuli vidimo da je ∇C zapravo vektor parcijalnih derivacija po težinama veza u mreži. (Hill, 2018)(Géron, 2017)

Jedan optimizacijski korak gradijentnog spusta možemo matematički zapisati na sljedeći način:

$$w_{t,i} = w_{t-1,i} - \eta * \frac{\partial C}{\partial w_{t-1,i}}$$

gdje $w_{t,i}$ predstavlja novu vrijednost težine i-te veze u neuronskoj mreži, $w_{t-1,i}$ trenutnu vrijednost težine i-te veze, $\frac{\partial C}{\partial w_{t-1,i}}$ parcijalnu derivaciju funkcije koštanja po težini i-te veze, a η stopu učenja (engl. learning rate). Stopu učenja do sada nismo spominjali, no ona je izrazito važna kod koraka optimizacije. Vrijednost $\frac{\partial C}{\partial w_{t-1,i}}$ govori nam koliki utjecaj ima težina w_i na povećanje funkcije koštanja. Kako imamo algoritam kojim želimo minimizirati funkciju koštanja u formuli koristimo znak „minus“ kako bi ažurirali težine, ne na način da povećamo vrijednost funkcije koštanja, već smanjimo što nam je i cilj. Isto tako moramo objasniti zašto koristimo stopu učenja, a to možemo vidjeti prema sljedećoj, vrlo pojednostavljenoj slici funkcije koštanja:



Slika 14 GD sa različitim stopama učenja[3]

Tijekom koraka optimizacije potrebno je smanjiti vrijednost funkcije koštanja na način da ažuriramo težine koje imaju utjecaj na ukupnu grešku, a povećanje ili smanjivanje vrijednosti težina ne smije biti preveliko, ali ni premalo. Ono što tražimo prilikom optimizacije jest **globalni minimum** funkcije koštanja. Stopa učenja u tom slučaju govori nam koliki pomak prema globalnom minimumu treba napraviti u jednom koraku optimizacije. Na prethodnoj slici imamo vrlo jednostavno prikazanu funkciju koštanja koja ima samo jedan minimum i to

[3] Izvor: https://builtin.com/data-science/gradient-descent

globalni. U takvom slučaju ako vrijednost stope učenja postavimo prevelikom, globalni minimum bi se mogao preskočiti te bi svakom iteracijom čak mogli biti dalje od najboljeg rješenja. U suprotnom slučaju ako je stopa učenja premala bilo bi potrebno jako puno iteracija da se stigne do globalnog minimuma, odnosno najboljeg rasporeda težina u mreži koji bi doveo do najmanje ukupne greške modela. Naravno, funkcija koštanja često neće imati samo jedan minimum (i to globalni), već mogu postojati i lokalni minimumi, a kod premale stope učenja algoritam bi mogao stati u lokalnom minimumu što neće dovesti do minimalne moguće greške modela. Iz tog razloga potrebno je odrediti dovoljno dobru stopu učenja kod koje se neće događati ovakvi problemi. (Géron, 2017)

Ovim problemom u današnje vrijeme ne trebamo se previše baviti upravo iz razloga što postoje različite varijante GD algoritma koje često dinamički određuju stopu učenja, ali i određene druge parametre tako da učenje bude uvijek najoptimalnije moguće, no više o tome kasnije.

Kod osnovnog algoritma gradijentnog spusta korak optimizacije izvodi se jedanput u jednoj iteraciji (epohi) učenja, odnosno ažuriranje veza izvršava se tek nakon što se cijeli skup podataka propagira kroz neuronsku mrežu, izračuna funkcija koštanja te izračuna gradijentni vektor. Na ovakav način zapravo funkcionira osnovni GD algoritam, a u ovom kontekstu naziva se i „**Batch Gradient Descent“** (u nastavku skraćeno „BGD“)**.** Kod malih skupova podataka računanje gradijentnog vektora nije prezahtjevna operacija, no kod velikih to predstavlja problem, ne samo u pogledu memorijske kompleksnosti (cijeli skup podataka potrebno je učitati u radnu memoriju), već i vremenske kompleksnosti. Jedna trening epoha trajala bi jako dugo, zbog čega se koriste drugačije varijante gradijentnog spusta. (Géron, 2017)

7.1.1. Stochastic Gradient Descent

Jedna od osnovnih varijanti gradijentnog spusta jest i „**Stochastic Gradient Descent**“ (u nastavku skraćeno „SGD“). SGD algoritam, samim svojim imenom, upućuje na određeni „stohastički proces“, odnosno slučajnost koja se pojavljuje u njegovom radu. Problem sa „Batch Gradient Descent“ algoritmom koji se očituje u njegovom radu s velikim skupovima podataka rješava SGD algoritam. Osnovni princip rada SGD algoritma jest malo drugačiji. Kako BGD koristi cijeli skup podataka kod računanja gradijenata, što je izrazito računalno skup proces, SGD radi suprotno. Kod SGD algoritma uzima se instanca skupa podataka, propagira se kroz mrežu prema naprijed, računa gradijente te u istom koraku ažurira težine veza u mreži. Stohastičnost se očituje u slučajno odabranoj instanci skupa podataka za propagaciju kroz mrežu, odnosno instance koje ulaze u trening ne odabiru se po redu od prve do posljednje,

već se slučajno odabiru. Isto tako ne pamti se koje su instance odabrane pa se određene instance mogu više puta propagirati kroz mrežu, a neke nijednom.(Géron, 2017)

Ovaj pristup rješava problem s računanjem gradijenata nad velikim trening skupom podataka što uvelike smanjuje vremensku kompleksnost treniranja, ali sa manjim negativnim utjecajem na točnost treniranja. Naime, kod BGD-a, vrijednost funkcije koštanja tijekom iteracija glatko se spušta do minimuma gdje ostaje, dok kod SGD-a krivulja funkcije koštanja nije toliko glatka, već tijekom iteracija skače, ali u prosjeku pada prema minimumu gdje se zadržava „skačući“ oko minimuma. Ono što je problem jest da će kod SGD-a, po završetku treniranja, vrijednost funkcije koštanja biti blizu optimuma, ali ne i potpuno optimalna.(Géron, 2017)

S druge strane ova „neregularnost“ kod učenja algoritmom SGD ima jednu veliku prednost, a to je izlazak iz lokalnog minimuma. Spomenuto je da se BGD tijekom iteracija glatko spušta prema minimumu, no može se dogoditi da prvi minimum koji pronađe jest lokalni minimum. SGD zbog svoje neregularnosti ima veće šanse da preskoči lokalni minimum te pronađe globalni minimum.(Géron, 2017)

7.1.2.Mini-batch Gradient Descent

Posljednja osnovna varijanta algoritma gradijentnog spusta, i u praksi najkorištenija, jest takozvani **„Mini-batch Gradient Descent**“ (u nastavku skraćeno MGD). Iz samog naziva ovog algoritma moglo bi se utvrditi na koji način on radi. MGD algoritam kombinacija je algoritma „Batch Gradient Descent“ i „Stochastic Gradient Descent“. Prilikom treniranja, kod MGD-a, koriste se „mini batches“, odnosno manji slučajno odabrani podskupovi podataka iz cijelog trening skupa podataka. MGD radi na isti princip kao i SGD, ali umjesto da se svaka instanca skupa propagira kroz mrežu, računaju gradijenti i ažuriraju težine, propagiraju se manji podskupovi. Znači slučajno se odabire manji skup iz trening skupa, propagira se kroz mrežu, na temelju njega računaju se gradijenti te se ažuriraju težine u mreži i tako iterativno dok se prođe kroz cijeli skup podataka. MGD ima prednost pred SGD-om zbog još veće brzine treniranja, ali i regularniji je algoritam od SGD-a, odnosno MGD na kraju treniranja ostaje bliže minimumu od SGD-a. (Géron, 2017)

7.1.2.1. Upotreba GD u razvojnom okviru Tensorflow

Već smo utvrdili da BGD nije najbolji algoritam ponajviše iz razloga što bi treniranje ovim algoritmom trajalo predugo zbog čega SGD i MGD imaju prednost pogotovo kod većih skupova podataka. Naravno, ovisno o našim potrebama, kroz Tensorflow/Keras možemo iskoristiti različite optimizatore. Što se tiče osnovnog algoritma GD postoji samo optimizator

koji se zove „SGD“, što nam prema nazivu govori da je navedeni optimizator „Stochastic Gradient Descent“, no prilikom odabira dodatnih trening parametara možemo mijenjati takozvani „*batch_size*“. Do sada smo utvrdili da su BGD, SGD i MGD gotovo identični algoritmi s jedinom razlikom veličine skupa podataka koji se koristi u jednoj iteraciji povratne propagacije. Pa tako možemo odabirati „*batch_size*“ i na taj način iskoristiti sva tri tipa GD algoritma. Ako želimo BGD možemo „*batch_size“* postaviti na veličinu cijelog trening skupa podataka, za SGD „*batch_size*“ na 1, a MGD na neku proizvoljnu veličinu između 1 i veličine skupa podataka. Na koji bi način postavili ova tri tipa algoritma u Tensorflow/Keras-u možemo vidjeti na sljedećim slikama:

```
ann_sgd.compile(optimizer = 'SGD')
ann_sgd.fit(X_train, y_train, batch_size = len(X_train), epochs = 20)
```

Slika 15 "Batch Gradient Descent" u Tensorflow/Keras

```
ann_sgd.compile(optimizer = 'SGD')
ann_sgd.fit(X_train, y_train, batch_size = 1, epochs = 20)
```

Slika 16 „Stochastic Gradient Descent“ u Tensorflow/Keras

```
ann_sgd.compile(optimizer = 'SGD')
ann_sgd.fit(X_train, y_train, batch_size = 16, epochs = 20)
```

Slika 17 "Mini-batch Gradient Descent" sa batch_size=16 u Tensorflow/Keras

7.2. AdaGrad algoritam

Nakon pojašnjavanja rada osnovnog algoritma gradijentnog spusta krećemo na njegove, često u vidu performansi bolje, modifikacije/proširenja, a prvi je AdaGrad algoritam. Za AdaGrad algoritam mogli bi reći da je najjednostavnija modifikacija GD algoritma, a njegov naziv sam govori o kakvoj je modifikaciji riječ. Naziv AdaGrad dolazi od „Adaptive Gradient“ iz čega možemo izvući zaključak da se radi o određenoj adaptaciji. Adaptacija se očituje u adaptaciji stope učenja tijekom koraka optimizacije, a na koji način se to radi vrlo je jednostavan.(Brownlee, 2021b)

Tijekom opisa osnovnog algoritma gradijentnog spusta zapisali smo formulu koraka optimizacije koja je sljedeća:

$$w_{t,i} = w_{t-1,i} - \eta * \frac{\partial C}{\partial w_{t-1,i}}$$

Ovdje smo spomenuli da se nova vrijednost težine veze računa na način da se od prethodne vrijednosti oduzme stopa učenja pomnožena sa parcijalnom derivacijom greške po

vrijednosti težine veze. Kod AdaGrad algoritma formula je gotovo identična sa jedinom razlikom vrijednosti stope učenja η:

$$w_{t,i} = w_{t-1,i} - \eta^{t,i} * \frac{\partial C}{\partial w_{t-1,i}}$$

Ono što direktno vidimo iz formule jest različita vrijednost stope učenja η kod svakog koraka optimizacije. Naime spomenuto jest da je AdaGrad algoritam adaptivni algoritam gdje se adaptivnost očituje u adaptivnoj stopi učenja. Vrijednost stope učenja prilikom svakog koraka optimizacije računa se posebno ovisno o težini veze koja se ažurira, a formula stope učenja je sljedeća:

$$\eta^i = \frac{\eta}{\sqrt{s} + \varepsilon}$$

$\boldsymbol{\eta}$ označava konstantnu stopu učenja koju definiramo prilikom postavljanja parametara algoritma kao i kod osnovne verzije gradijentnog spusta, najčešće 0.01 ili 0.001. $\boldsymbol{s}$ predstavlja kvadriranu sumu svih parcijalnih derivacija izračunatih u prethodnim koracima optimizacije, a $\boldsymbol{s}$ možemo zapisati kao:

$$s = \sum_{t=1}^{t-1} \left(\frac{\partial C}{\partial w_{t-1,i}} \right)^2$$

ε jednostavno označava parametar zaglađivanja, a njegova jedina svrha jest osiguravanje da se u formuli za η^i ne dijeli s nulom. (Brownlee, 2021b)(Géron, 2017)

Svrha ove adaptacije jest ubrzanje procesa optimizacije na način da se kod detekcije rijetkih značajki stopa učenja povećava, odnosno rijetke značajke će imati veći utjecaj na pomak prema globalnom minimumu, dok će kod čestih značajki stopa učenja biti manja.(Ruder, 2016)

Problem koji se javlja kod ovog algoritma jest da će kod većeg broja iteracija optimizacije suma kvadrata parcijalnih derivacija postati vrlo velika što direktno znači da će stopa učenja η^i pri velikom broju iteracija postati mala. Kako stopa učenja postaje sve manja i manja algoritam pri većem broju iteracija više ne omogućava dodatno učenje, a u slučaju da se još nije stiglo do optimalne vrijednosti funkcije koštanja, algoritam nikad neće stići do globalnog minimuma.(Ruder, 2016)

7.3. RMSProp

Spomenuti je problem AdaGrad algoritma gdje se kod većeg broja iteracija stopa učenja znatno smanjuje što onemogućava neuronskoj mreži dodatno učenje. Navedeni

problem rješava algoritam RMSProp (skraćeno od „Root Mean Square Propagation“). Navedeni algoritam vrlo je sličan AdaGrad algoritmu sa jednom manjom, ali izrazito značajnom razlikom. Naime, RMSProp za razliku od AdaGrad algoritma ima drugačiji pristup računanju varijable $\boldsymbol{s}$ prilikom definiranja varijabilne stope učenja. AdaGrad radi na način da vrijednost varijable $\boldsymbol{s}$ uvećava za eksponencijalno opadajući kvadrirani prosjek gradijenata u svakoj iteraciji optimizacije, što ujedno znači da se stopa učenja s vremenom sve više smanjuje. RMSProp računa varijablu $\boldsymbol{s}$ na način da se najnovijim parcijalnim derivacijama daje najveće značenje, odnosno računa se „pomični prosjek“ pomoću čega se veće značenje daje novijim parcijalnim derivacijama čime se daje fokus na trenutni prostor za pretraživanje rješenja.(Brownlee, 2021d)

Vrijednost varijable $\boldsymbol{s}$ možemo izračunati na sljedeći način:

$$s_t = \beta * s_{t-1} + (1-\beta) * \left(\frac{\partial C}{\partial w_{t-1,i}}\right)^2$$

Sa β označavamo takozvanu „stopu opadanja“ (engl. „*decaying rate*“) koja je najčešće postavlja na vrijednost 0.9, s s_{t-1} označavamo vrijednost $\boldsymbol{s}$ u prethodnoj iteraciji optimizacije, a $\frac{\partial C}{\partial w_{t,i}}$ parcijalnu derivaciju funkcije koštanja po težini veze i u trenutnom promatranom trenutku. Vrijednosti nove težine i stope učenja računaju se na identičan način opisan u prethodnim poglavljima, sa jedinom razlikom drugačijeg računanja varijable $\boldsymbol{s}$**.** (Géron, 2017)

Iz svega opisanog očekivani su puno bolji rezultati korištenjem ovog algoritma od AdaGrad-a ponajviše zbog boljeg izračuna varijabilne stope učenja u svakom koraku optimizacije koja u ovom slučaju ne bi smjela postati toliko niska da se prestane sa učenjem prije postizanja globalnog minimuma.

7.4. Adam

Kroz prethodna poglavlja opisane su dvije modifikacije algoritma gradijentnog spusta koje su temelj jednog od najboljih algoritama učenja neuronskih mreža naziv „*Adam*“. „*Adam*“, odnosno njegov puni naziv „*Adaptive Moment Estimation*“, smatra se trenutno najboljim algoritmom, ne samo zbog brzine kojom konvergira prema minimumu funkcije koštanja, već i točnosti, odnosno konvergenciji prema globalnom, a ne lokalnom minimumu. Algoritam „*Adam*“ kombinacija je dvije vrste algoritama, „RMSProp“ i optimizacije gradijentnog spusta momentom (engl. „*Gradient Descent with Momentum*“).

Optimizacija gradijentnog spusta momentom nije dosad spominjana u ovom radu, no ovaj algoritam je često korišten umjesto osnovnog algoritma gradijentnog spusta, no još češće

u sklopu algoritma „*Adam*“. Optimizacija momentom ubrzava proces traženja na način da se prilikom traženja rješenja, algoritam gradijentnog spusta usmjerava upravo prema globalnom minimumu. Ovaj algoritam u kombinaciji sa „RMSProp“ algoritmom čini sam algoritam „*Adam*“.(Brownlee, 2021c)

„*Adam*“ funkcionira na način da prati prvi moment oko sredine gradijenata (eksponencijalno opadajući prosjek gradijenata, varijabla **m**) i drugi moment gradijenata (eksponencijalno opadajući kvadrirani prosjek gradijenata, varijabla **s**).(Brownlee, 2021a)

Formula ažuriranja težina vrlo je slična već opisanim algoritmima sa jedinom razlikom uvođenja varijable korigiranog prvog i drugog momenta:

$$w_{t,i} = w_{t-1,i} - \frac{\eta * \hat{m}}{\sqrt{\hat{s}} + \varepsilon}$$

Kao i do sad w_i predstavlja težinu veze koju treba ažurirati, η početnu, konstantnu, stopu učenja, a ε predstavlja parametar zaglađivanja koji onemogućava dijeljenje s nulom. $\hat{m}$ i $\hat{s}$ predstavljaju korigirani prvi i drugi moment, a računaju se na sljedeći način:

$$\hat{m} = \frac{m_t}{1 - \beta_1^T}$$

$$\hat{s} = \frac{s_t}{1 - \beta_2^T}$$

Varijable **m** i **s** na početku treninga inicijaliziraju se na nulu, ali i u prvim iteracijama biti će blizu nule, zbog čega ih je potrebno korigirati na početku treninga. β_1 naziva se i „moment opadanja“ (engl. „*moment decay*“), a β_2 „stopa opadanja“ (engl. „*decaying rate*“), dok T označava iteraciju treninga. Varijabla s (drugi moment) računa se na identičan način kao i kod „*RMSProp*“ algoritma, osim što u ovom slučaju stopu opadanja β označavamo β_2:

$$s_t = \beta_2 * s_{t-1} + (1 - \beta_2) * \left(\frac{\partial C}{\partial w_{t-1,i}}\right)^2$$

Prvi moment m računa se na sličan način:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \frac{\partial C}{\partial w_{t-1,i}}$$

β_1 i β_2 su konstantne vrijednosti koje se na inicijalizaciji treniranju definiranju najčešće kao $\beta_1 = 0.9$ i $\beta_2 = 0.999$. (Géron, 2017)

Iako je spomenuto da je algoritam „Adam“ trenutno jedan od najboljih algoritama učenja, postoje još dva algoritma koji proširuju ovaj algoritam te često daju čak i bolje rezultate.

7.5. AdaMax

„*AdaMax*“ algoritam jedna je od varijanti „*Adam*“ algoritma te je sam njegov rad gotovo identičan „*Adam*“ algoritmu uz jedinu razliku računanja varijable s_t. Kod ovog algoritma s_t računa se na sljedeći način:

$$s_t = \max\left(\beta_2 * s_{t-1}, \left|\frac{\partial C}{\partial w_{t-1,i}}\right|\right)$$

Vrijednost varijable s_t poprima maksimalnu vrijednosti između umnoška vrijednosti varijable s u prethodnoj iteraciji ažuriranja težine veze sa stopom opadanja β_2 i apsolutne vrijednosti parcijalne derivacije funkcije koštanja po težini veze. Isto tako nije potrebno odrađivati korekciju vrijednosti već se u formulu ažuriranja težine veze uzima vrijednosti s_t umjesto $\hat{s}$. U određenim slučajevima ovaj algoritam, sa ovakvom malom razlikom od „*Adam*“ algoritma može dati nešto bolja rješenja zbog konvergencije prema točnijoj vrijednosti minimuma funkcije koštanja. (Ruder, 2016)

7.6. NAdam

Druga varijanta „*Adam*“ algoritma naziva se „*NAdam*“ kao skraćenica od „*Nestorov Adaptive Moment Estimation*“. Ovaj algoritam koristi takozvanu „*NAG*“ ili „*Nestorov Accelerated Gradient*“ metodu računanja prvog momenta (varijable m odprije). „*NAG*“ se pokazao kao bolja metoda računanja momenta jer se ne koristi gradijent funkcije koštanja na trenutnoj vrijednosti težine veze, već za težinu uvećanu za $\beta * m$. Računanje momenta na taj način pokazao se bolji , odnosno pokazalo se da algoritam konvergira do minimuma funkcije koštanja znatno brže računanjem momenta pomoću „*NAG*“ metode.(Géron, 2017)

Ovu modifikaciju „*Adam*“ algoritma izvršio je Timothy Dozat, a u svom radu naziva „*INCORPORATING NESTEROV MOMENTUM INTO ADAM*“ opisao je način na koji „*Adam*“ funkcionira koristeći „*NAG*“ metodu izračuna momenta. Završna formula razlikuje se od „*Adam*“ algoritma u koraku izračuna korigirane vrijednosti momenta $\widehat{m}$ (Dozat, 2016):

$$\widehat{m} = \frac{\beta_{1,t+1} * m_t}{1 - \prod_{i=1}^{t+1} \beta_{1,i}} + \frac{(1 - \beta_{1,t}) * \frac{\partial C}{\partial w_{t-1,i}}}{1 - \prod_{i=1}^{t} \beta_{1,i}}$$

Ovakav način izračuna momenta u teoriji bi trebao djelomično ubrzati proces konvergencije prema minimumu funkcije koštanja zbog čega bi „*NAdam*“ trebao biti nešto brži od „*Adam*“ algoritma.

8. Karakteristike skupova podataka

Do sada je kroz ovaj rad opisana teorijska podloga neuronskih mreža i opisan rad raznih algoritama čije će se performanse ispitati u nastavku ovog rada. Kroz opis algoritama mogli smo početi donositi zaključke o tome kakve će performanse određeni algoritmi imati, odnosno očekuje se da algoritmi „*Adam*“ i njegove modifikacije biti puno bolje od primjerice algoritma „*AdaGrad*“ i osnovnog algoritma gradijentnog spusta. U ovom poglavlju opisat će se kakve karakteristike skupovi podataka mogu imati, a utjecaj nekih od njih na performanse opisanih algoritama testirat će se u praktičnom djelu rada.

Karakteristike skupova podataka također nazivamo i meta-značajke. One opisuju razna svojstva skupova podataka koje mogu imati utjecaj na performanse algoritama učenja. Meta-značajke mogu biti razne, a prema (Rivolli et al., 2022) možemo ih organizirati u šest grupa: Jednostavne (osnovne), statističke, informacijsko-teorijske, bazirane na modelima, „Landmarking“ značajke i ostale.

8.1. Jednostavne značajke

Jednostavne ili osnovne značajke skupa podataka su, kao i što samo ime govori, najjednostavnije značajke koje opisuju neki skup podataka. One uključuju broj atributa skupa podataka, broj instanci u skupu podataka, broj atributa prema tipu podataka itd., a cijeli popis značajki i njihov opis možemo vidjeti u Tablici 1:

Tablica 1 Popis osnovnih meta-značajki

Naziv	Opis
attrToInst	Omjer broja atributa i instanci
catToNum	Omjer broja kategorijskih i numeričkih varijabli
classToAttr	Omjer broja klasa i atributa
freqClass	Frekvencija instanci po klasama
instToAttr	Omjer broja instanci i atributa
instToClass	Omjer broja instanci i klasa
nrAttr	Broj atributa
nrBin	Broj binarnih atributa
nrCat	Broj kategorijskih atributa
nrClass	Broj klasa
nrInst	Broj instanci
nrInstMissing	Broj instanci s vrijednostima koje nedostaju

nrNum	Broj numeričkih atributa
numToCat	Omjer numeričkih i kategorijskih atributa

Kao što možemo vidjeti iz tablice ovih desetak značajki daju vrlo jednostavan, ali isto tako možda i najvažniji opis svakog skupa podataka. Primjerice, značajka nrInst nam govori koliko je velik skup podataka. Poznato je da neki algoritmi strojnog učenja ne mogu dobro raditi sa velikim skupovima podataka, ponajviše zbog vremena treniranja koje može biti vrlo dugo. Isto tako bilo bi dobro ispitati algoritme neuronskih mreža postoji li razlika među njima ovisno o veličini skupa podataka nad kojim se izvodi treniranje. Značajke instToClass i freqClass nam mogu reći puno o klasnoj nebalansiranosti skupa podataka, gdje mogu postojati skupovi podataka sa jednom klasom koja dominira nad drugim klasama prema broju instanci. (Rivolli et al., 2022)

Značajke koje se odnose na vrijednosti koje nedostaju u skupu podataka daju uvid u kvalitetu skupa podataka, odnosno neki algoritmi mogu i sa velikim brojem podataka koji nedostaju pronaći uzorke i dati veću točnost u predviđanju u odnosu na druge algoritme. (Rivolli et al., 2022)

U praktičnom dijelu rada najviše će biti govora o ovim značajkama, odnosno skupovi podataka nad kojima će se raditi testiranja biti će različitih veličina, tipova podataka, kvalitete, ali i vrste varijable koja se treba predvidjeti, a samim time i tipova problema.

8.2. Statističke meta-značajke

Druga vrlo važna grupa meta-značajki su statističke značajke. Ove meta-značajke daju informacije o distribuciji podataka, mjerama centralne tendencije, disperzije podataka, korelacije među podacima,… Važno je napomenuti da su statističke metrike specifične za atribute s numeričkim vrijednostima te ne mogu raditi s kategorijskim atributima. Cijeli popis možemo vidjeti u Tablici 2 (Rivolli et al., 2022):

Tablica 2 Popis statističkih meta-značajki

Naziv	**Opis**
canCor	Kanonska korelacija
cor	Korelacija
cov	Kovarijanca
nrDisc	Broj diskriminantnih vrijednosti
eigenvalues	Svojstvene vrijednosti matrice kovarijance
gMean	Geometrijska sredina atributa
hMean	Harmonijska sredina atributa
iqRange	Interkvartilni raspon
kurtosis	Kurtoza
mad	Srednje apsolutno odstupanje
max	Maksimalna vrijednost
mean	Srednja vrijednost
median	Medijan
min	Minimalna vrijednost
nrCorAttr	Broj parova atributa s visokom korelacijom
nrNorm	Broj atributa s normalnom distribucijom
nrOutliers	Broj atributa s outlier-ima
range	Raspon atributa
sd	Standardna devijacija atributa
sdRatio	Raspon
skewness	Test homogenosti kovarijanci
tMean	Obrezana srednja vrijednost atributa
var	Varijanca
wLambda	Wilkins Lambda test

Ovdje također imamo nekoliko meta-značajki koje opisuju skup podataka te koje mogu predvidjeti ponašanje određenih algoritama strojnog učenja. Primjerice nrOutliers nam govori koliko ima atributa s „outlier“ vrijednostima koji imaju često negativan utjecaj na točnost modela kod pojedinih algoritama. Osim toga korelacija i kovarijanca daju uvid u međuzavisnost atributa. Visoke vrijednosti korelacije nam govore da postoji redundantnost u podacima. (Rivolli et al., 2022) Ako imamo atribute koji međusobno imaju jako visoku korelaciju postoji mogućnost koristiti samo jedan od međusobno visokokoreliranih atributa prilikom treninga što bi kod velikih skupova podataka moglo ubrzati trening kod nekih algoritama. Isto tako ako imamo veći broj nezavisnih varijabli koje su u korelaciji s zavisnom varijablom, mogli bi očekivati veće točnosti u predikciji.

8.3. Informacijsko-teorijske meta-značajke

Kako statističke meta-značajke specifične za numeričke varijable, tako su informacijsko-teorijske specifične za kategorijske varijable. Informacijsko-teorijske meta-značajke opisuju varijabilnost i redundantnost nezavisnih varijabli koje utječu na zavisnu varijablu. Ovdje također imamo nekoliko meta-značajki koje označavaju, primjerice, čistoću skupa podataka kao što su entropija klasa i atributa. (Rivolli et al., 2022) Cijeli popis meta-značajki koje pripadaju ovoj grupi opisane su u Tablici 3:

Tablica 3 Popis informacijsko-teorijskih meta-značajki

Naziv	**Opis**
attrEnt	Entropija atributa
classEnt	Entropija klase
eqNumAttr	Ekvivalentni broj atributa
jointEnt	Zajednička entropija
mutInf	Zajednička informacija
nsRatio	Omjer šuma i signala

8.4. Meta-značajke bazirane na modelima

Sljedeće dvije grupe meta-značajki malo se razlikuju od prethodno opisnih, a odnose se na meta-značajke koje se ekstrahiraju iz rezultata jednostavnih prediktivnih modela. U slučaju ove grupe meta-značajki opisane su meta-značajke odnosi na model izrađen pomoću stabla odlučivanja. Iako se možda na pravi pogled ne vidi značenje ovih meta-značajki, one nam mogu puno reći o kompleksnosti skupa podataka. Popis svih meta-značajki baziranih na modelima stabla odlučivanja možemo vidjeti u sljedećoj tablici:(Rivolli et al., 2022)

Naziv	**Opis**
leaves	Broj „listova“ u „stablu“
leavesBranch	Broj „grana“ u „stablu“
leavesCorrob	Omjer broja instanci i „listova“
leavesHomo	Distribucija „listova“
leavesPerClass	Omjer „listova“ i klasa
nodes	Broj čvorova u „stablu“
nodesPerAttr	Omjer broja čvorova i atributa
nodesPerInst	Omjer broja čvorova i instanci
nodesPerLevel	Broj čvorova po razinama „stabla“
nodesRepeated	Broj ponavljajućih čvorova
treeDepth	Dubina „stabla“
treeImbalance	Nebalansiranost „stabla“

treeShape	Oblik „stabla“
varImportance	Važnost atributa

Jedna od najvažnijih meta-značajki koju možemo dobiti iz ove grupe meta-značajki bila bi *varImportance* odnosno važnost atributa. Izrada modela stabla odlučivanja puno je brža od primjerice modela neuronske mreže. U slučaju ekstrakcije ovih meta-značajki, prije detaljnije analize skupa podataka, možemo na vrlo jednostavan i brzi način dobiti važne informacije o atributima na temelju kojih će se graditi model neuronske mreže.

8.5. „Landmarking“ i ostale meta-značajke

„Landmarking“ meta-značajke zapravo su meta-značajke koje koriste performanse raznih brzih i jednostavnih algoritama strojnog učenja. One se razlikuju od meta-značajki baziranih na modelima u tome što se ne gledaju informacije izvučene iz modela (npr. izgled i struktura istreniranog stabla), već performanse algoritama tipa točnosti, greške, itd. (Rivolli et al., 2022) Ove meta-značajke mogu nam okvirno reći kakve performanse možemo očekivati od modela neuronske mreže.

Naravno, postoje meta-značajke koje ne možemo svrstati u opisanih nekoliko kategorija ili se ne koriste u širokoj upotrebi. U kategoriju „Ostale meta-značajke“ mogu spadati meta-značajke koje se odnose na točno određenu domenu kao što su primjerice značajke koje se odnose na „unsupervised“, odnosno tehnike nenadziranog strojnog učenja. U tu grupu primjerice spadaju meta-značajke bazirane na tehnici nenadziranog učenja klasteriziranja. (Rivolli et al., 2022)

9. Opis korištenih metrika

Kako bi mogli usporediti rezultate svih algoritama potrebno je odrediti skup metrika, naravno ovisno o problemu. Izrađene su različite funkcije računanja metrika ovisno o tipu problema, a u nastavku će biti opisane implementirane funkcije.

9.1. Metrike za probleme klasifikacije

Što se tiče problema klasifikacije koriste se ukupna točnost predviđanja (Accuracy), prostor ispod ROC krivulje (ROC AUC), matrica konfuzije (Confusion matrix), odaziv (Recall), preciznost (Precision), F1 mjera (F1 score) i vrijeme treniranja modela. Ove metrike koriste se nakon svakog treniranja modela, a za potrebe brzog pregleda izrađena je sljedeća funkcija.

```
def metrike(modeli, testData, y_test, time):
    for ind, model in enumerate(modeli):
        print(F"Algoritam: {type(model.optimizer).__name__}")
        #ako je ANN
        y_pred = model.predict(testData)
        y_pred = (y_pred > 0.5)
        print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
        print('ROC AUC:', metrics.roc_auc_score(y_test, y_pred))
        print("Confusion matrix")
        CM = metrics.confusion_matrix(y_test, y_pred)
        TN = CM[0, 0]
        TP = CM[1, 1]
        FP = CM[0, 1]
        FN = CM[1, 0]
        print("    P0          P1")
        print(f"S0  {TN}        {FP}")
        print(f"S1  {FN}        {TP}")
        print("Recall: ",metrics.recall_score(y_test,y_pred))
        print("Precision: ",  metrics.precision_score(y_test, y_pred))
        print("F1 score: ", metrics.f1_score(y_test, y_pred))
        print(f"Training time (sec): {time[ind]}" )
        print("------------------------------------------------")
```

Slika 18 Funkcija za računanje raznih metrika

Kao što možemo vidjeti iz priloženog, funkcija za parametre prima skup modela, testni skup podataka, stvarne očekivane vrijednosti zavisne varijable testnog skupa i vremena trajanja treniranja modela. Na početku se izvode predviđanja svakog istreniranog, proslijeđenog modela. Predviđene vrijednosti su u rasponu od 0 do 1 (zbog funkcije „sigmoid“ na izlaznom neuronu) pa u varijablu „*y_pred*“ spremaju vrijednosti 0 ili 1 ovisno o tome da li je vrijednost predviđanja manja od 0.5 ili veća. Nakon toga koriste se funkcije iz ML biblioteke „scikit-learn“ koje računaju sve navedene metrike klasifikacije.

Najosnovnija metrika kod klasifikacije jest točnost. Točnost je zapravo omjer svih točnih klasifikacija unutar svih ukupnih klasifikacija modela. Kada imamo klasnu balansiranost u skupu podataka ovo je svakako jedna od najvažnijih metrika. (Agarwal, 2019) Kod nebalansiranih skupova podataka točnost nam ne daje pravi pogled na performanse modela, a primjer toga vidjet ćemo na jednom skupu podataka koji će se analizirati u nastavku. ROC AUC računa se kao prostor ispod tzv. „Receiver operating characteristic“ krivulje koja u obzir uzima vrijednosti mjera TPR („True Positive Rate“) i FPR („False Positive Rate“) na raznim razinama pragova. ROC AUC daje bolji pogled na ukupne performanse modela, osobito kod nebalansiranih skupova podataka. (Ng, n.d.)

Matrica konfuzije nam daje podatke o konkretnom broju ispravnih i neispravnih klasifikacija, odnosno u matrici dobivamo vrijednosti ispravnih pozitivnih klasifikacija (True Positives), ispravnih negativnih klasifikacija (True Negatives), neispravnih pozitivnih klasifikacija (False Positives) i neispravnih negativnih klasifikacija (False Negatives). Preko ovih vrijednosti možemo doći do vrlo važnih metrika odaziva i preciznosti. Preciznost nam govori koji je omjer predviđenih pozitivnih vrijednosti u ukupnom broju pozitivnih predikcija (ispravnih i neispravnih), odnosno koliko je model precizan u otkrivanju pozitivnih vrijednosti klasifikacije. Odaziv je omjer ispravnih pozitivnih klasifikacija i sume ispravnih pozitivnih i krivih negativnih klasifikacija, odnosno odaziv nam govori koliko je model osjetljiv na detekciju pozitivnih instanci u skupu podataka. F1 mjera daje ukupne performanse modela uzimajući u obzir ove dvije metrike te također daje puno bolje vrijednosti ukupnih performanse kod nebalansiranih skupova podataka od ukupne točnosti. (Ng, n.d.)

Vrijeme treniranja modela nije standardna metrika analize performansi modela, no u ovom radu biti će jedna od najvažnijih iz razloga što različiti algoritmi imaju različita ukupna vremena treniranja. Preko ove mjere usporedit će se koliko je bilo potrebno određenom algoritmu da dođe do svojeg minimuma funkcije gubitka.

Osim ove funkcije za izračun metrika izrađena je još jedna funkcija koja u obliku „Pandas Dataframe-a“ prikazuje sve navedene evaluacijske metrike modela u jednoj preglednoj tablici. U ovoj tablici dodatno prikazujemo vrijeme testiranja i broj epoha koji je bio potreban određenom algoritmu da dođe do minimuma svoje funkcije gubitka.

```python
def metrike_table(modeli, testData, y_test, tr_time, histories):
    data = []
    for ind, model in enumerate(modeli):
        start = time.time()
        y_pred = model.predict(testData)
        y_pred = (y_pred > 0.5)
        test_time = time.time()-start
        data.append([
            type(model.optimizer).__name__,
            metrics.accuracy_score(y_test, y_pred),
            metrics.roc_auc_score(y_test, y_pred),
            metrics.recall_score(y_test,y_pred),
            metrics.precision_score(y_test, y_pred),
            metrics.f1_score(y_test, y_pred),
            tr_time[ind],
            test_time,
            len(histories[ind].history['loss'])
            ])
        acc_loss_plot(histories, model, ind)
    df = pd.DataFrame(data, columns = ['Algoritam', "Accuracy",'ROC AUC',"Recall","Precision","F1 score", "Training time (sec)","Test time (sec)",'Epochs'])
    return df
```

Slika 19 Funkcija za prikaz metrika u Pandas Dataframe i krivulja točnosti i gubitka

Osim toga u ovoj funkciji poziva se funkcija za crtanje krivulja funkcija gubitka i validacijskog gubitka zajedno sa krivuljama trening točnosti i validacijske točnosti po epohama pomoću biblioteke „Matplotlib“. Analiza ovih krivulja također će nam dati vrlo važne poglede u tijek treniranja modela različitim algoritmima.

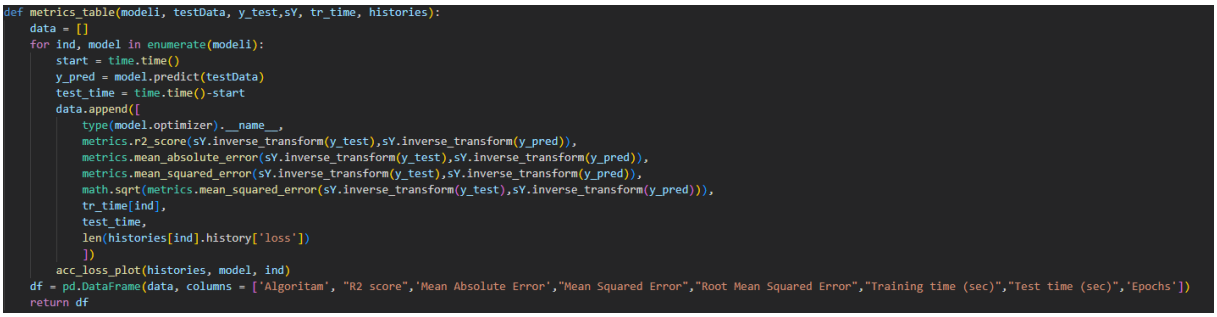
```python
def acc_loss_plot(histories, model, ind=0):
        plt.plot(pd.DataFrame(histories[ind].history["accuracy"]))
        plt.plot(pd.DataFrame(histories[ind].history["val_accuracy"]))
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.title(f'{type(model.optimizer).__name__} accuracy')
        plt.figure(figsize=(6,6), dpi=500)
        plt.show()

        plt.plot(pd.DataFrame(histories[ind].history["loss"]))
        plt.plot(pd.DataFrame(histories[ind].history["val_loss"]))
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.title(f'{type(model.optimizer).__name__} loss')
        plt.figure(figsize=(6,6), dpi=500)
        plt.show()
```

Slika 20 Funkcija za crtanje grafova

9.2. Metrike za regresijske probleme

Kod regresijskih problema izlazna vrijednost neuronske mreže jest kontinuirana numerička varijabla pa metrike opisane kod klasifikacijskih problema ne možemo primijeniti na ovakve probleme. Iz toga razlog evaluacija će se odvijati pomoću nekoliko drugačijih metrika, a to su koeficijent determinacije (R2 mjera), prosječna apsolutna greška (Mean Absolute Error/MAE), prosječna kvadrirana greška (Mean Squared Error/MSE) i korjenovana prosječna kvadrirana greška (Root Mean Squared Error/RMSE). Kao i kod problema klasifikacije u analizi performansi značajnu ulogu ima i vrijeme treniranja modela.



Slika 21 Funkcija za prikaz ukupnih metrika kod regresijskih problema

Sve ove mjere temelje se na računanju razlika između stvarne vrijednosti i predviđene vrijednosti zavisne numeričke varijable. Prosječna kvadrirana greška u obzir uzima kvadrirane greške, odnosno razlike između stvarne i predviđene vrijednosti. Na taj način veće razlike imaju veliki utjecaj na ukupnu vrijednost greške od manjih razlika. MSE se također često koristi kao funkcija gubitka kod učenja neuronskih mreža. (Brownlee, 2021f)

Korjenovana prosječna kvadrirana greška (RMSE) jest ono što i njezino ime govori, vrijednost MSE nad kojom je primijenjeno korjenovanje. Kako sama vrijednost nije baš u povoljnom obliku za interpretaciju, korjenovanje za posljedicu daje vrijednost MSE u obliku jedinice tražene varijable. (Brownlee, 2021f) Primjerice, ako vrijednosti tražene varijable u kilogramima, RMSE će dati prosječnu kvadriranu vrijednost u kilogramima.

Treća metrika grešaka je prosječna apsolutna greška (MAE). MAE je zapravo suma apsolutnih vrijednosti razlika stvarnih i predviđenih vrijednosti zavisne varijable podijeljena sa ukupnim brojem instanci grešaka. Ova je greška također izražena u obliku originalne jedinice tražene zavisne varijable te ju je najlakše interpretirati od opisanih pošto zapravo uzima prosjek apsolutnih razlika stvarnih i predviđenih vrijednosti.(Brownlee, 2021f)

Sve opisane metrike želimo minimizirati, dok koeficijent determinacije želimo maksimizirati. Koeficijent determinacije govori koliki je udio varijacije podataka oko aritmetičke sredine opisan regresijskim modelom te je u rasponu od 0 do 1.(Dobša, 2021) Model je bolji što je R2 bliži 1, a vrijednosti veće od 0.8 (80%) pokazuju da je model vrlo dobar.

10. Analiza skupova podataka

U ovom poglavlju analizirat će se svi odabrani skupovi. Skupovi su različitih karakteristika posebice veličina, tipova podataka i problema. Ispitat će se šest skupova podataka koristeći algoritme „*Adam*“, „*Adamax*“, „*Adagrad*“, „*Nadam*“, „*SGD*“ i „*RMSProp*“.

10.1. Breast Cancer detection

10.1.1. Opis domene

Prvi skup podataka koji će se analizirati jest skup koji se odnosi na predviđanje raka dojke kod žena. U današnje vrijeme bolest koja pogađa sve više ljudi ima mogućnosti prevencije i predviđanja u ranom stadiju pomoću tehnologija strojnog učenja. Podaci za ovaj skup podataka izračunati su iz slika dobivenih FNA postupkom, a iz samih podataka cilj je odrediti tip raka (maligni/benigni). Skup podataka preuzet je sa UCI ML repozitorija.[4]

10.1.2. Analiza i čišćenje skupa podataka

Skup podataka sastoji se od 32 atributa i 562 instance. Ekstrakcija osnovnih meta-značajki izrađena je pomoću alata „*pymfe*“ (Python Meta-Feature Extractor[5]) koji omogućava ekstrakciju raznih meta-značajki skupova podataka uključujući od prije opisanih osnovnih, statističkih, informacijsko-teorijskih, baziranih na modelima te „Landmarking“ značajki.

Navedena 32 atributa izvedena su iz deset osnovnih značajki izvedenih iz slika staničnih jezgri dobivenih FNA postupkom:

- Radius
- Texture
- Perimeter
- Area
- Smoothness
- Compactness
- Concavity
- Concave points
- Symmetry
- Fractal dimension

Svaka od ovih deset značajki izračunata je na temelju deset prikupljenih vrijednosti iz staničnih jezgri osobe, a sami finalni skup podataka sadrži prosjek, standardnu grešku i „najgoru“ prikupljenu vrijednost za svaku značajku što čini trideset atributa finalnog skupa podataka. Skup podataka također sadrži ID instance i vrijednost tražene klase koju želimo predvidjeti (maligni/benigni tumor).

---

[4] Poveznica na skup podataka: https://archive-beta.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+diagnostic

[5] Poveznica na „*pymfe*“:https://pypi.org/project/pymfe/

Sam broj atributa od 32 čini ovaj skup visokodimenzionalan, dok broj instanci od 569 čini skup vrlo mali. Isto tako nezavisne varijable u skupu podataka su kontinuirane numeričke varijable, dok je zavisna varijabla kategorijska varijabla. Navedene informacije dobivene su pomoću „*pymfe*“ alata, dok će se detaljna ekstrakcija preostalih karakteristika opisati u nastavku.

Na početku su učitane važne biblioteke koje će se biti potrebne za analizu skupova i treniranje modela.

Učitavanje biblioteka

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
import time
```

✓ 2.7s

Slika 22 Učitavanje potrebnih biblioteka

Sljedeći korak bio je učitavanje skupa podataka i prikaz prvih nekoliko redaka informacija. Na prvi pogled vidimo da su atributi pretežito numeričke varijable.

Učitavanje skupa podataka

```
dataset = pd.read_csv('../datasets/breast_cancer_wisconsin.csv')
dataset.head()
```

✓ 0.6s

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20

Slika 23 Učitavanje skupova podataka

Analiza deskriptivne statistike također je važan element analize skupa podataka iz kojeg možemo dobiti vrlo važne informacije o skupu.

```
dataset.describe()
```

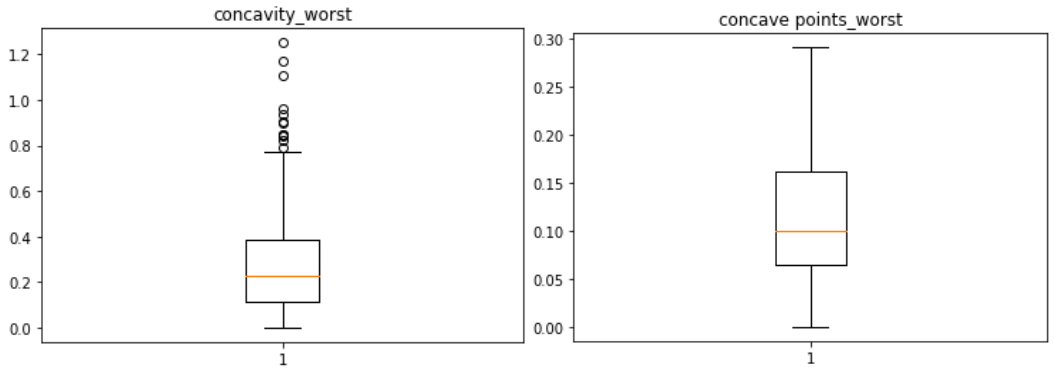
✓ 0.9s

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000

Slika 24 Deskriptivna statistika skupa podataka

Uočeno je da broj vrijednosti za sve atribute iznosi 569 što znači da nema vrijednosti koje nedostaju. To ćemo potvrditi na koraku čišćenja skupa podataka. Vrijednosti koje nedostaju, zajedno sa takozvanim „outlier“ vrijednostima mogu donijeti šum u skup podataka.

Detekcija „outlier“ vrijednosti može se izvesti pomoću „boxplot-ova“. Pomoću njih možemo vidjeti koliko vrijednosti odstupaju od većine podataka unutar interkvartilnog raspona. U ovom slučaju kod svih atributa osim „concave_points_worst“ postoje outlieri.



Slika 25 Analiza outlier vrijednosti

Na „boxplot-u“ za atribut „concavity_worst“ vidimo da se određeni broj instanci nalazi iznad gornjeg limita. Te točke mogle bi označavati ekstremne vrijednosti odnosno „outlier-e“. U ovakvom skupu ista pojava je za sve atribute. Kako atributi nisu najpovoljniji za interpretaciju, ne možemo biti u potpunosti sigurni da se ovi outlieri odnose na krive unose u skup podataka koji bi narušili kvalitetu skupa podataka zbog čega se ove instance neće izbacivati iz već dovoljno malog skupa podataka. Isto tako manji šum u podacima može otežati algoritmu „prenaučenost“(Brownlee, 2018)

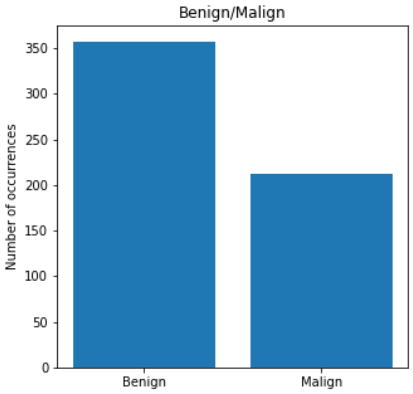
Važna karakteristika skupa podataka jest i balansiranost skupa podataka. Iz tog razloga potrebno je prikazati koja je distribucija dviju klasa koje želimo predvidjeti.

```
dataset['diagnosis'].value_counts()['B']/len(dataset)*100
#62.74% instanci klase Benign
```

✓ 0.4s

```
62.741652021089635
```

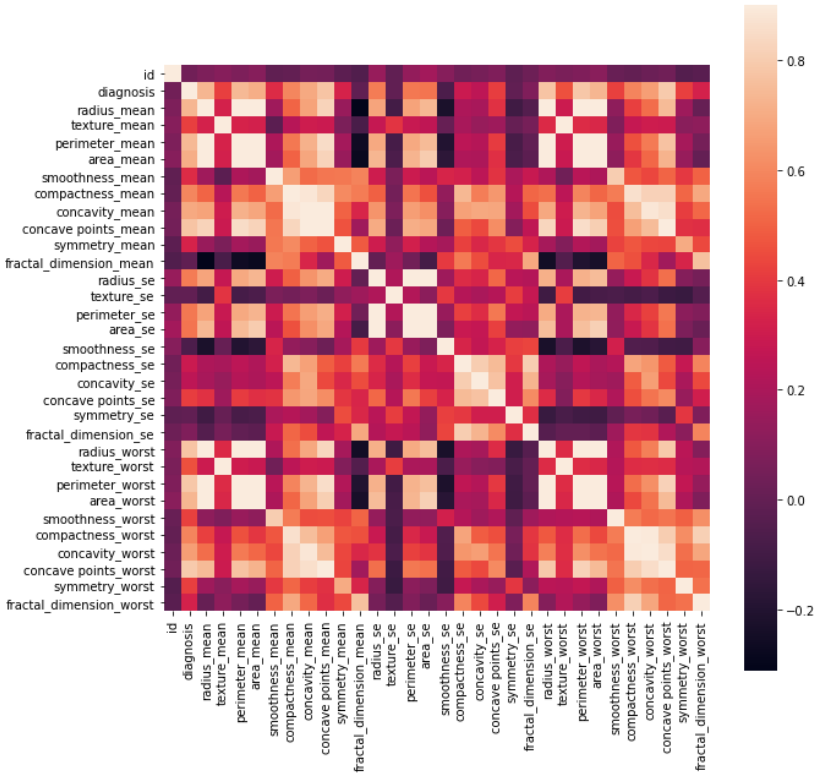
Slika 26 Analiza nebalasniranosti skupa podataka



Slika 27 Distribucija klasa

Klasa „Benign“ čini 62.74% skupa podataka. Mogli bi reći da postoji manja klasna nebalansiranost.

Također važan korak analize skupa podataka jest i analiza korelacija između nezavisnih varijabli i zavisne varijable.



Slika 28 Matrica korelacija

Najveću pozitivnu korelaciju na našu zavisnu varijablu „diagnose“ čini se da ima atribut „concave_points_worst“, odnosno što je veća vrijednost ove varijable, veća je šansa da će dijagnoza biti „maligni tumor“. Najveću negativnu korelaciju s dijagnozom ima atribut „smoothness_se“, odnosno, što je veća vrijednost ove varijable, manje su šanse da će dijagnoza biti „maligni tumor“.

Možemo iščitati vrijednosti korelacija svih varijabli s varijablom „diagnose“;

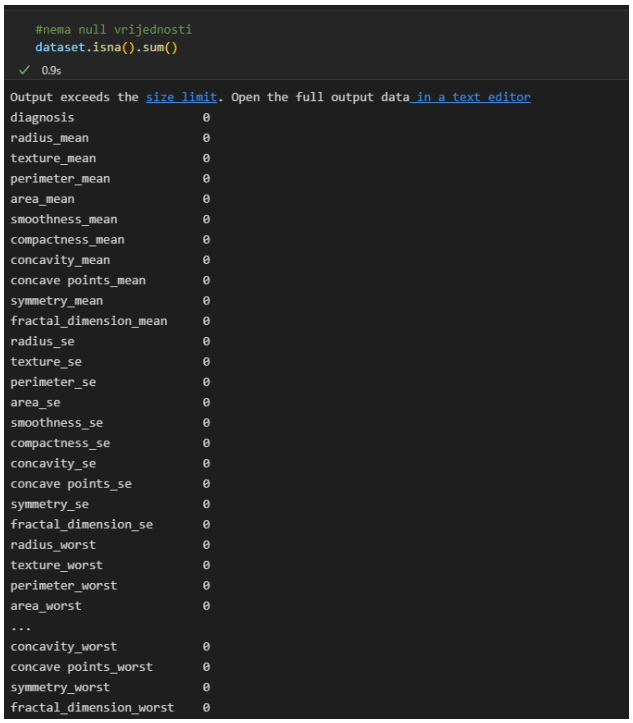
	diagnosis		
id	0.039769	smoothness_se	-0.067016
radius_mean	0.730029	compactness_se	0.292999
texture_mean	0.415185	concavity_se	0.253730
perimeter_mean	0.742636	concave points_se	0.408042
area_mean	0.708984	symmetry_se	-0.006522
smoothness_mean	0.358560	fractal_dimension_se	0.077972
compactness_mean	0.596534	radius_worst	0.776454
concavity_mean	0.696360	texture_worst	0.456903
concave points_mean	0.776614	perimeter_worst	0.782914
symmetry_mean	0.330499	area_worst	0.733825
fractal_dimension_mean	-0.012838	smoothness_worst	0.421465
radius_se	0.567134	compactness_worst	0.590998
texture_se	-0.008303	concavity_worst	0.659610
perimeter_se	0.556141	concave points_worst	0.793566
area_se	0.548236	symmetry_worst	0.416294
		fractal_dimension_worst	0.323872

Slika 29 Vrijednosti korelacija s varijablom dijagnoze

Vidimo da ima veliki broj varijabli s većom pozitivnom (iznad 0.5) korelacijom s varijablom dijagnoza.

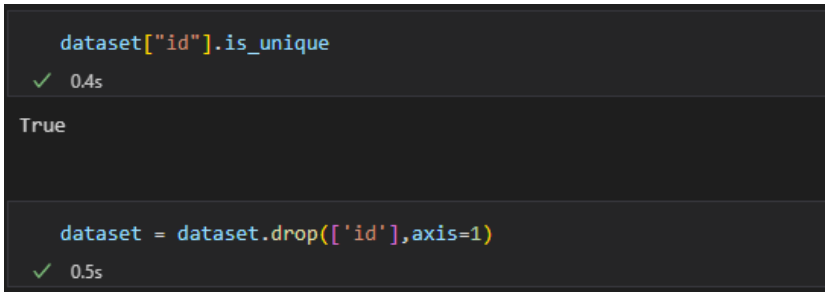
Iz ove analize možemo zaključiti da je skup podataka visokodimenzionalan, ali prema proju instanci malen, dok su sve nezavisne varijable numeričkog tipa. Skup sadrži malu količinu šuma u obliku „outlier“ vrijednosti te je djelomično balansiran (63%/37%). Iz matrice korelacija mogli smo vidjeti da postoji veliki broj atributa koji su u pozitivnoj korelaciji s zavisnom varijablom „diagnose“.

Što se tiče čišćenja skupa podataka provest ćemo nekoliko koraka. Prvi je osigurati da ne postoje vrijednosti koje nedostaju za sve atribute pošto ne možemo provesti treniranje sa takvim vrijednostima.



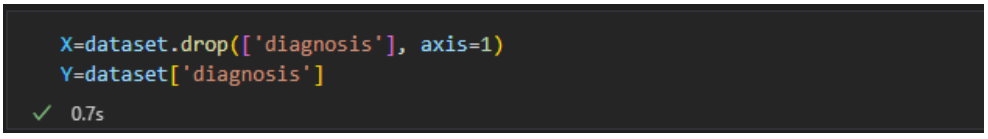
Slika 30 Analiza vrijednosti koje nedostaju

Možemo vidjeti da ne postoje vrijednosti koje nedostaju. Osim toga pretpostavka je da atribut „id“ sadrži različite vrijednosti koje nemaju nikakve prediktivne sposobnosti u ovom skupu podataka, odnosno samo označuju redoslijed instanci u skupu podataka. To možemo provjeriti naredbom „*is_unique*“ i izbaciti ovaj atribut.



Slika 31 Eliminacija varijable "id"

Posljednji korak jest podijeliti skup podataka na nezavisne i zavisne varijable. Skup vrijednosti nezavisnih varijabli označit će se sa X, a zavisna varijabla sa Y.



Slika 32 Raspodjela skupa na nezavisne i zavisnu varijablu

Prije izrade modela potrebno je podijeliti skup podataka na skup za treniranje i testiranje. U testni skup raspoređeno je 25% originalnog skupa podataka.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25, random_state=42)
✓ 0.6s
```

Slika 33 Podjela skupa na treniranje i testiranje

Nakon toga moramo standardizirati skup podataka. Algoritmi neuronskih mreža zahtijevaju standardizaciju skupova podataka zbog čega ovaj korak nužan je prije provođenja treniranja i to na nakon podjele skupa podataka na trening i test skupove. Kako bi trening i testni skupovi podataka trebali biti nezavisni (imati različite prosjeke i standardne devijacije) preporuka je provođenje standardizacije nakon podjele skupova podataka.

```
scX = StandardScaler()
scY = StandardScaler()
X_train = scX.fit_transform(X_train)
X_test = scX.transform(X_test)
✓ 0.3s
```

Slika 34 Skaliranje trening i test skupova

10.1.3. Postavke i treniranje

Nakon čišćenja i analize skupova podataka odredit će se struktura neuronskih mreža. Struktura će biti identična za sve algoritme kako bi mogli raditi usporedbe između algoritama.

Kako ne postoji univerzalno pravilo koje bi govorilo kakva struktura neuronskih mreža mora biti isprobane su različite strukture zajedno sa različitim aktivacijskim funkcijama. Najbolje performanse s prihvatljivim vremenom treniranja imala je struktura sa tri skrivena sloja sa po 8, 16 i 32 neurona zajedno sa aktivacijskom funkcijom „ReLU“. Kako imamo problem binarne klasifikacije na izlaznom sloju nalazi se jedan neuron sa aktivacijskom funkcijom „Sigmoid“. Ova aktivacijska funkcija daje nam mogućnosti dobivanja predviđene vrijednosti u rasponu od 0 do 1 (opisano u poglavlju „Aktivacijske funkcije“). Na taj način izlazna vrijednost neuronske mreže je u obliku vjerojatnosti te zapravo možemo vidjeti koliko je model siguran u svoje predviđanje. Izlazne vrijednosti blizu 1 govore da je model izrazito siguran da je izlazna vrijednost 1, a vrijednosti blizu 0 govore da je model siguran da je izlazna vrijednost 0. Što su vrijednosti blizu 0.5 to je model manje siguran u svoja predviđanja.

```
ann_adam = tf.keras.models.Sequential()
#3 skirvena sloja sa po 8, 16 i 32 neurona
ann_adam.add(tf.keras.layers.Dense(units=8, activation='relu'))
ann_adam.add(tf.keras.layers.Dense(units=16, activation='relu'))
ann_adam.add(tf.keras.layers.Dense(units=32, activation='relu'))
# Dodavanje izlaznog sloja
ann_adam.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
# kompajliranje mreže
ann_adam.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Slika 35 Postavljanje strukture mreže

Što se tiče funkcije gubitka odabrana je funkcija binarna unakrsna entropija (engl. „binary_crossentropy“). Kod problema binarne klasifikacije najčešće se koristi ova funkcija.(Ronaghan, 2018)

Kao što je spomenuto, prikazana struktura ista je za sve algoritme. Isto tako kod ovog skupa podataka veličina uzorka za treniranje („*batch_size*“) iznosi 16. Kako imamo manji skup podataka možemo si zbog manje vremenske i memorijske kompleksnosti dopustiti nešto manju veličinu uzorka koja se propagira kroz mrežu. Isto tako svi algoritmi trenirani su kroz različiti broj epoha, te je odabran optimalni broj epoha (kod kojeg je funkcija gubitka bila na minimumu i prije pojave „prenaučenosti“). U nastavku će se za svaki algoritam dati osnovne metrike zajedno sa grafovima funkcija gubitka i točnosti po epohama.

10.1.4. Algoritam Adam

Što se tiče algoritma „*Adam*“, broj epoha kojim je funkcija gubitka došla na svoj minimum bila je 10.

```
start = time.time()
adam_history = ann_adam.fit(X_train, y_train, batch_size = 16, epochs = 10, validation_data=(X_test,y_test))
adam_time = time.time()-start
```

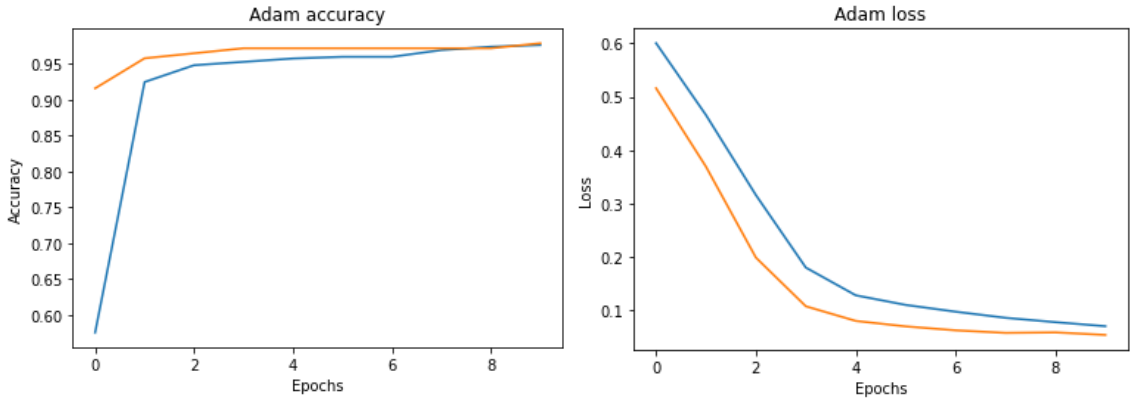
Slika 36 Pokretanje treniranja algoritmom Adam

Nakon 10 epoha učenja algoritam je došao do točnosti od vrlo visokih 97.902%. Isto tako vrlo visoki bio je i ROC AUC, a iznosi 97.586%. Od 143 validacijskih instanci, algoritam je sveukupno napravio tri pogreške, dvije instance označio je kao krive negative (FN) i jednu instancu kao krivi pozitiv (FP). To dovodi odaziv na 96.297%, a preciznost na 98.11%. F1 mjera zato iznosi vrlo visokih 97.196%. Algoritam je kroz 10 epoha prošao ispod jedne sekunde.

```
Algoritam: Adam
Accuracy:  0.9790209790209791
ROC AUC: 0.9758635039533915
Confusion matrix
    P0          P1
S0  88        1
S1  2        52
Recall:  0.9629629629629629
Precision:  0.9811320754716981
F1 score:  0.9719626168224299
Training time (sec): 0.6525924205780029
```

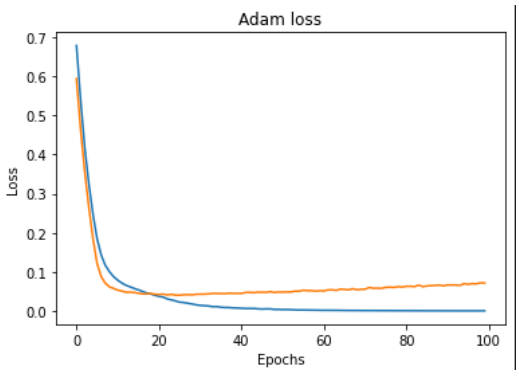
Slika 37 Rezultati testiranja Adam

Možemo analizirati i vrijednosti funkcije gubitka i točnost kroz epohe.



Slika 38 Točnost i vrijednost funkcije gubitka po epohama Adam

Validacijski gubitak i točnost označeni su sa narančastom bojom, a trening plavom. Vidimo da vrijednost funkcije gubitka pada sve do desete epohe. Uočeno je da povećanjem broja epoha validacijski gubitak počinje ponovno rasti, a trening stagnirati pa samim time validacijska točnost počinje padati. Ta pojava govori nam da se počinje javljati „prenaučenost“, odnosno algoritam se previše prilagođava trening podacima sve do razine da ne može dobro predviđati na temelju „neviđenih“ podataka. Možemo vidjeti primjer kada pustimo algoritam „Adam“ da uči do 100 epoha:



Slika 39 Pojava "prenaučenosti"

10.1.5. Algoritam Adamax

Kod Adamax algoritma dobiveni su nešto bolji rezultati. Ovaj je algoritam na tek nakon 50. epohe došao do minimuma funkcije gubitka.

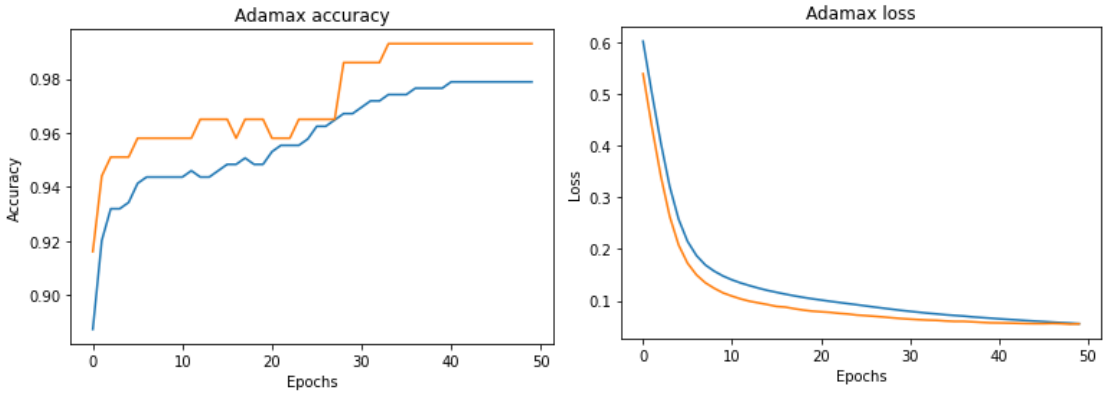
```
start = time.time()
adamax_history = ann_adamax.fit(X_train, y_train, batch_size = 16, epochs = 50, validation_data=(X_test,y_test))
adamax_time = time.time()-start
```

Slika 40 Pokretanje treniranja algoritmom Adamax

„Adamax“ je napravio samo jednu pogrešku i to predvidio je da je rak benigni kada je zapravo bio maligni. Kako je algoritmu bilo potrebno više epoha za treniranje, samo vrijeme sporije je od „Adam“ algoritma za nešto više od sekunde te iznosi 1.817 sekundi.

```
Algoritam: Adamax
Accuracy:  0.993006993006993
ROC AUC: 0.9907407407407407
Confusion matrix
    P0          P1
S0  89        0
S1  1        53
Recall:  0.9814814814814815
Precision:  1.0
F1 score:  0.9906542056074767
Training time (sec): 1.8166496753692627
```

Slika 41 Rezultati testiranja Adamax



Slika 42 Točnost i vrijednost funkcije gubitka po epohama Adamax

Možemo uočiti da validacijski i trening gubitak počinju stagnirati oko 50. epohe. Povećanjem broja epoha treniranja dovodi algoritam do prenaučenosti.

10.1.6. Algoritam Adagrad

Kod teorijskog opisa rada algoritma „Adagrad“, zaključili smo da je ovo najjednostavnija modifikacija osnovnog algoritma gradijentnog pada pa je za očekivati da će ovaj algoritam

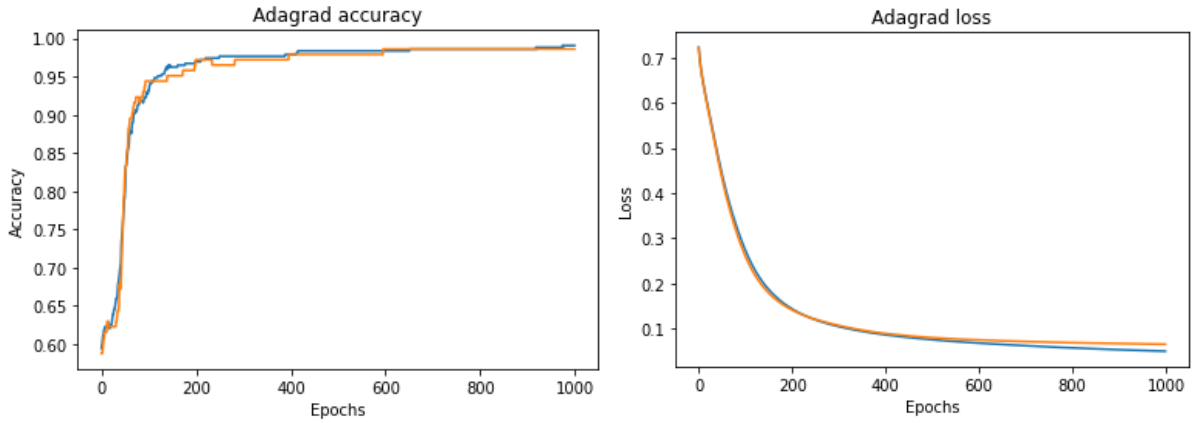
imati najslabije performanse. Broj epoha koji je bio potreban algoritmu „Adagrad“ da dođe (barem blizu) minimuma vrijednosti funkcije gubitka iznosi 1000.

Algoritam je imao sve skupa dva netočna predviđanja, jedan krivi pozitiv i jedan krivi negativ. Vrijeme treniranja bilo je daleko sporije od ostalih algoritama, a iznosi 30 sekundi.

```
Algoritam: Adagrad
Accuracy:  0.986013986013986
ROC AUC: 0.9851227632126508
Confusion matrix
    P0          P1
S0  88        1
S1  1        53
Recall:  0.9814814814814815
Precision:  0.9814814814814815
F1 score:  0.9814814814814815
Training time (sec): 30.081620454788208
```

Slika 43 Rezultati testiranja Adagrad

Vidimo da točnost i vrijednost funkcije gubitka počinju stagnirati zapravo oko 800. epohe. Za razliku od prethodnih algoritama uočeno je da se ne počinje javljati prenaučenost sve većim brojem epoha.



Slika 44 Točnost i vrijednost funkcije gubitka po epohama Adagrad

10.1.7. Algoritam Nadam

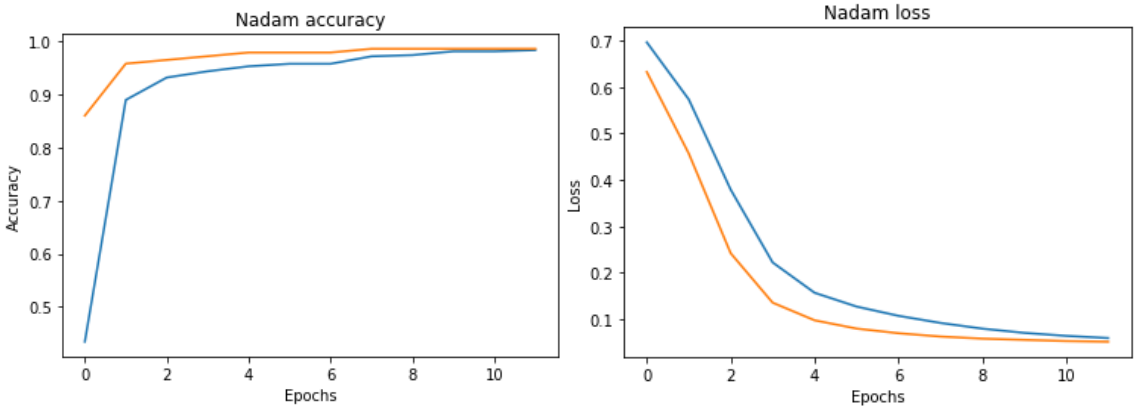
Algoritam NAdam, kao i Adam algoritam ima vrlo brzu konvergenciju prema minimumu pa je bilo potrebno samo 12 epoha treniranja.

Kao i Adagrad imao je po jedan krivi negativ i krivi pozitiv, ali njemu je bilo potrebno samo 12 epoha i 0.893 sekunde treniranja.

```
Algoritam: Nadam
Accuracy:  0.986013986013986
ROC AUC: 0.9851227632126508
Confusion matrix
    P0          P1
S0  88        1
S1  1        53
Recall:  0.9814814814814815
Precision:  0.9814814814814815
F1 score:  0.9814814814814815
Training time (sec): 0.8931665420532227
```

Slika 45 Rezultati testiranja NAdam

Vidimo da do stagnacije funkcije gubika dolazi oko 12. epohe treniranja.



Slika 46 Točnost i vrijednost funkcije gubitka po epohama NAdam

10.1.8. Algoritam SGD

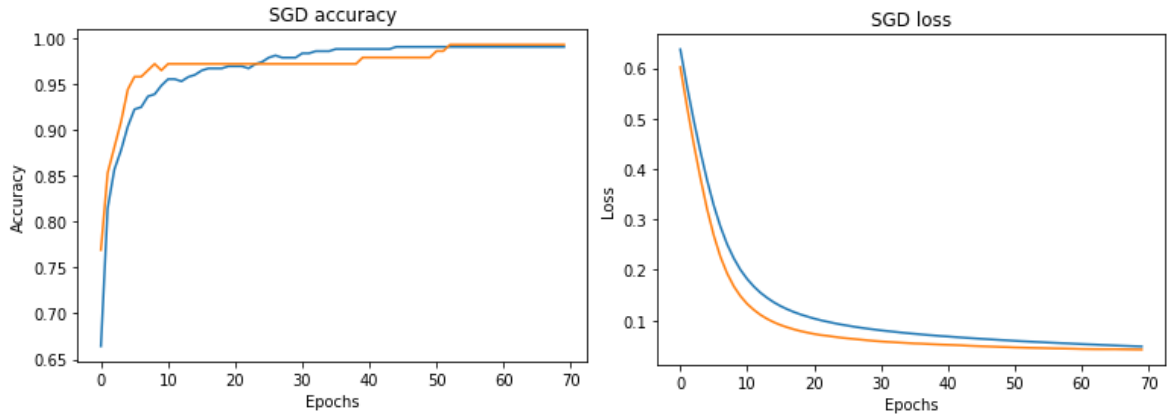
Najosnovnijoj verziji algoritma gradijentnog spusta od opisanih u ovom radu dolazak do minimuma funkcije gubitka bilo je potrebno 70 epoha.

Algoritam je isto tako imao odlične rezultate te je napravio samo jednu pogrešku kao i „Adamax“ i to jedan krivi negativ. Vrijeme treniranja ovog algoritma bilo je 2.2 sekunde.

```
Algoritam: SGD
Accuracy:  0.993006993006993
ROC AUC: 0.9907407407407407
Confusion matrix
    P0          P1
S0  89        0
S1  1        53
Recall:  0.9814814814814815
Precision:  1.0
F1 score:  0.9906542056074767
Training time (sec): 2.2042672634124756
```

Slika 47 Rezultati testiranja SGD

Iz grafova možemo iščitati da algoritmu treba oko 70 epoha za stagnaciju vrijednosti funkcije gubitka. Također, uočeno je da povećanjem broja epoha ne javlja se toliko značajna „prenaučenost“ kao kod ostalih algoritama.



Slika 48 Točnost i vrijednost funkcije gubitka po epohama SGD

10.1.9. Algoritam RMSProp

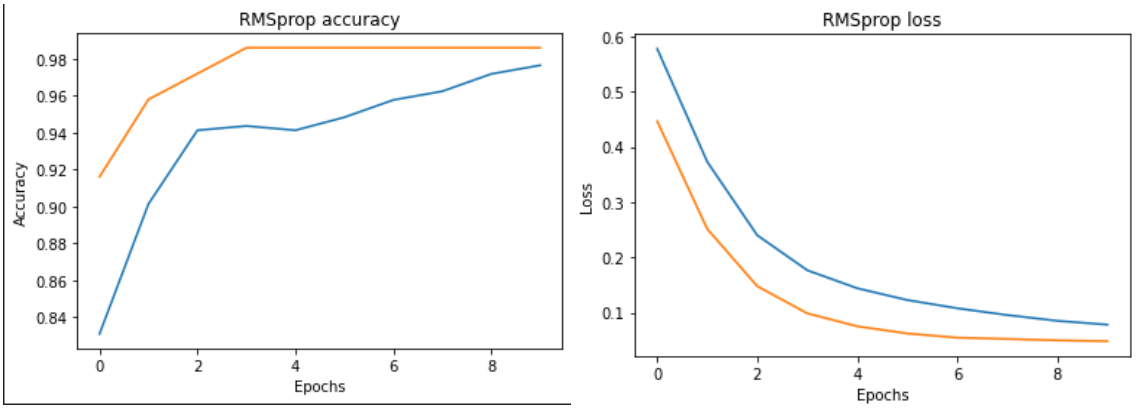
Posljednji analizirani algoritam jest „RMSProp“. Ovo je također jedan od bržih algoritama, a broj epoha nakon kojih se dolazi do minimuma gubitka jest 10.

Ovaj model napravio je po jednu krivu krivo pozitivnu i krivu negativnu klasifikaciju. Vrijeme treniranja bilo je 0.7 sekunde.

```
Algoritam: RMSprop
Accuracy:  0.986013986013986
ROC AUC: 0.9851227632126508
Confusion matrix
    P0          P1
S0  88        1
S1  1        53
Recall:  0.9814814814814815
Precision:  0.9814814814814815
F1 score:  0.9814814814814815
Training time (sec): 0.7086441516876221
------------------------------------------------
```

Slika 49 Rezultati testiranja RMSProp

Stagnacija gubitka počinje se javljati oko 10. epohe.



Slika 50 Točnost i vrijednost funkcije gubitka po epohama RMSProp

10.1.10. Zaključak na performanse algoritama

Za lakšu komparaciju prikazat će se sve metrike u jednoj tablici:

	Algoritam	Accuracy	ROC AUC	Recall	Precision	F1 score	Training time (sec)	Test time (sec)	Epochs
0	Adam	0.979021	0.975864	0.962963	0.981132	0.971963	0.652592	0.027025	10
1	Adamax	0.993007	0.990741	0.981481	1.000000	0.990654	1.816650	0.023021	50
2	Adagrad	0.986014	0.985123	0.981481	0.981481	0.981481	30.081620	0.022020	1000
3	Nadam	0.986014	0.985123	0.981481	0.981481	0.981481	0.893167	0.023021	12
4	SGD	0.993007	0.990741	0.981481	1.000000	0.990654	2.204267	0.023021	70
5	RMSprop	0.986014	0.985123	0.981481	0.981481	0.981481	0.707772	0.023021	10

Slika 51 Ukupne performanse svih algoritama

Što se tiče vremena treniranja algoritmi Adam, Adamax i RMSProp pokazali su se najboljima. Ovim algoritmima bilo je potrebno oko 10 epoha da dođu do minimuma funkcije koštanja, a treniranje je time završeno unutar jedne sekunde. Vremena testiranja validacijskog skupa vrlo su slična svim algoritmima te iznose oko 0.02 sekunde. Točnosti svih algoritama također vrlo su slične. Kako imamo mali validacijski skup podataka, izgleda da postoji velika razlika u ukupnim točnostima, no „najslabiji“ algoritam „Adam“ imao je samo dvije krive predikcije više u odnosu na „najbolje“ algoritame „Adamax“ i „SGD“. Isto tako sama treniranja provedena su nekoliko puta te su „Adam“, „RMSProp“, „NAdam“ i „Adamax“ imali različite točnosti, od 97% do 99%. Razlog bi mogao biti u tome što kod ovakvog problema gdje funkcija gubitka vrlo brzo pada, velika varijabilnost vrijednosti stope učenja kod ovih algoritama može naškoditi padu prema minimumu vrijednosti funkcije gubitka. Algoritmi „Adagrad“ i „SGD“ imali su dosta dulja vremena treniranja, odnosno broj epoha koje su im trebale da dostignu minimum. Isto tako provođenje treniranja nekoliko puta nije davalo drugačije rezultate već su ovi algoritmi bili konzistentniji. Čini se da polako spuštanje prema minimumu ima prednost u točnosti kod ovakvog, jednostavnog problema, no to dolazi s problemom duljeg treniranja.

10.2. Credit card fraud detection dataset

10.2.1. Opis domene

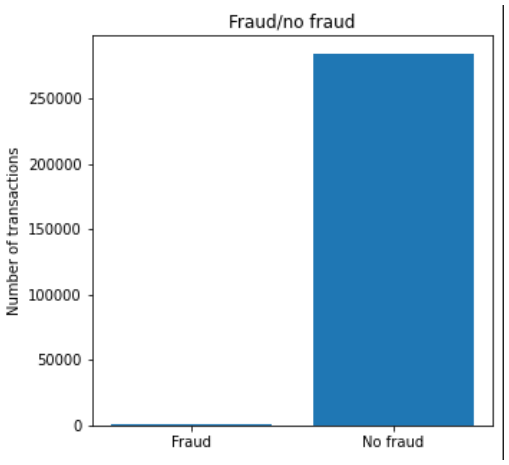
Drugi skup podataka koji će se analizirati odnosi se na predviđanje lažnih transakcija kreditnim karticama. Podaci koji se nalaze u skupu odnose se na transakcije kreditnim karticama u vlasništvu Europljana koje su se odvile kroz dva dana u rujnu 2013. godine. Pomoću skupa podataka pokušat će se predvidjeti postoji li prijevara u transakciji ili ne. [6]

10.2.2. Analiza i čišćenje skupa podataka

Kao i kod prethodno skupa podataka inicijalna ekstrakcija meta-značajki odrađena je pomoću „*pymfe“* alata. Broj atributa skupa jest 31, dok je broj instanci 284 807. Skup se sastoji od 30 nezavisnih varijabli, dok je zavisna varijabla binarna kategorijska varijabla. Od navedenih 30 atributa jedan („*Time*“) se odnosi na broj sekundi koji je protekao od prve transakcije. Također postoji atribut „*Amount“* koji govori kolika je vrijednost transakcije. Ostalih 28 atributa su 28 PCA komponenti dobivenih PCA analizom. PCA analiza izvršena je prije preuzimanja ovog skupa pa je samo značenje atributa originalnog skupa podataka nepoznato. Iako je kod ovog skupa PCA analiza izvršena u svrhu sakrivanja povjerljivih podataka, primarna svrha ovog postupka jest smanjivanje dimenzionalnosti skupa podataka na način da se visokodimenzionalni skup podataka transformira u skup podataka nižih dimenzija s tim da transformirani skup podataka sadrži čim veću količinu prediktivnih informacija originalnog skupa podataka.(Cheng, 2022)

Vidjeli smo da je skup podataka vrlo velik (284 807 instanci), pa je potrebno analizirati kako izgleda raspodjela instanci prema klasama.

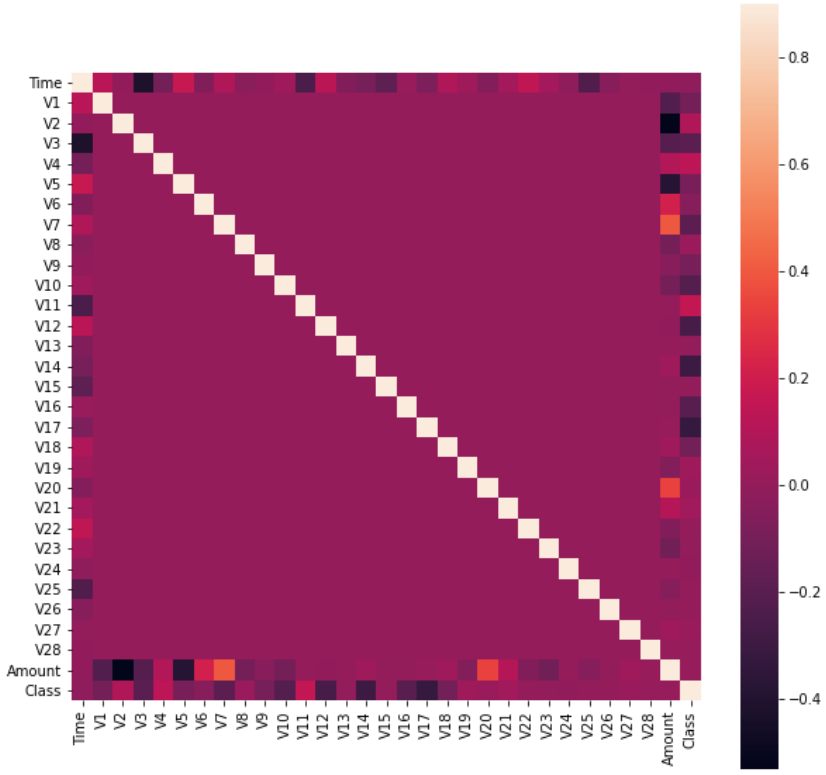
[6] Poveznica na skup podataka: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud



Slika 52 Odnos broja instanci prema klasama

Prema grafu broja instanci koje pripadaju određenoj klasi vidimo jako veliku klasnu nebalansiranost. Klasi „Fraud“ pripada samo 0.17% ukupnog broja instanci.

Možemo analizirati i korelacijsku matricu:



Slika 53 Korelacijska matrica

Ono što možemo odmah uočiti jest nikakva korelacija između PCA komponenti. To jest i očekivano pošto izrađene PCA komponente moraju biti međusobno nezavisne. Možemo detaljnije pogledati korelacije između nezavisnih varijabli i varijable „Class“:

	Class		
Time	-0.012323	V15	-0.004223
V1	-0.101347	V16	-0.196539
V2	0.091289	V17	-0.326481
V3	-0.192961	V18	-0.111485
V4	0.133447	V19	0.034783
V5	-0.094974	V20	0.020090
V6	-0.043643	V21	0.040413
V7	-0.187257	V22	0.000805
V8	0.019875	V23	-0.002685
V9	-0.097733	V24	-0.007221
V10	-0.216883	V25	0.003308
V11	0.154876	V26	0.004455
V12	-0.260593	V27	0.017580
V13	-0.004570	V28	0.009536
V14	-0.302544	Amount	0.005632

Slika 54 Korelacije nezavisnih varijabli s zavisnom

Vidimo da nijedna varijabla nema preveliku korelaciju s zavisnom varijablom, no vidimo da postoji nekoliko varijabli s djelomičnom negativnom korelacijom s varijablom „Class“.

Skup podataka također ne sadrži podatke koji nedostaju ni za jednu varijablu.

```
#nema null vrijednosti
dataset.isna().sum()
```

```
Output exceeds the size limit. Open the full output data in a text editor
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
...
V27       0
V28       0
Amount    0
Class     0
```

Slika 55 Broj vrijednost koje nedostaju po varijablama

Ovom analizom možemo zaključiti da je skup podataka visokodimenzionalan i vrlo velik prema broju instanci. Nad skupom je podataka prije preuzimanja provedena PCA analiza što

za posljedicu znači kvalitetan skup podataka bez nedostajućih vrijednosti i outlier-a. Što se tiče korelacija između nezavisnih i zavisne varijable uočeno je da postoji veliki broj podataka sa djelomičnom negativnom korelacijom. Vidjeli smo također da je skup izrazito nebalansiran zbog čega će tijekom analize rezultata veći naglasak biti na ROC AUC i F1 točnosti.

Što se tiče dodatne pripreme potrebno je podijeliti skup na nezavisne i zavisnu varijablu. Iz skupa prediktora izbacit će se varijabla *„Time“* pošto nam ona govori samo u kojem trenutku je transakcija generirana u odnosu na prvu transakciju u danu, a i vidjeli smo da je korelacija s zavisnom varijablom oko nule.

```
X=dataset.drop(['Time','Class'], axis=1)
Y=dataset['Class']
```

Slika 56 Podjela skupa na nezavisne i zavisnu varijablu

U validacijski skup podataka stavljeno je 30% originalnog skupa podataka je nad podijeljenim skupom izvršena standardizacija.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state=42)
```

Slika 57 Podjela skupa na testni i trening

10.2.3. Postavke i treniranje

Kao i kod prošlog skupa podataka struktura neuronske mreže ista je za sve algoritme. Isprobano je nekoliko različitih struktura te je nabolje rezultate od ispitanih i najstabilnije treniranje dala struktura sa tri skrivena sloja od po 8, 16 i 32 neurona. Aktivacijske funkcije u skrivenim slojevima su ponovno „ReLU“, a na izlaznom „Sigmoida“. Funkcija gubitka je binarna unakrsna entropija, a metrika koja se računa tijekom treninga jest ROC AUC.

```
metric = tf.keras.metrics.AUC()

ann_adam = tf.keras.models.Sequential()
#3 skirvena sloja sa po 8, 16 i 32 neurona
ann_adam.add(tf.keras.layers.Dense(units=8, activation='relu'))
ann_adam.add(tf.keras.layers.Dense(units=16, activation='relu'))
ann_adam.add(tf.keras.layers.Dense(units=32, activation='relu'))
# Dodavanje izlaznog sloja
ann_adam.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
# kompajliranje mreže
ann_adam.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = [metric])
```

Slika 58 Struktura neuronske mreže

Veličina uzorka za jednu iteraciju povratne propagacije jest u ovom slučaju (zbog puno većeg skupa podataka) veći i iznosi 32 instance.

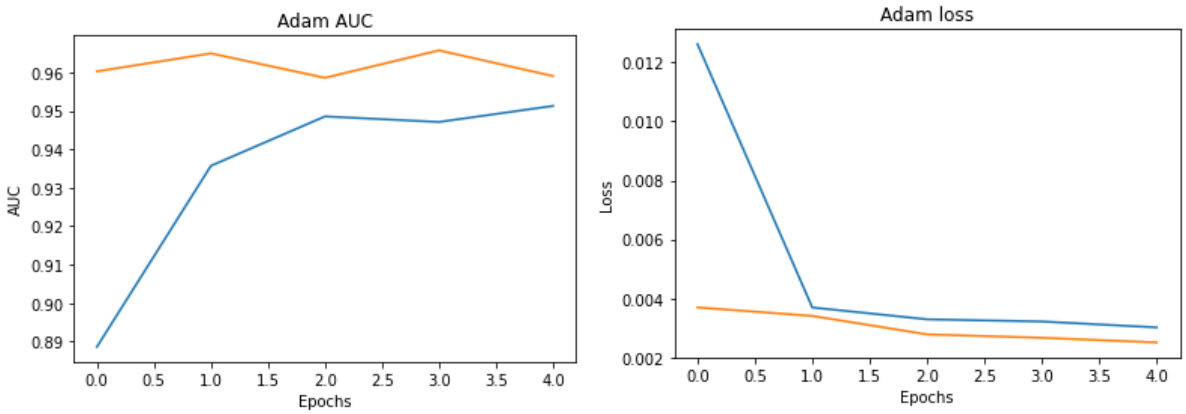
10.2.4. Algoritam Adam

Prvi algoritam koji je ispitan je „*Adam*“. *Adam* je nakon 5 epoha došao do minimuma funkcije gubitka te je nakon 22 sekunde završeno treniranje.

```
Algoritam: Adam
Accuracy:  0.9995084442259752
ROC AUC: 0.9116592045753089
Confusion matrix
    P0          P1
S0  85289       18
S1  24        112
Recall:  0.8235294117647058
Precision:  0.8615384615384616
F1 score:  0.8421052631578948
Training time (sec): 21.596610069274902
```

Slika 59 Rezultati testiranja Adam

Zanimljivo jest što već nakon prve epohe treniranja funkcija gubitka jest vrlo blizu minimuma.



Slika 60 Točnost i vrijednost funkcije gubitka po epohama Adam

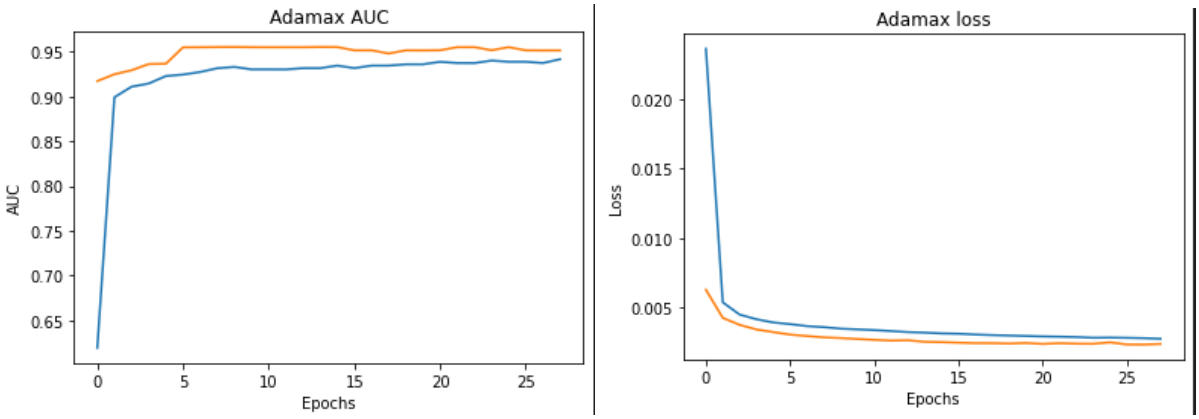
10.2.5. Algoritma Adamax

Drugi algoritam jest Adamax. Kao i kod prethodnog skupa podataka Adamax je sporiji od „Adam“ algoritma, te je broj epoha nakon kojeg se vrijednost funkcije gubitka ustabilila na minimumu iznosio 28. Naravno, time je i ukupno vrijeme treniranja nešto dulje, a iznosi 116 sekundi.

```
Algoritam: Adamax
Accuracy:  0.9994850368081645
ROC AUC: 0.9153180916123703
Confusion matrix
    P0          P1
S0  85286       21
S1  23        113
Recall:  0.8308823529411765
Precision:  0.8432835820895522
F1 score:  0.8370370370370371
Training time (sec): 115.82010459899902
```

Slika 61 Rezultati testiranja Adamax

Vidimo da se funkcija gubitka ustabilila čak i nešto prije 30 epohe.



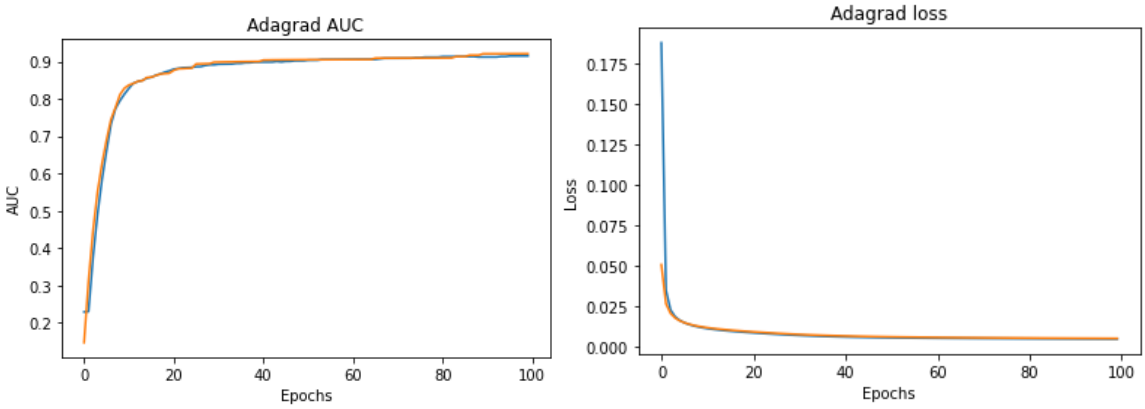
Slika 62 Točnost i vrijednost funkcije gubitka po epohama Adamax

10.2.6. Algoritam Adagrad

Kod ovakvog većeg i kompleksnijeg skupa podataka možemo uočiti nedostatke algoritma „Adagrad“. Uz najsporije vrijeme treniranja, algoritam je imao i najslabije točnosti. Trening je trajao 100 epoha, no oko 60. epohe vrijednost funkcije gubitka postala je minimalna te više nije padala. U odnosu na ostale algoritme algoritam ima znatno niže točnosti, osobito vrijednost odaziva gdje je napravio čak 42 neispravne negativne klasifikacije.

```
Algoritam: Adagrad
Accuracy:  0.9993211848834895
ROC AUC: 0.845494456354523
Confusion matrix
    P0          P1
S0  85291       16
S1  42        94
Recall:  0.6911764705882353
Precision:  0.8545454545454545
F1 score:  0.7642276422764227
Training time (sec): 426.9864070415497
```

Slika 63 Rezultati testiranja Adagrad



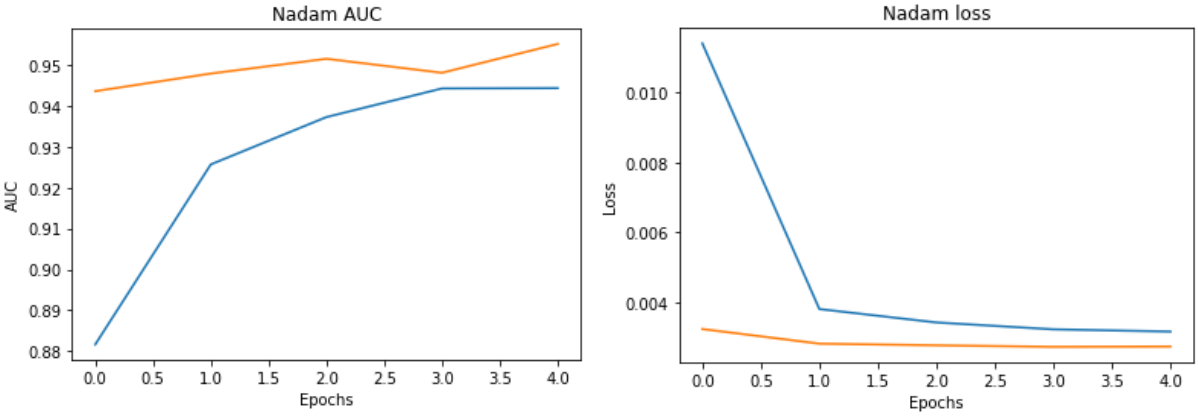
Slika 64 Točnost i vrijednost funkcije gubitka po epohama Adagrad

10.2.7. Algoritam NAdam

I u ovom slučaju NAdam, zajedno s algoritmom Adam, imao je najbrže vrijeme treniranje, dolazak do minimuma funkcije gubitka dogodio se nakon 7 epoha, ali i u ovom slučaju nakon prve epohe već se približio minimumu. Za razliku od „Adam“ algoritma napravio je nešto više pogrešaka zbog čega je prema svim metrikama algoritam nešto slabiji.

```
Algoritam: Nadam
Accuracy:  0.9994148145547324
ROC AUC: 0.9152829245100224
Confusion matrix
    P0          P1
S0  85280       27
S1  23        113
Recall:  0.8308823529411765
Precision:  0.8071428571428572
F1 score:  0.818840579710145
Training time (sec): 23.42632484436035
```

Slika 65 Rezultati testiranja NAdam



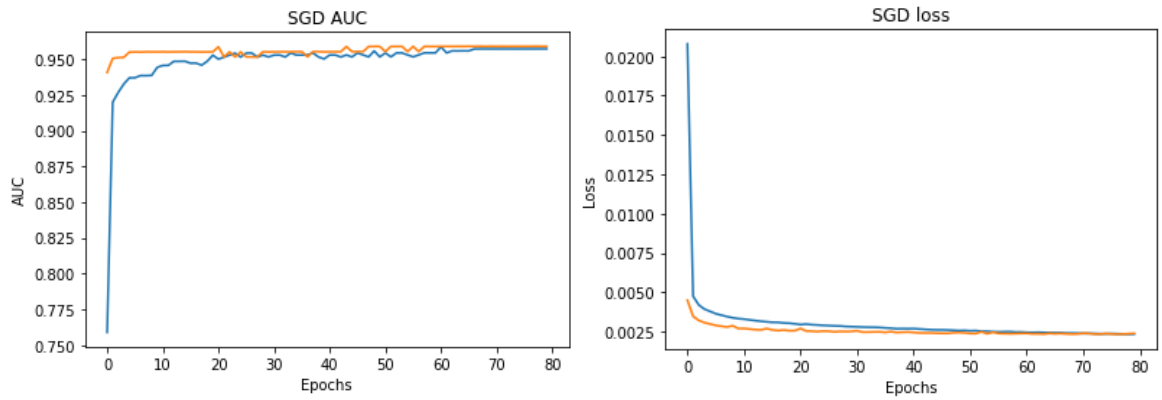
Slika 66 Točnost i vrijednost funkcije gubitka po epohama NAdam

10.2.8. Algoritam SGD

Osnovnom „Mini-Batch Gradient Descent“ algoritmu ponovno je bilo potrebno nešto dulje vremena za treniranje, no algoritam je ponovno pokazao vrlo dobre rezultate. Treniranje je trajalo 80 epoha gdje se vrijednost funkcije gubitka ustabilila.

```
Algoritam: SGD
Accuracy:  0.9995552590615966
ROC AUC: 0.9116826493102076
Confusion matrix
    P0          P1
S0  85293       14
S1  24        112
Recall:  0.8235294117647058
Precision:  0.8888888888888888
F1 score:  0.8549618320610687
Training time (sec): 323.0984671115875
```

Slika 67 Rezultati testiranja SGD



Slika 68 Točnost i vrijednost funkcije gubitka po epohama SGD

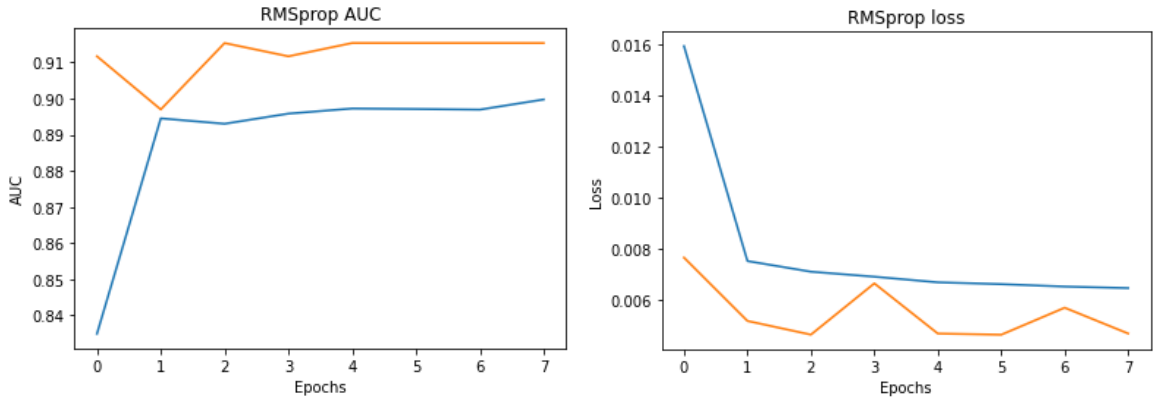
10.2.9. Algoritam RMSProp

RMSProp također je imao dosta kraće vrijeme treniranja te je funkcija gubitka vrlo brzo došla do minimuma. Trening je trajao 8 epoha, odnosno 35 sekundi

```
Algoritam: RMSprop
Accuracy:  0.9993797034280163
ROC AUC: 0.9042535127453164
Confusion matrix
    P0          P1
S0  85280       27
S1  26        110
Recall:  0.8088235294117647
Precision:  0.8029197080291971
F1 score:  0.805860805860806
Training time (sec): 35.02104353904724
```

Slika 69 Rezultati testiranja RMSProp

U odnosu na ostale algoritme funkcija gubitka imala je velike oscilacije, pogotovo treniranjem kroz više epoha, no točnosti su vrlo slične drugim algoritmima.



Slika 70 Točnost i vrijednost funkcije gubitka po epohama RMSProp

10.2.10. Zaključak na performanse algoritama

Popis vrijednosti metrika svih algoritama možemo vidjeti u sljedećoj tablici:

	Algoritam	Accuracy	ROC AUC	Recall	Precision	F1 score	Training time (sec)	Test time (sec)	Epochs
0	Adam	0.999508	0.911659	0.823529	0.861538	0.842105	21.596610	0.637579	5
1	Adamax	0.999485	0.915318	0.830882	0.843284	0.837037	115.820105	0.605538	28
2	Adagrad	0.999321	0.845494	0.691176	0.854545	0.764228	426.986407	0.603548	100
3	Nadam	0.999415	0.915283	0.830882	0.807143	0.818841	23.426325	0.609556	5
4	SGD	0.999555	0.911683	0.823529	0.888889	0.854962	323.098467	0.630573	80
5	RMSprop	0.999380	0.904254	0.808824	0.802920	0.805861	35.021044	0.632574	8

Slika 71 Ukupne performanse svih algoritama

Očekivano najbrži algoritmi ponovno su bili *„Adam“* i *„NAdam“* iako prema ukupnim performansama nisu bili najbolji, no *„Adam“* imao vrlo slične performanse kao i „najbolji“ SGD. Gledajući ukupne performanse prema ROC AUC i F1 mjeri malo bolji od brzog *„Adam“*

algortima pokazao se SGD, no imao je i drugo najdulje vrijeme treniranja. Ovaj algoritam također je imao vrlo stabilne vrijednosti funkcije gubitka po epohama, što je i očekivano pošto ne postoji varijabilnost u stopi učenja. Najslabiji algoritam prema ROC AUC i F1 mjeri pokazao se „Adagrad“ koji je, čini se, prerano prestao s dodatnim učenjem zbog preniske stope učenja na višim iteracijama. Gotovo podjednak broj krivi pozitivnih i negativnih predikcija imali su „*Adamax*“ i „*RMSProp*“ pa su vrijednosti odaziva i preciznosti slične, no RMSprop imao je nešto više krivih predikcija pa je ukupno nešto slabiji. Zbog vrlo kratkog vremena treniranja, ali i vrlo visoke točnosti, preporuka bi, kod ovakvih velikih skupova podataka, bila korištenje algoritma „*Adam“* i „*NAdam“*.

10.3. Rain tommorrow prediction

10.3.1. Opis domene

Ovaj skup podataka odnosi se na predviđanje pojave kiše sljedeći dan. Podaci skupa prikupljeni su kroz svakodnevno praćenje vremena u desetogodišnjem vremenskom periodu na različitim meteorološkim stanicama u Australiji. [7]

10.3.2. Analiza i čišćenje skupa podataka

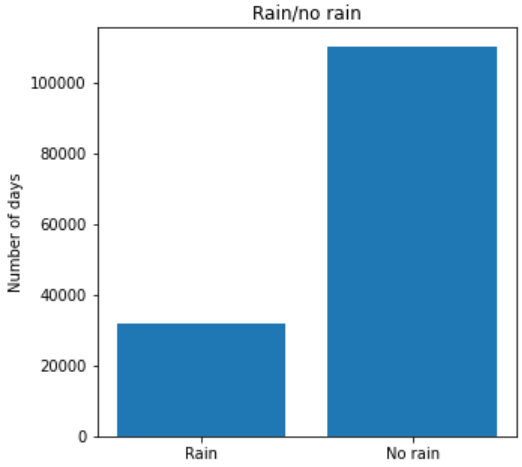
Ekstrakcijom meta-značajki pomoću *„pymfe“* alata dobivaju se osnovne informacije skupa podataka. Skup se sastoji od 23 atributa koji su kategorijskog i numeričkog tipa, dok zavisna varijabla ima dvije moguće klase koje se predviđaju, biti će kiše ili neće biti kiše. Ukupni broj instanci jest 140 787.

Što se tiče nezavisnih varijabli postoje različiti prikupljeni podaci. Atribut „*Date*“ sadrži informaciju o tome na koji je dan u godini prikupljena instanca, a „*Location*“ na kojoj meteorološkoj stanici. Također imamo informacije o temperaturi na određeni dan, a za to se odnose atributi „*MinTemp“,* koji govori koja je minimalna izmjerena temperatura u danu, „*MaxTemp*“ koja je najviša izmjerena temperatura u danu, *„Temp9am“* temperatura u 9 sati, *„Temp3pm“* u 15 sati. Atribut „*Rainfall*“ daje informaciju o tome koliko je kiše palo na promatrani dan, a *„Evaporation“* koja je količina isparavanja u „mm“ u proteklom dvadeset četverosatnom razdoblju. Isto tako postoji i informacija o tome koja je količina ukupnog broja sati sijanja sunca u danu, a to nam govori varijabla „*Sunshine*“. Također imamo nekoliko različitih atributa koji daju informacije o vjetru, a to su *„WindGustDir“, „WindGustSpeed“,*

[7] Poveznica na skup podataka: https://www.kaggle.com/jsphyg/weather-dataset-rattle-package

„WindDir9am“, „WindDir3pm“, „WindSpeed9am“ i „*WindSpeed3pm“.* Varijable „*Humidity9am*“ i „*Humidity9am*“ govore kolika je bila vlaga u 9 sati, odnosno, 15 sati u promatranom danu. Atributi „*Pressure9am*“ i „*Pressure3pm*“ daju informacije o tlaku zraka u 9 i 15 sati, a *„Cloud9am“* i „*Cloud3pm*“ o pokrivenosti neba oblacima u 9 i 15 sati. Na kraju imamo binarnu kategorijsku varijablu „*RainToday*“ koja govori je li padala kiša na promatrani dan, a binarna kategorijska varijabla koju želimo predvidjeti jest „*RainTommorow*“ kojom želimo predvidjeti hoće li padati kiša na sljedeći dan.

Ponovno će se analizirati koja je raspodjela instanci po klasama zavisne varijable:



Slika 72 Broj instanci prema klasama

Vidimo da postoji manja klasna nebalansiranost, no nije velika kao kod prethodnog skupa podataka. Odnos klasa jest 22%/78%, no i u ovom slučaju bolju sliku kod analize rezultata dati će ROC AUC i F1 mjera.

Što se tiče čišćenja skupa provest će se nekoliko koraka. Prvi jest ekstrakcija važnih informacija iz varijable datum. Iako se često kod predikcije ova varijabla zanemaruje, kod skupa podataka gdje želimo predvidjeti meteorološke elemente datum prikupljanja podataka mogao bi sadržavati prediktivne sposobnosti. Određeni mjeseci u godini mogli bi imati veću šansu pojave kiše pa bi ova varijabla mogla omogućiti bolju točnost. Zbog toga su kreirane tri nove varijable *„Day“, „Month“ i „Year“* koje se izvlače iz varijable *„Date“*

Prethodne skupove podataka mogli bi okarakterizirati kao vrlo „čiste“. Glavna karakteristika ovog skupa podataka su vrlo veliki broj vrijednosti koje nedostaju. Gotovo sve varijabli imaju vrijednosti koje nedostaju. Čišćenje od nedostajućih vrijednosti provedeno je na razne načine ovisno o tipu varijabli.

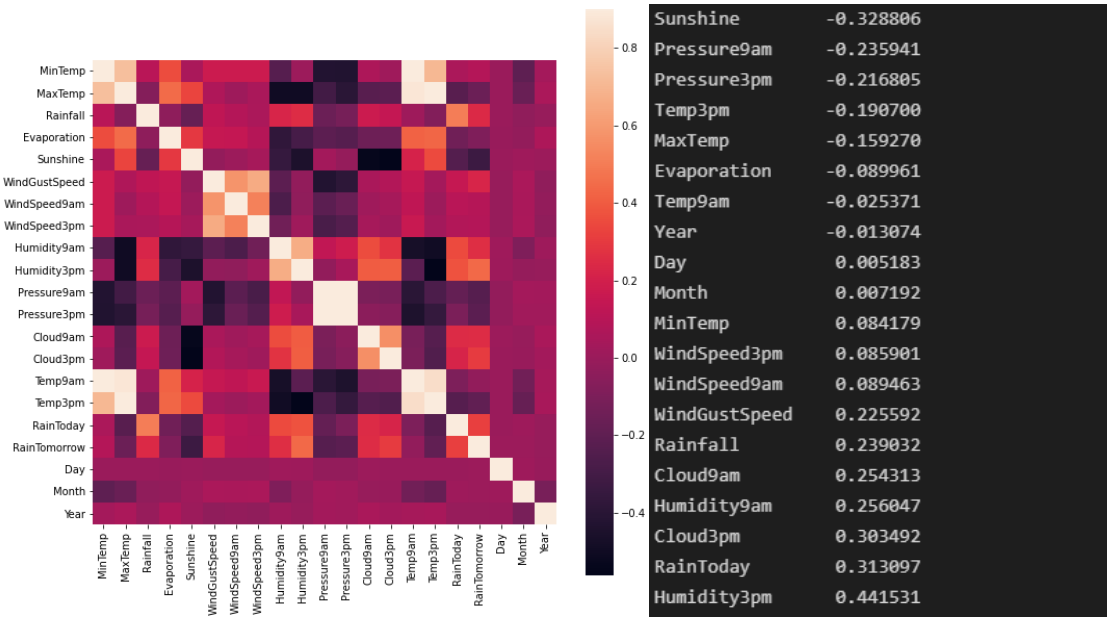
```
dataset.isna().sum()
```

```
Location           0
MinTemp         1485
MaxTemp         1261
Rainfall        3261
Evaporation    62790
Sunshine       69835
WindGustDir    10326
WindGustSpeed  10263
WindDir9am     10566
WindDir3pm      4228
WindSpeed9am    1767
WindSpeed3pm    3062
Humidity9am     2654
Humidity3pm     4507
Pressure9am    15065
Pressure3pm    15028
Cloud9am       55888
Cloud3pm       59358
Temp9am         1767
Temp3pm         3609
RainToday       3261
RainTomorrow    3267
Day                0
Month              0
Year               0
```

Slika 73 Broj vrijednosti koje nedostaju po atributima

Sve instance s nedostajućim vrijednosti za varijable „*RainToday*“ i „*RainTomorrow*“ obrisane su pošto ih nije bilo previše, za kategorijske varijable nedostajuće vrijednosti zamijenjene su s vrijednosti „*Unknown*“, dok je za numeričke varijable uzeta prosječna vrijednost svakog od atributa. Iako za atribute poput „*Sunshine*“ i „*Evaporation*“ gotovo 50% vrijednosti nedostaje, ipak se oni neće brisati jer se pokušava istražiti koliko dobro algoritmi treniranja neuronskih mreža rade nad skupovima podataka sa šumom. Također postoje varijable s „outlier“ vrijednostima koje ćemo također ostaviti u skupu podataka upravo iz spomenutog razloga.

Možemo analizirati korelaciju nezavisnih varijabli s zavisnom.




```
Sunshine       -0.328806
Pressure9am    -0.235941
Pressure3pm    -0.216805
Temp3pm        -0.190700
MaxTemp        -0.159270
Evaporation    -0.089961
Temp9am        -0.025371
Year           -0.013074
Day             0.005183
Month           0.007192
MinTemp         0.084179
WindSpeed3pm    0.085901
WindSpeed9am    0.089463
WindGustSpeed   0.225592
Rainfall        0.239032
Cloud9am        0.254313
Humidity9am     0.256047
Cloud3pm        0.303492
RainToday       0.313097
Humidity3pm     0.441531
```

Slika 74 Korelacijska matrica i korelacije nezavisnih varijabli s zavisnom

Vidimo da najveću pozitivnu korelaciju sa pojavom kiše sljedeći dan ima vlaga u 15 sati, padanje kiše na promatrani dan i pokrivenost oblacima u 15 sati na promatrani dan. Najveću negativnu korelaciju imaju broj sati u danu sa suncem te tlak zraka u 9 i 15 sati.

Posljednji koraci pripreme podataka ponovno su uključivali podjelu skupa na nezavisne varijable i zavisnu varijablu kao i standardizacija skupa podataka. U testni skup podataka postavljeno je 25% instanci iz originalnog skupa, a ostatak instanci otišao je u trening skup.

Analizom skupa podataka uočeno je da je skup podataka poveći prema broju instanci, no glavna karakteristika bila bi šum u podacima. Za određene atribute gotovo polovica skupa imala je vrijednosti koje nedostaju, ali uočene su i outlier vrijednosti. Također, postoji podjednaki broj varijabli s pozitivnom i negativnom korelacijom, no nijedna nezavisna varijabla nema znatnu korelaciju sa pojavom kiše sljedeći dan. Kako postoji klasna nebalansiranost skupa, kod analize rezultata bolju sliku dati će ROC AUC i F1 mjera.

10.3.3. Postavke i treniranje

Struktura neuronske mreže koja se pokazala najoptimalnijom sastoji se od dva skrivena sloja od po 16 i 32 neurona. Ovakva struktura ponovno će biti ista kod svih algoritama. Aktivacijske funkcije ponovno su „ReLU“ u skrivenim slojevima te „Sigmoida“ na izlaznom sloju. Funkcija gubitka jest binarna unakrsna entropija.

```
ann_adam = tf.keras.models.Sequential()
#2 skirvena sloja sa po 16 i 32 neurona
ann_adam.add(tf.keras.layers.Dense(units=16, activation='relu'))
ann_adam.add(tf.keras.layers.Dense(units=32, activation='relu'))
# Dodavanje izlaznog sloja
ann_adam.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
# kompajliranje mreže
ann_adam.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Slika 75 Struktura neuronske mreže

Zbog bolje stabilnosti vrijednosti funkcije gubitka kroz epohe treniranja odabrana je manja veličina uzorka propagacije (*„batch_size“*) te iznosi 16 instanci.

10.3.4. Algoritam Adam

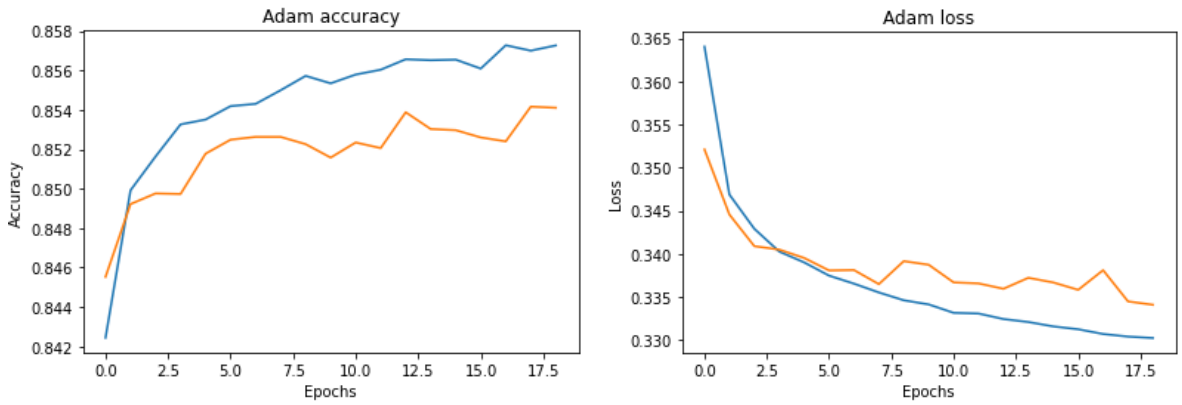
Do sad se pokazalo da je algoritam „*Adam*“ jedan od najbržih algoritama, a to možemo vidjeti i kod ovog skupa. Algoritmu je trebalo 19 epoha do završetka treniranja. Nad ovim skupom možemo vidjeti puno slabije performanse. Sama točnost iznosi 0.8541 uz ROC AUC od 0.734. F1 mjera iznosi dosta niskih 61.21%. Razlog tome je vrlo slabi odaziv od samo

0.5188 iz razloga što je nad validacijskim skupom napravljeno veliki broj grešaka u obliku neispravnih negativnih klasifikacija, čak 3758 u validacijskom skupu od 35 197 instanci. Ipak, kako je odaziv veći od 0.5, algoritam je napravio više točnih klasifikacija, a to je 4052. Preciznost je nešto veća i iznosi 0.746 uz 1377 neispravnih neispravnih pozitivnih klasifikacija.

```
Algoritam: Adam
Accuracy:  0.8541068841094411
ROC AUC: 0.7342713467192183
Confusion matrix
    P0          P1
S0  26010       1377
S1  3758        4052
Recall:  0.5188220230473751
Precision:  0.7463621293055811
F1 score:  0.6721308255910567
Training time (sec): 67.95605945587158
```

Slika 76 Rezultati testiranja Adam

Vidimo da je funkcija gubitka padala do 19. epohe nakon čega je u ispitivanju validacijska funkcija gubitka počela rasti.



Slika 77 Točnost i vrijednost funkcije gubitka po epohama Adam

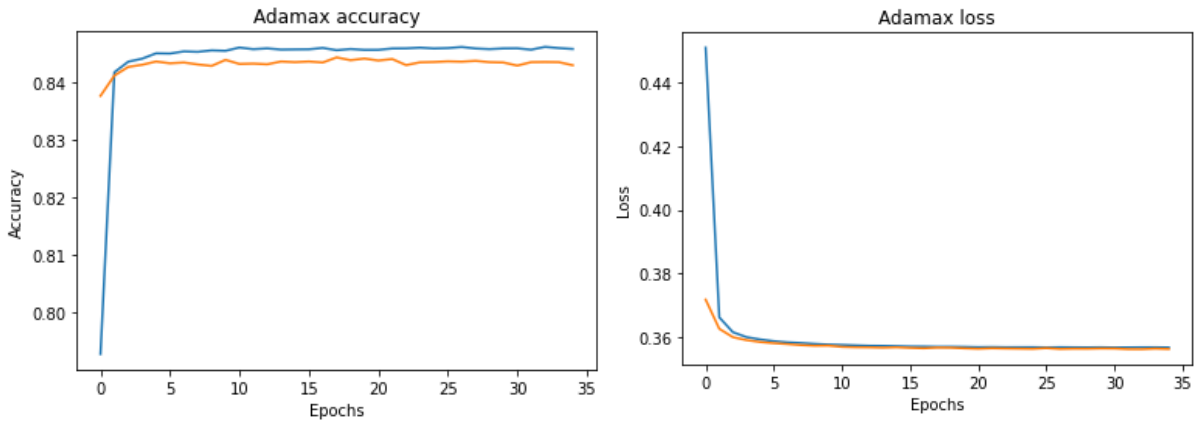
10.3.5. Algoritam Adamax

Performanse Adamax algoritama nešto su slabije od Adam-a. Model s Adamax algoritmom treniran je nešto dulje te je nakon 35 epoha stigao do ROC AUC vrijednosti od 0.7129. Sve ostale metrike pokazale su da je ovaj model nešto slabiji od onog kreiranog Adam algoritmom. Ovaj je model nad testnim skupom zapravo napravio više neispravnih negativnih klasifikacija od ispravnih pozitivnih.

```
Algoritam: Adamax
Accuracy:  0.8428843367332444
ROC AUC: 0.7129189283755525
Confusion matrix
    P0          P1
S0  25924        1463
S1  4067        3743
Recall:  0.4792573623559539
Precision:  0.718978102189781
F1 score:  0.5751382913337431
Training time (sec): 108.92149424552917
```

Slika 78 Rezultati testiranja Adamax

Uočeno je i da je minimum funkcije gubitka postignut oko 20.epohe treniranja.



Slika 79 Točnost i vrijednost funkcije gubitka po epohama Adamax

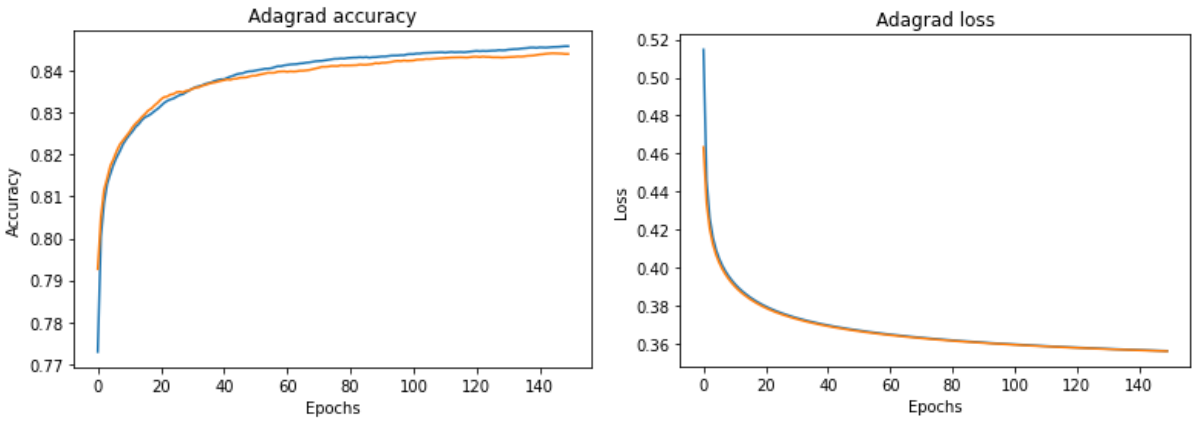
10.3.6. Algoritam Adagrad

Nešto slabije performanse od *Adam*-a i *NAdam*-a ponovno je imao algoritam *Adagrad*, no malo bolje od *Adamax*-a. Model izrađen *Adagrad* algoritmom treniran je 150 epoha, ROC AUC iznosio je 0.7196, a F1 mjera 0.5853. Odaziv je i ovdje iznosio manje od 0.5.

```
Algoritam: Adagrad
Accuracy:  0.844077620251726
ROC AUC: 0.7196349899554669
Confusion matrix
    P0          P1
S0  25836        1551
S1  3937        3873
Recall:  0.4959026888604353
Precision:  0.7140486725663717
F1 score:  0.5853105636995618
Training time (sec): 505.92572140693665
```

Slika 80 Rezultati testiranja Adagrad

Vrijednost funkcije gubitka i u ovom slučaju je vrlo sporo padala na višim epohama, no ponovno je bila vrlo stabilna i nije oscilirala.



Slika 81 Točnost i vrijednost funkcije gubitka po epohama Adagrad

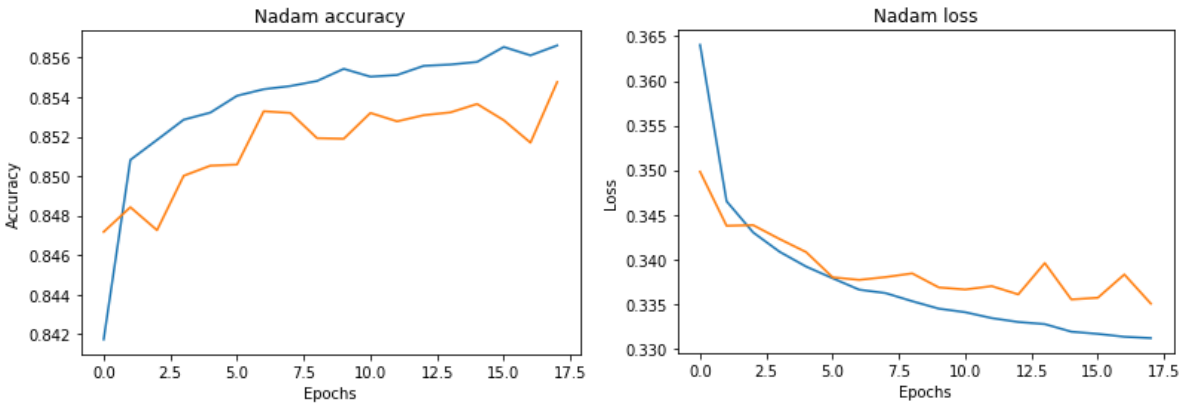
10.3.7. Algoritam NAdam

NAdam ponovno je bio najbrži algoritam te mu je bilo potrebno 18 epoha da dođe do najboljih rezultata. Prema ukupnim performansama bio je i nešto bolji od Adam algoritma, odnosno ROC AUC iznosi 0.7483, a F1 0.6299. F1 mjera nešto je viša od *Adam*-a jer je odaziv nešto veći, a iznosi 0.5569. U odnosu na dosad najbolji algoritam *Adam, NAdam* ima nešto nižu preciznost.

```
Algoritam: Nadam
Accuracy:  0.8547603488933716
ROC AUC: 0.7483288214867967
Confusion matrix
    P0          P1
S0  25735       1652
S1  3460       4350
Recall:  0.5569782330345711
Precision:  0.724758413862046
F1 score:  0.6298870547350129
Training time (sec): 65.24526047706604
```

Slika 82 Rezultati testiranja NAdam

Funkcija gubitka pada do oko 18. epohe nakon čega je uočena pojava prenaučenosti.



Slika 83 Točnost i vrijednost funkcije gubitka po epohama NAdam

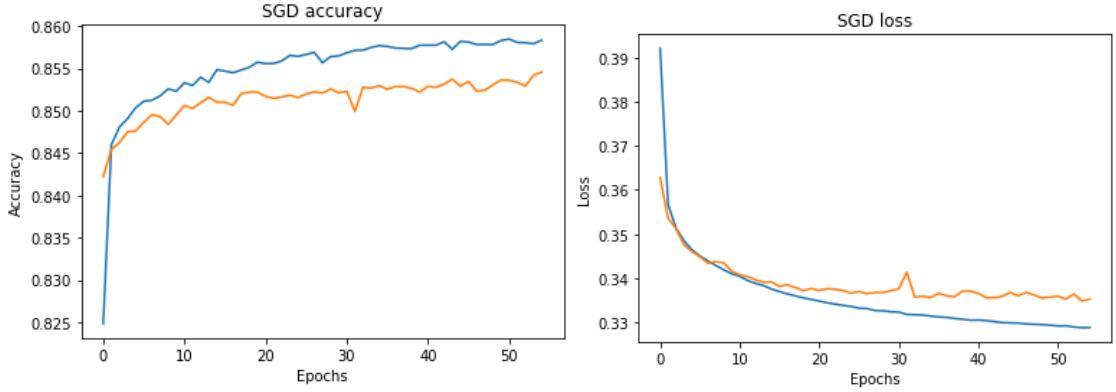
10.3.8. Algoritam SGD

I u ovom slučaju *SGD* ima točnosti na razini *Adam* i *NAdam* algoritma uz nešto dulje vrijeme treniranja. Treniranjem kroz 55 epoha SGD došao je do ROC AUC-a od 0.7396 i vrijednosti F1 mjere od 0.619. Kao i *Adam* i *NAdam* imao je odaziv veći od 0.5.

```
Algoritam: SGD
Accuracy:  0.854561468306958
ROC AUC: 0.7395516938955354
Confusion matrix
    P0          P1
S0  25917       1470
S1  3649        4161
Recall:  0.5327784891165173
Precision:  0.738945125199787
F1 score:  0.6791503608362473
Training time (sec): 180.92154145240784
```

Slika 84 Rezultati testiranja SGD

Vrijendost funkcije gubitka padala je do 55. epohe.



Slika 85 Točnost i vrijednost funkcije gubitka po epohama SGD

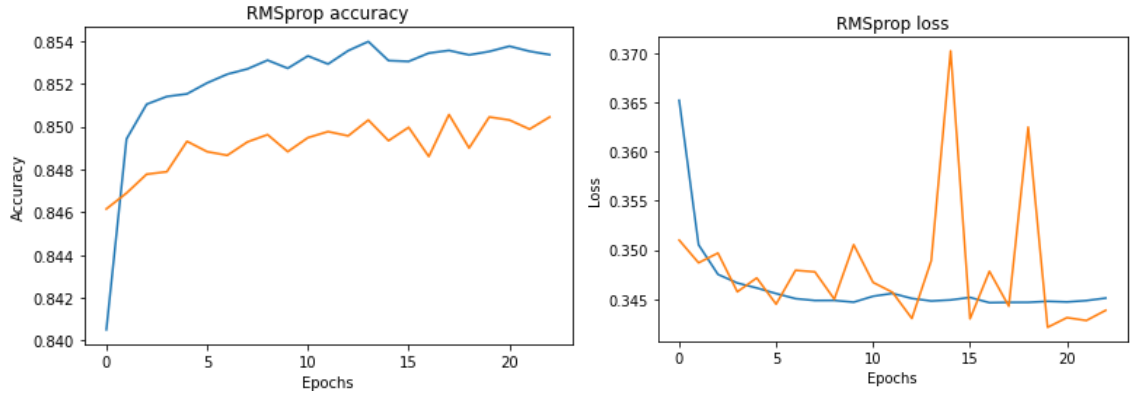
10.3.9. Algoritam RMSProp

Ovaj algoritam imao je nešto slabije ukupne performanse od *Adam*, *NAdam* i *SGD* algoritama. ROC AUC iznosio je 0.7214 uz F1 mjeru od 0.5923 nakon 23 epohe treniranja. S druge strane algoritam je imao najmanje krivih pozitivnih klasifikacija od ispitanih algoritama.

```
Algoritam: RMSprop
Accuracy:  0.8504417990169617
ROC AUC: 0.7214363390165162
Confusion matrix
    P0          P1
S0  26110       1277
S1  3987        3823
Recall:  0.48950064020486556
Precision:  0.7496078431372549
F1 score:  0.5922540666150271
Training time (sec): 78.91029596328735
```

Slika 86 Rezultati testiranja RMSProp

Kod funkcije gubitka uočeno je vrlo veliko osciliranje čak i kod nekoliko različitih ponavljanja treniranja. Funkcija se donekle ustabilila oko 22. epohe treniranja.



Slika 87 Točnost i vrijednost funkcije gubitka po epohama RMSProp

10.3.10. Zaključak na performanse algoritama

Kao što možemo vidjeti kod ovog skupa podataka nijedan algoritam nije pokazao znatno bolje performanse od drugih.

	Algoritam	Accuracy	ROC AUC	Recall	Precision	F1 score	Training time (sec)	Test time (sec)	Epochs
0	Adam	0.854107	0.734271	0.518822	0.746362	0.612131	67.956059	0.267243	19
1	Adamax	0.842884	0.712919	0.479257	0.718978	0.575138	108.921494	0.247226	35
2	Adagrad	0.844078	0.719635	0.495903	0.714049	0.585311	505.925721	0.262239	150
3	Nadam	0.854760	0.748329	0.556978	0.724758	0.629887	65.245260	0.260237	18
4	SGD	0.854561	0.739552	0.532778	0.738945	0.619150	180.921541	0.270246	55
5	RMSprop	0.850442	0.721436	0.489501	0.749608	0.592254	78.910296	0.264241	23

Slika 88 Ukupne performanse algoritama

Glavna karakteristika ovog skupa bila je šum u podacima. Veliki broj nedostajućih vrijednosti uvelike je otežao svim algoritmima predviđanje, pogotovo u pronalasku pozitivnih klasifikacija. Odaziv kod svih algoritama bio je vrlo nizak, oko 0.5. Što se tiče odaziva najboljim se pokazao *NAdam* sa odazivom od 0.557, a još su odaziv veći od 0.5 imali i *Adam* te osnovni *SGD* algoritam. Najslabijim se pokazao *Adamax* algoritam koji je imao najslabiji odaziv i drugu najslabiju preciznost pa samim time i najslabiju F1 mjeru. U vidu odaziva nešto bolji od *Adamax*-a pokazao se *Adagrad*, ali ponovno uz vrlo dugo vrijeme treniranja od 150 epoha i 506 sekundi. *RMSProp* imao je najveću preciznost od 0.7496, ali i vrlo slabi odaziv od 0.4895.

Uzimajući u obzir ukupne performanse ponovno su se najboljim pokazali *Adam* i *NAdam*, ne samo u pogledu sličnih točnosti, već i brzine treniranja pošto konvergencija prema minimumu ovih je algoritama daleko najbrža. *SGD* je također imao točnosti na razini ova dva algoritama, ali zbog nevarijabilne stope učenja algoritam puno sporije konvergira prema minimumu zbog čega je potrebno puno dulje vrijeme treniranja. Ipak, zbog nevarijabilne stope učenja stopa učenja kod SGD-a nešto je stabilnija kod pada prema minimumu pa imamo konzistentnije rezultate kod ponovnih provođenja treniranja modela (što je slučaj i kod *Adagrad* algoritma).

10.4. Cat/dog detection

10.4.1. Opis domene

Sljedeći skup podataka nad kojim će se ispitati performanse nešto je drugačiji od prethodnih, no i dalje se nastoji rješavati problem binarne klasifikacije. Ovaj skup podataka sastoji se od slika mačaka i pasa te će se slike pokušati klasificirati u dvije klase, slika mačke ili psa. [8]

10.4.2. Analiza i čišćenje skupa podataka

Originalni skup podataka sadržava 12 500 slika mačaka i 12 500 pasa u .jpg formatu. U ovoj analizi skup je smanjen na po 3999 slika za klasu mačka i 3999 slika klase pas što znači da se skup podataka za testiranje sastoji od 7998 slika. U testni skup podataka uzeto je 1671 slika (različitih od slika u trening skupu podataka) od čega je 853 slika mački i 818 slika pasa. Glavni razlog smanjenja veličine skupa podataka jest računalna kompleksnost treniranja modela neuronskih mreža. Za razliku od prethodno opisanih skupova podataka kod ovog skupa dodatno se odvija proces konvolucije. U ovom radu neće se detaljno ulaziti u vrlo opširnu teorijsku pozadinu konvolucije u domeni konvolucijskih neuronskih mreža.

Ukratko konvolucijska neuronska mreža sastoji se od dva dijela, slojeva za ekstrakciju značajki i već detaljno opisanih skrivenih slojeva koji služe za učenje i izlaznog sloja koji daje predviđenu vrijednost. Kako je slika jednostavno skup piksela koje računalo „vidi“, konvolucijski slojevi iz cijele slike, odnosno piksela, izvlače glavne značajke slike. Izlaz konvolucije je mapa značajki na temelju koje mreža može učiti i dati predviđanje. (Cornelisse, 2018)

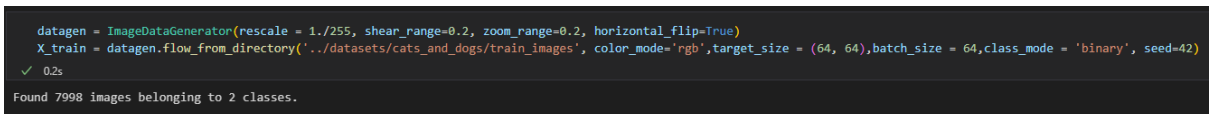
Naravno sam proces je daleko kompleksniji od opisanog na temelju kojeg bi se mogao napisati cijeli diplomski rad. U ovom radu analizirat će se rad algoritama učenja nad ovakvim, dosta drugačijim, ali i kompleksnijim skupom podataka.

Samo čišćenje skupa podataka u ovom slučaju izrazito je jednostavno. Sve što je potrebno napraviti jest učitati skup podataka pomoću *keras/tensorflow* modula „*ImageDataGenerator*[9]“. Osim učitavanja slika u oblik pogodan za treniranje, rade se određene izmjene na slikama u obliku rotiranja, zumiranja itd. Razlog tome jest sprečavanje

[8] Poveznica na skup podataka: https://www.microsoft.com/en-us/download/details.aspx?id=54765

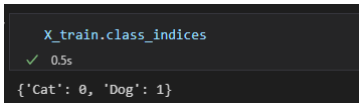
[9] Poveznica: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

„prenaučenosti“, to jest dodaje se skupu podataka „dobar“ šum. Pronalazak značajki iz raznih oblika slika omogućava mreži puno bolju ekstrakciju značajki iz slika.



Slika 89 Učitavanje i transformacija trening slika

Na početku se provodi skaliranje RGB vrijednosti piksela slike u raspon od 0-1 (faktorom 1/255 pošto svaka od vrijednosti RGB ima raspon od 0-255). U istom koraku provode se spomenute transformacije slika u obliku zumiranja, smicanja i okretanja. Također slike se postavljaju na veličinu 64x64. Što je slika veća, rezultati bi trebali biti bolji, no to dolazi sa velikim negativnim utjecajem na računalnu kompleksnost. 64x64 je vrlo niska veličina slike, no zbog dostupne računalne snage sve više od navedene veličine slika dovodi do vrlo dugog vremena treniranja mreža. U direktoriju gdje se nalazi trening skup podataka slike moraju biti odvojene prema nazivu kategorije u dvije mape, u ovom slučaju mapa „*Cat*“ sadrži sve slike mačaka, a mapa „*Dog*“ sve slike pasa. Na taj način pozivom funkcije *flow_from_directory* automatski se definiraju klase koje treba raspoznati. U ovom slučaju klasa „*Cat*“ označena je kao vrijednost za predikciju 0, a „*Dog*“ kao vrijednost 1.



Slika 90 Preslika klasa

Isti postupak izvodi se za testni skup slika, ne uključujući spomenute transformacije iz razloga što želimo vidjeli koliko dobro model radi s stvarnim, neizmijenjenim slikama.

Slika 91 Učitavanje testnih slika

10.4.3. Postavke i treniranje

Za razliku od prethodnih mreža, ova struktura nešto je drugačija pošto imamo konvolucijsku neuronsku mrežu. Ono što je dodano jest dio koji se odnosi na konvoluciju. Na početku mreže imamo tri sloja konvolucije sa pripadnim slojevima sažimanja koji smanjuju dimenzionalnost izlaznih vrijednosti konvolucije. Prvi sloj konvolucije ima 32 izlazna filtera uz „*kernel_size*“ (veličinu konvolucijskog prozora) od 3 sa 3. Prvi sloj također ima zadani *input_shape* govori koje su dimenzije ulaznih podataka, a već je opisano da su sve u RBG slike postavljene na dimenzije 64x64. Nakon svakog konvolucijskog sloja imamo sloj

sažimanja koji smanjuje dimenzionalnost izlaznih vrijednosti konvolucijskog sloja, a prozor sažimanja dimenzije je 2 sa 2. Definirana su još dva konvolucijska sloja od po 64 i 128 filtera sa pripadnim slojevima sažimanja. Na kraju konvolucijskog dijela mreže imamo „*Flattening*“ sloj koji jednostavno uzima finalnu mapu značajki te ju pretvara u jednodimenzionalni vektor koji je moguće propagirati kroz skrivene slojeve mreže. U nastavku mreže imamo dva skrivena sloja sa 64 neurona i aktivacijskom funkcijom „ReLU“ te je na kraju ponovno izlazni neuron sa funkcijom „Sigmoida“. Funkcija gubitka je ponovno binarna unakrsna entropija. Veličina uzorka za propagiranje, u ovom slučaju, definirana je kod učitavanja slika te iznosi 64 slike.

```
cnn_adam = tf.keras.Sequential()
#tri konvolucijska sa po 32, 64 i 128 filtera i 3 sloja sažimanja
cnn_adam.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
cnn_adam.add(tf.keras.layers.MaxPooling2D((2, 2)))
cnn_adam.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
cnn_adam.add(tf.keras.layers.MaxPooling2D((2, 2)))
cnn_adam.add(tf.keras.layers.Conv2D(128, (3, 3), activation='relu'))
cnn_adam.add(tf.keras.layers.MaxPooling2D((2, 2)))
#sloj za spljoštavanje mape
cnn_adam.add(tf.keras.layers.Flatten())
#dva skrivena sloja sa 64 neurona
cnn_adam.add(tf.keras.layers.Dense(64, activation='relu'))
cnn_adam.add(tf.keras.layers.Dense(64, activation='relu'))

cnn_adam.add(tf.keras.layers.Dense(1, activation='sigmoid'))
cnn_adam.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy'])
```

Slika 92 Struktura neuronske mreže

Kod konvolucijskih mreža imamo još više parametara za podešavanje, a također kako je ovo puno kompleksniji problem od prethodnih veći broj skrivenih slojeva mogao bi poboljšati točnosti. Povećavanje broja skrivenih slojeva sa sobom donosi veću računalnu kompleksnost. Korištenje grafičke kartice za treniranje ovakvih mreža jest gotovo neophodno, no kako se treniranja u ovom radu odvijaju na procesoru bilo je potrebno žrtvovati potencijalne veće točnosti za „razumnije“ vrijeme treniranja.

10.4.4. Algoritam Adam

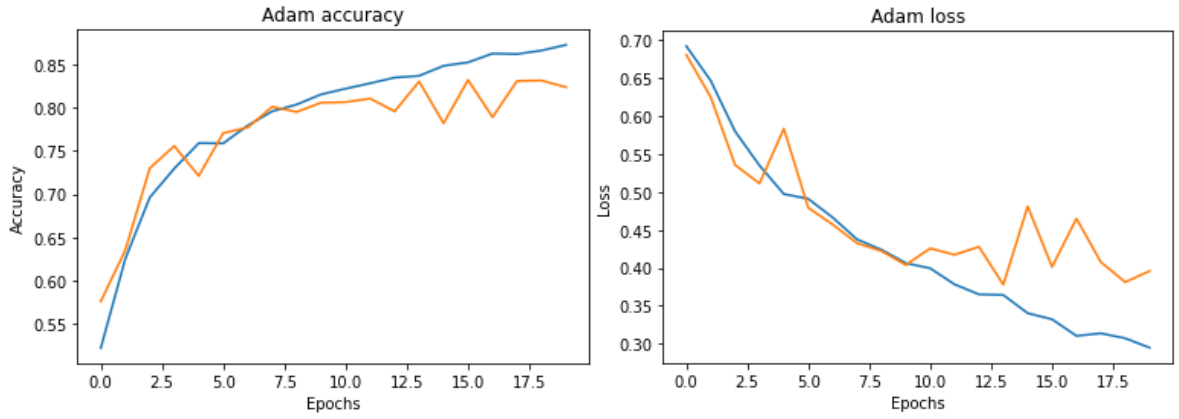
Prvi algoritam ponovno jest *Adam*. Već smo od prije mogli zaključiti da bi korištenje ovog algoritma moglo u ovom slučaju trebalo biti prioritet zbog izrazito velike brzine treniranja u odnosu na ostale algoritme. Upravo to i jest slučaj ovdje.

```
Algoritam: Adam
Accuracy:  0.8240574506283662
ROC AUC: 0.8259866657876559
Confusion matrix
    P0          P1
S0  626        227
S1  67        751
Recall:  0.9180929095354523
Precision:  0.7678936605316974
F1 score:  0.8363028953229398
Training time (sec): 416.2949547767639
```

Slika 93 Rezultati testiranja Adam

Adam je postigao točnosti od 0.824 nakon 20 epoha treniranja što je iznosilo 416 sekundi, odnosno skoro 7 minuta. Model je imao vrlo visoki odaziv od 0.92 uz nešto nižu preciznost od 0.77. F1 mjera zbog toga iznosi 0.836. Klasa 0 u ovom slučaju jest „mačka“, a klasa 1 „pas“ zbog toga možemo malo drugačije protumačiti vrijednosti u matrici konfuzije. Model je u 626 slučaja na slici točno prepoznao mačku i u 751 slučajeva psa. U 67 slučaja krivo je odredio da je na slici pas, a bila je mačka, dok je u nešto više slučajeva, njih 227, kada je na slici bila mačka, odredio da je pas.

Validacijska funkcija gubitka padala je do 13 epohe nakon čega je oscilirala oko svojeg minimuma do 20. epohe nakon koje je počela rasti zbog čega je na 20. epohi prekinuto treniranje.



Slika 94 Točnost i vrijednost funkcije gubitka po epohama Adam

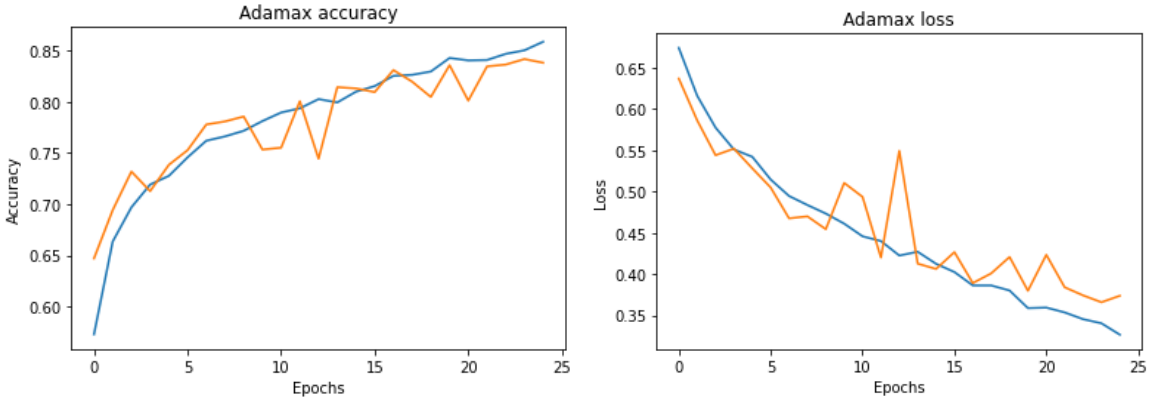
10.4.5. Algoritam Adamax

Algoritmom „Adamax“ model je treniran 25 epoha te je dao malo bolju točnost od 0.838. Također imao je sličnije vrijednosti odaziva i preciznosti od 0.87 i 0.81. Iz takvih vrijednosti odaziva i preciznosti vidimo da je napravio malo više krivih detekcija mačke, ali manje krivih detekcija psa. *Adamaxom* je treniranje trajalo dulje, odnosno malo manje od 9 minuta.

```
Algoritam: Adamax
Accuracy:  0.8384201077199281
ROC AUC: 0.8391266836162888
Confusion matrix
    P0          P1
S0  687        166
S1  104        714
Recall:  0.8728606356968215
Precision:  0.8113636363636364
F1 score:  0.8409893992932862
Training time (sec): 525.0796406269073
```

Slika 95 Rezultati testiranja Adamax

Vidimo nešto sporiji pad funkcije gubitka, no vrijednost validacijske funkcije nešto dulje je pratila vrijednost trening funkcije pa je zato točnost malo viša nego kod „Adam-a“.



Slika 96 Točnost i vrijednost funkcije gubitka po epohama Adamax

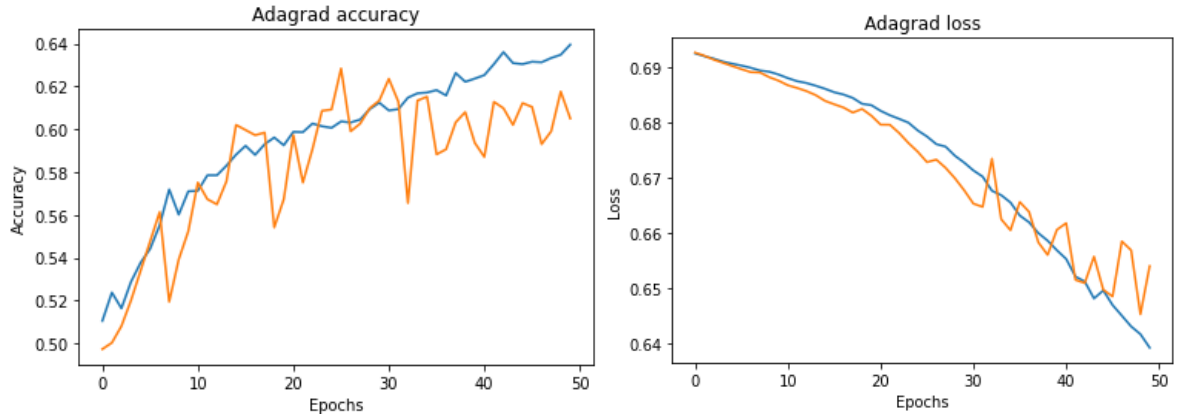
10.4.6. Algoritam Adagrad

Negativne strane jednostavnosti ovog algoritma mogu se vrlo dobro vidjeti na ovakvom skupu podataka. Spora konvergencija prema minimumu ovog algoritma donosi vrlo slabe rezultate kod ovako kompleksnog skupa podataka. Model treniran „Adagrad“ algoritmom ne samo što je dao najslabije rezultate, već je i pušten na treniranje do 50. epohe nakon čega je zbog vrlo dugog treniranja ipak zaustavljen. Ukupna točnost daleko je najniža od svih algoritama te iznosi 0.605. Odaziv iznosi dobrih 0.861, no preciznost je vrlo niska te iznosi 0.56. Vidimo da je algoritam zbog nedovoljne „naučenosti“ napravio previše krivih klasifikacija mačke kao psa. Veći problem jest ukupno trajanje treniranja, a iznosi 1040 sekundi, odnosno nešto više od 17 minuta.

```
Algoritam: Adagrad
Accuracy:  0.6050269299820467
ROC AUC: 0.6102709550930556
Confusion matrix
    P0          P1
S0  307        546
S1  114        704
Recall:  0.8606356968215159
Precision:  0.5632
F1 score:  0.6808510638297872
Training time (sec): 1040.2026207447052
```

Slika 97 Rezultati testiranja Adagrad

Može se uočiti da do 50. epohe algoritam nije stigao do stagnacija funkcija gubitka što znači da bi daljnje treniranje dovelo do boljih rezultata, ali i već predugo treniranje govori da se algoritam ne nosi dobro sa ovakvim skupom podataka.



Slika 98 Točnost i vrijednost funkcije gubitka po epohama Adagrad

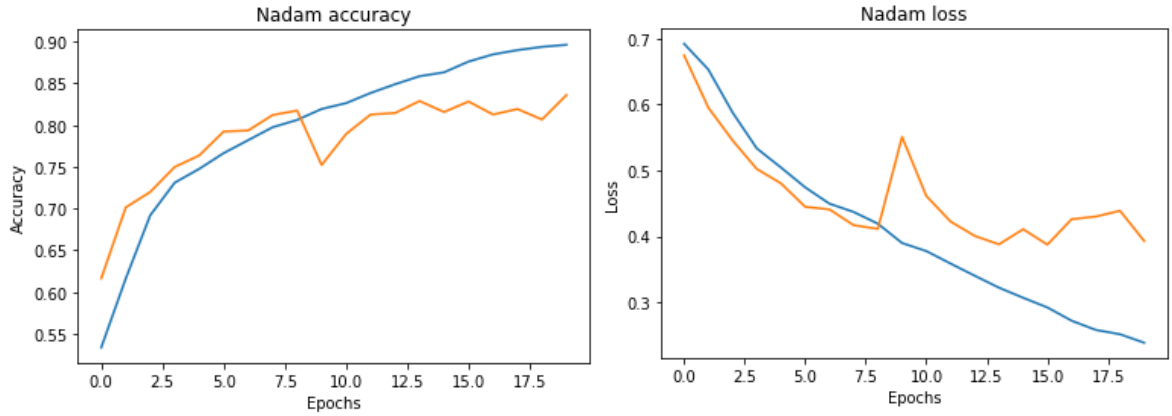
10.4.7. Algoritam NAdam

Kao i do sada u samom vrhu prema performansama je *NAdam* algoritam. Ovaj algoritam je na razini već opisanih *Adamax* i *Adam* algoritama. Ukupna točnost iznosi 0.836, dok je odaziv 0.856 s preciznosti od 0.818. Kako imamo nešto nižu vrijednost odaziva kod ovog algoritma u odnosu na *Adamax*, F1 mjera iznosi malo manjih 0.836. Iako je algoritmom model treniran kroz isti broj epoha kao i *Adam*-om, vrijeme treniranja je 11 sekundi dulje te iznosi 427 sekundi. Čini se da *NAdam*, zbog kompleksnijeg računanja stope učenja, ima malo sporije vrijeme učenja od *Adam*-a.

```
Algoritam: Nadam
Accuracy:  0.8360263315380012
ROC AUC: 0.8364308911163533
Confusion matrix
    P0          P1
S0  697         156
S1  118         700
Recall:  0.8557457212713936
Precision:  0.8177570093457944
F1 score:  0.8363201911589009
Training time (sec): 427.00253462791443
```

Slika 99 Rezultati testiranja NAdam

Vidimo da je vrijednost validacijske funkcije gubitka kod *NAdam*-a počela stagnirati oko 13 epohe.



Slika 100 Točnost i vrijednost funkcije gubitka po epohama NAdam

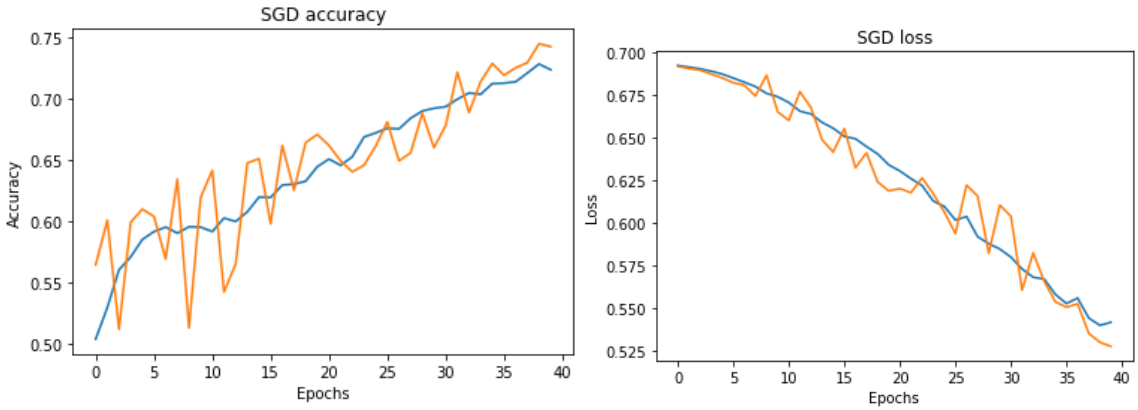
10.4.8. Algoritam SGD

SGD je do sada davao odlične rezultate, no uvijek je imao dulja vremena treniranja u odnosu na druge algoritme (ne uključujući *Adagrad*). Ovim algoritmom model je treniran 40 epoha, te je daje bolje rezultate od *Adagrad* algoritma koji je treniran 50 epoha. Ukupna točnost ovog algoritma nakon 40 epoha treniranja jest 0.743 uz odaziv od 0.77 i preciznost od 0.72. Vrijeme treniranja iznosilo je 844 sekunde odnosno 14 minuta.

```
Algoritam: SGD
Accuracy:  0.7426690604428486
ROC AUC: 0.7432332885228891
Confusion matrix
    P0          P1
S0  611         242
S1  188         630
Recall:  0.7701711491442543
Precision:  0.7224770642201835
F1 score:  0.7455621301775149
Training time (sec): 844.1893692016602
```

Slika 101 Rezultati testiranja SGD

Kao i kod *Adagrad* algoritma prema grafovima funkcija gubitka i točnosti vidimo da algoritam model ima još prostora za učenje, no zbog već i onako dugog trajanja treniranja ne može se preporučiti korištenje ovog algoritma kod ovakvih problema.



Slika 102 Točnost i vrijednost funkcije gubitka po epohama SGD

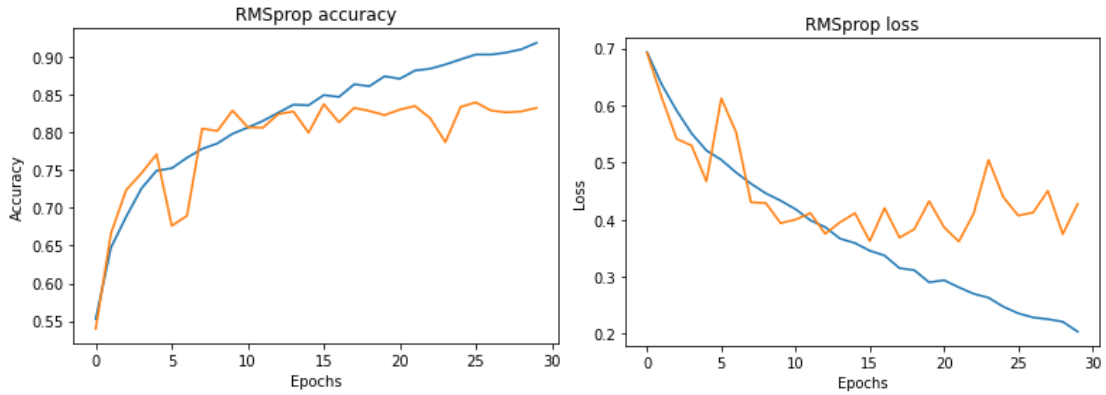
10.4.9. Algoritam RMSProp

Vrlo dobre rezultate imao je i algoritam RMSProp. Uz točnost od 0.832 model je imao odaziv od 0.835 i najvišu dosad preciznost od 0.825. Uz 135 krivih detekcija psa kao mačke, model je imao 145 krivih detekcija mačke kao psa. Treniranje je trajalo 30 epoha što iznosi 620 sekundi ili oko 10 minuta.

```
Algoritam: RMSprop
Accuracy:  0.8324356672651108
ROC AUC: 0.8324875242563998
Confusion matrix
    P0          P1
S0  708         145
S1  135         683
Recall:  0.8349633251833741
Precision:  0.8248792270531401
F1 score:  0.8298906439854192
Training time (sec): 619.8428914546967
```

Slika 103 Rezultati testiranja RMSProp

Grafovi funkcije gubitka i točnosti pokazuju početak stagnacije na 15. epohi što traje do 30. epohe nakon čega je validacijska točnost počela padati.



Slika 104 Točnost i vrijednost funkcije gubitka po epohama RMSProp

10.4.10. Zaključak na performanse algoritama

Možemo analizirati i ukupne performanse svih algoritama:

	Algoritam	Accuracy	ROC AUC	Recall	Precision	F1 score	Training time (sec)	Test time (sec)	Epochs
0	Adam	0.824057	0.825987	0.918093	0.767894	0.836303	416.294955	1.648499	20
1	Adamax	0.838420	0.839127	0.872861	0.811364	0.840989	525.079641	1.633486	25
2	Adagrad	0.605027	0.610271	0.860636	0.563200	0.680851	1040.202621	1.634488	50
3	Nadam	0.836026	0.836431	0.855746	0.817757	0.836320	427.002535	1.634487	20
4	SGD	0.742669	0.743233	0.770171	0.722477	0.745562	844.189369	1.647499	40
5	RMSprop	0.832436	0.832488	0.834963	0.824879	0.829891	619.842891	1.627481	30

Slika 105 Ukupne performanse algoritama kod detekcije mačke i psa

Ovaj skup podataka nešto je drugačiji od ostalih, a se sastoji od slika. Detekcija objekata na slikama kompleksniji je problem od ostalih problema koji se ispituju i rješavaju u ovom radu gdje smo kod jednostavnijih algoritama *Adagrad* i osnovnog algoritma gradijentnog pada *SGD* vidjeli veće probleme. Vrlo dugo treniranje kod ovih algoritma predstavlja veliki problem, a u teoriji spora konvergencija prema minimumu dolazi na vidjelo u praktičnom aspektu kod ovakvog problema. Iako *Adagrad* ima varijabilnu stopu učenja, koja bi trebala ubrzati treniranje, vidimo da to u ovom slučaj nema nikakve prednosti u odnosu na osnovni *SGD* algoritam koji je čak uz manje epoha treniranja došao do znatno boljih točnosti, iako je i *SGD* algoritam, u odnosu na najbolje algoritme, nakon 40 epoha imao slabije rezultate. Ostali algoritmi *Adam*, *Adamax*, *NAdam*, i *RMSProp* imali su vrlo slične točnosti i vrijednosti metrika. Možemo vidjeti da su *Adam* i *NAdam* ponovno među najboljim algoritmima uz najkraće vrijeme treniranja. Ukupno se najboljim pokazao *Adamax* koji ima malo bolju ukupnu točnost detekcije, no za razliku od dva najbrža treniranje je trajalo gotovo dvije minute dulje. Najbalansiranije rezultate dao je *RMSProp* algoritam koji je imao vrlo slične vrijednosti odaziva i preciznosti. Svakako možemo zaključiti da korištenje *SGD* i *Adagrad* algoritama kod konvolucijskih

neuronskih mreža nema smisla zbog izrazito spore konvergencije prema minimumu što za rezultate daje predugo vrijeme treniranja.

10.5. House Sales value prediction in King County

10.5.1. Opis domene

Posljednja dva skupa podataka koji će se analizirati odnose se na predviđanje cijene nekretnina. Predviđanje cijene odnosi se na regresijski problem, odnosno problem u kojem želimo predvidjeti konkretnu vrijednost kontinuirane numeričke varijable. Ovaj skup podataka sadrži podatke o prodanim nekretninama u okrugu King u okolici Seattle-a. Podaci su prikupljani između svibnja 2014. i svibnja 2015. godine.[10]

10.5.2. Analiza i čišćenje skupa podataka

Što se tiče osnovnih karakteristika skupa podataka, on se sastoji od 21 varijable od čega su njih 20 numeričke varijable, kontinuirane i diskretne, a jedna je binarna kategorijska varijabla. Skup podataka srednje je veličine, odnosno sastoji se od 21 613 instanci.

Zavisna varijabla ovog skupa jest „*price*“, odnosno cijena nekretnine. Nezavisnih varijabli je 20 te se odnose na razne podatke o prodanoj nekretnini. „*id*“ je jednostavno id instance te ne sadrži nikakve prediktivne informacije o cijeni. „*date*“ se odnosi na datum prodaje. „*bedrooms*“ i „*bathrooms*“ sadrže informacije o broju spavaćih soba, odnosno kupaonica. Varijabla „*sqft_living*“ govori kolika je kvadratura nekretnine, a „*sqft_lot*“ kvadratura cijelog zemljišta. „*floors*“ govori koliki je broj katova u kući, a „*waterfront*“ je li nekretnina na obali. Ostatak varijabli daje informacije o stanju nekretnine, ocjeni, kvadraturi katova i podruma, godini izgradnje, godini renovacije, geografskoj širini i visini i poštanskom broju.

Što se tiče zavisne varijable možemo detaljnije analizirati deskriptivnu statistiku ove varijable.

```
pd.DataFrame(data= dataset['price'].describe().to_numpy().reshape(8,1).T, columns=['count','mean','std','min','25%','50%','75%','max'])
```

✓ 0.6s

	count	mean	std	min	25%	50%	75%	max
0	21613.0	540088.141767	367127.196483	75000.0	321950.0	450000.0	645000.0	7700000.0

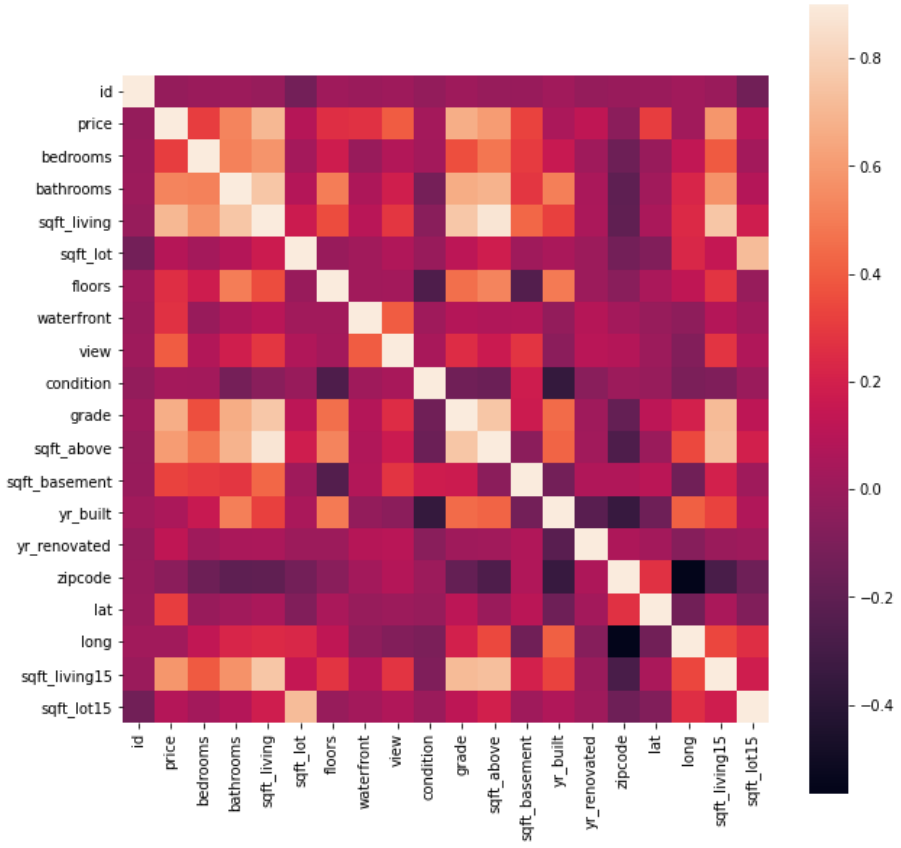
Slika 106 Deskriptivna statistika zavisne varijable

Prosječna cijena nekretnina iznosi 540 088 dolara uz standardnu devijaciju od 367 127. Minimalna vrijednost neke nekretnine u skupu jest 75 000 dolara. Medijan iznosi 450 000,

[10] Poveznica na skup podataka: https://www.kaggle.com/datasets/harlfoxem/housesalesprediction

odnosno pola nekretnina u skupu košta manje od 450 000 dolara, a pola više. Maksimalna vrijednost neke nekretnine iznosi 7 700 000 dolara.

Možemo pogledati korelacije između varijabli.



Slika 107 Matrica korelacija

Ono što nas detaljnije zanima koja je korelacija nezavisnih varijabli s zavisnom varijablom „*price*“. Na sljedećoj slici imamo sortirane vrijednosti korelacija nezavisnih varijabli s varijablom „*price*“. Vidimo da postoji nekoliko varijabli s većom pozitivnom korelacijom. Najveću pozitivnu korelaciju s varijablom „*price“* ima „*sqft_living*“ od čak 0.7 što je i očekivano pošto će površinski veći stanovi često biti skuplji. Korelaciju veću od 0.5 imaju i varijable „*grade*“, „*sqft_above*“, „*sqft_living15*“ i „*bathrooms*“. Nijedna varijabla nema znatnu negativnu korelaciju s varijablom „*price*“.

```
print(korelacijska_matrica['price'].drop(['price']).sort_values())
✓ 0.8s
zipcode         -0.053203
id              -0.016762
long             0.021626
condition        0.036362
yr_built         0.054012
sqft_lot15       0.082447
sqft_lot         0.089661
yr_renovated     0.126434
floors           0.256794
waterfront       0.266369
lat              0.307003
bedrooms         0.308350
sqft_basement    0.323816
view             0.397293
bathrooms        0.525138
sqft_living15    0.585379
sqft_above       0.605567
grade            0.667434
sqft_living      0.702035
```

Slika 108 Korelacije nezavisnih varijabli sa zavisnom

Što se tiče čišćenja skupa nema previše posla. Nema vrijednosti koje nedostaju ni za jednu varijablu.

```
dataset.isna().sum()
✓ 0.4s
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
```

Slika 109 Ukupni broj vrijednosti varijabli koje nedostaju

Što se tiče outlier vrijednost, vrijednosti nekih instanci izgledaju kao potencijalni outlier-i. Jedan možemo uočiti iz same maksimalne cijene nekretnine od preko 7 milijuna dolara, ali naravno ne mora biti slučaj da je ovo neispravni unos. Ako pogledamo vrijednosti nezavisnih varijabli ove instance vidimo da većina atributa ima dosta visoke vrijednosti pa bi mogli

zaključiti da se radi o vrlo velikoj kući. Npr. ova nekretnina ima 6 soba, 8 kupaonica, kvadratura iznosi velikih 12 050 kvadratnih stopa pa bi mogli zaključiti da ovo nije outlier.

```
dataset[dataset['price']==dataset['price'].max(axis=0)].T
✓ 0.6s
```

	7252
id	6762700020
date	20141013T000000
price	7700000.0
bedrooms	6
bathrooms	8.0
sqft_living	12050
sqft_lot	27600
floors	2.5
waterfront	0
view	3
condition	4
grade	13
sqft_above	8570
sqft_basement	3480
yr_built	1910
yr_renovated	1987
zipcode	98102
lat	47.6298
long	-122.323
sqft_living15	3940
sqft_lot15	8800

Slika 110 Instanca s vrijednosti cijene od 7 700 000 dolara

Još jedan potencijalni outlier jest nekretnina sa brojem soba od 33. 33 spavaćih soba na 1620 kvadratnih stopa (150 metara kvadratnih) izgleda nerealno pa bi mogli zaključiti da je ovo outlier u skupu podataka.

	15870
id	2402100895
date	20140625T000000
price	640000.0
bedrooms	33
bathrooms	1.75
sqft_living	1620
sqft_lot	6000
floors	1.0
waterfront	0
view	0
condition	5
grade	7
sqft_above	1040
sqft_basement	580
yr_built	1947
yr_renovated	0
zipcode	98103
lat	47.6878
long	-122.331
sqft_living15	1330
sqft_lot15	4700

Slika 111 Instanca s 33 spavaće sobe

Skup podataka nema previše šuma zbog čega se neće djelovati na outlier vrijednosti u ovom slučaju. Prije treniranja modela skup je podijeljen na skup nezavisnih varijabli i zavisnu varijablu. Iz skupa su izbačene varijable „*id*“ i „*date*“ koje nemaju utjecaj na vrijednost cijene

nekretnine. Nakon toga provedena je standardizacija skupa te je skup podijeljen na testni i trening od čega je 25% originalnog skupa otišlo na testni podskup.

Možemo zaključiti da je ovaj skup podataka srednje veličine prema broju instanci srednje velike dimenzionalnosti u odnosu na ostale skupove podataka analizirane u ovom radu. Varijable su pretežito numeričke sa kombinacijom kontinuiranih i diskretnih tipova. Skup podataka nema previše šuma, a postoji nekoliko varijabli s znatnom korelacijom s zavisnom varijablom „*price*“. Isto tako, u odnosu na prethodne skupove, u ovom slučaju rješava se problem regresije pa će metrike biti drugačije od prethodnih.

10.5.3. Postavke i treniranje

Struktura neuronske mreže koja se pokazala najboljom sastoji se od tri skrivena sloja sa po 8, 16 i 32 neurona. Aktivacijske funkcije u skrivenim slojevima su „ReLU“. Na izlaznom sloju imamo jedan neuron koji u ovom slučaju ima linearnu aktivacijsku funkciju pošto se nastoji predvidjeti kontinuirana numerička varijabla. Kako imamo regresijski problem potrebno je koristiti drugačiju funkciju gubitka, a u opisu metrika ukratko ih je opisano nekoliko. Funkcija gubitka u ovom slučaju jest prosječna apsolutna greška.

```
ann_adam = tf.keras.models.Sequential()
#3 skrivena sloja sa po 8 16 i 32 neurona
ann_adam.add(tf.keras.layers.Dense(units=8, activation='relu'))
ann_adam.add(tf.keras.layers.Dense(units=16, activation='relu'))
ann_adam.add(tf.keras.layers.Dense(units=32, activation='relu'))
# Dodavanje izlaznog sloja
ann_adam.add(tf.keras.layers.Dense(units=1, activation='linear'))
# kompajliranje mreže
ann_adam.compile(optimizer = 'adam', loss = 'mean_absolute_error', metrics = ['mean_squared_error'])
```

Slika 112 Struktura neuronske mreže

Veličina uzorka za propagaciju jest 16 instanci kod svih algoritama.

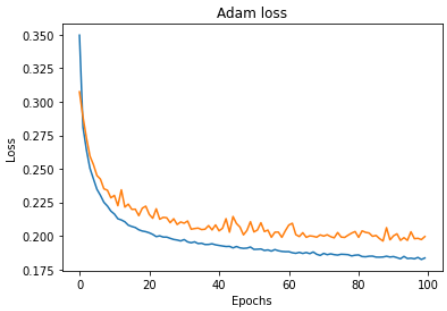
10.5.4. Algoritam Adam

Ponovno je prvi testirani algoritam bio *Adam*. Uz najbrže vrijeme treniranja ponovno je imao i najbolje vrijednosti svih metrika. Model je treniran 100 epoha te je treniranje trajalo 55 sekundi Koeficijent determinacije iznosi vrlo visokih 0.893, odnosno 89.3% varijacije podataka oko aritmetičke sredine protumačeno je ovim modelom. Od ostalih metrika onu koji najlakše možemo interpretirati jest prosječna apsolutna greška koja iznosi 71906.88. To nam govori da je prosječna greška koju model napravi kod predikcije 71 906 dolara. U odnosu na to da je prosječna vrijednost nekretnina oko 500 tisuća dolara, ova greška nije prevelika.

```
Algoritam: Adam
R2 score:0.8931007754893622
Mean Absolute Error:71906.88167329755
Mean Squared Error:15990801036.0741
Root Mean Squared Error:126454.73908111986
Training time (sec): 55.08686637878418
```

Slika 113 Rezultati testiranja Adam

Vidimo da vrijednosti funkcije gubitka (MAE) pada do 100. epohe nakon čega je uočen postepeni rast validacijske funkcije gubitka.



Slika 114 Točnost i vrijednost funkcije gubitka po epohama Adam

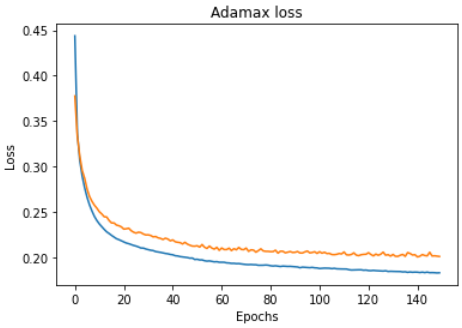
10.5.5. Algoritam Adamax

Vrlo slične rezultate imao je i algoritam *Adamax* uz nešto dulje treniranje do 150.epohe. Koeficijent determinacije kod ovog algoritma iznosi također vrlo visokih 0.888. Prosječna greška kod ovog algoritma veća je od one kod Adam-a za oko 600 dolara te iznosi 72 513.

```
Algoritam: Adamax
R2 score:0.8878894381731329
Mean Absolute Error:72513.39038212436
Mean Squared Error:16770352604.733025
Root Mean Squared Error:129500.39615666441
Training time (sec): 80.18430185317993
```

Slika 115 Rezultati testiranja Adamax

Ovdje vidimo da funkcije gubitka stagnira oko 150.epohe.



Slika 116 Točnost i vrijednost funkcije gubitka po epohama Adamax

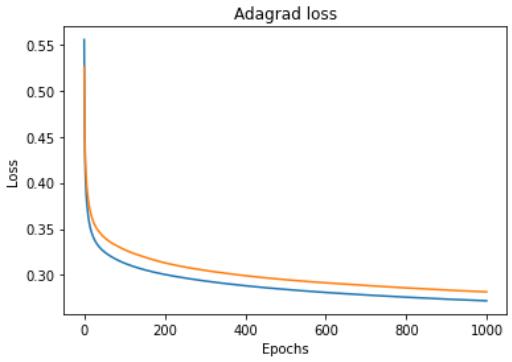
10.5.6. Algoritam Adagrad

Kao i do sad najsporiji algoritam jest *Adagrad*. *Adagrad* algoritam pušten je na treniranje kroz 1000 epoha i 518 sekundi (8.6 minuta) nakon čega su točnosti, u odnosu na ostale algoritme i dalje vrlo niske. Koeficijent determinacije iznosi 0.77, dok je prosječna apsolutna greška nakon 1000 epoha treniranja dosta viša nego kod ostalih algoritama te iznosi 101552.

```
Algoritam: Adagrad
R2 score:0.7749314158877325
Mean Absolute Error:101552.51174477239
Mean Squared Error:33667474806.162216
Root Mean Squared Error:183486.988111316
Training time (sec): 518.1999402046204
```

Slika 117 Rezultati testiranja Adagrad

Može se vidjeti da vrijednost funkcije gubitka nakon 1000 epoha i dalje pada što znači da bi algoritam nastavkom treniranja imao bolje rezultate od trenutnih, no već i ovako dugo treniranje govori da je algoritam prespor te nema smisla nastaviti s treniranjem.



Slika 118 Točnost i vrijednost funkcije gubitka po epohama Adagrad

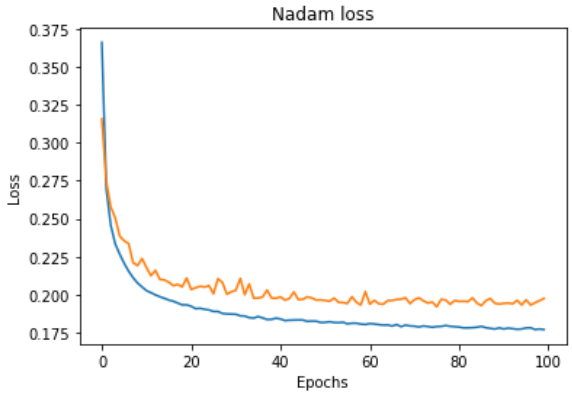
10.5.7. Algoritam NAdam

NAdam ponovno ima rezultate na razini *Adam* algoritma. Trening je trajao 100 epoha (56 sekundi) te je i ovaj algoritam imao vrlo slične rezultate kao i Adam, uz nešto manji koeficijent determinacije od 0.8879, ali malo manju prosječnu apsolutnu grešku od 71154. S druge strane ovaj model imao je malo veću korjenovanu prosječnu kvadriranu grešku od 129481. To nam govori da je ovaj model u nekim slučajevima napravio nekoliko većih grešaka od Adamovog modela pošto RMSE penalizira veće greške. Sve u svemu *NAdam* također ima vrlo dobre rezultate.

```
Algoritam: Nadam
R2 score:0.8879225662043917
Mean Absolute Error:71154.42397992229
Mean Squared Error:16765397061.238668
Root Mean Squared Error:129481.26142897538
Training time (sec): 57.21845245361328
```

Slika 119 Rezultati testiranja NAdam

Funkcija gubitka stagnira od 80 epohe te nakon 100. počinje rasti.



Slika 120 Točnost i vrijednost funkcije gubitka po epohama NAdam

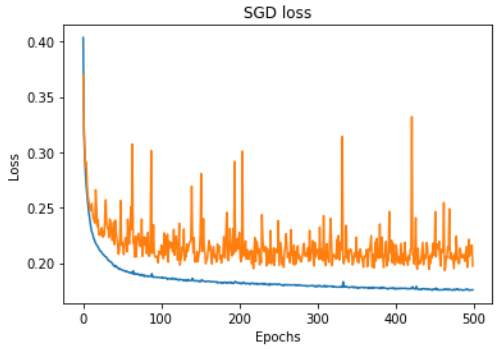
10.5.8. Algoritam SGD

SGD ponovno ima rezultate na razini ostalih algoritama uz dulje vrijeme treniranja od 500 epoha, odnosno 277 sekundi (4.6 minuta). Koeficijent determinacije iznosi vrlo visokih 0.8876 uz najmanji MAE od svih ispitanih algoritama od 71069. Kao i NAdam algoritam je napravio nešto više većih grešaka u odnosu na Adam pa RMSE iznosi nešto većih 129642.

```
Algoritam: SGD
R2 score:0.8876441063639068
Mean Absolute Error:71059.26100457994
Mean Squared Error:16807051207.245012
Root Mean Squared Error:129642.01173711018
Training time (sec): 277.10888934135437
```

Slika 121 Rezultati testiranja SGD

Zanimljivo je vidjeti vrijednosti validacijske funkcije gubitka po epohama. U odnosu na ostale algoritme vidimo vrlo velike oscilacije po epohama. Lako se moglo dogoditi da se algoritam zaustavi na visokoj vrijednosti funkcije gubitka što bi značilo slabije točnosti.



Slika 122 Točnost i vrijednost funkcije gubitka po epohama SGD

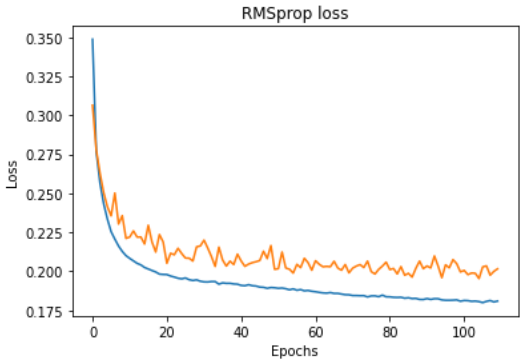
10.5.9. Algoritam RMSProp

Posljednji algoritam ponovno je *RMSProp* kojim je model treniran 120 epoha. Koeficijent determinacije vrlo sličan je kao i kod ostalih algoritama te iznosi 0.881. MAE iznosi 72529, a RMSE nešto većih 133195.

```
Algoritam: RMSprop
R2 score:0.881400592697404
Mean Absolute Error:72629.12808799038
Mean Squared Error:17741003584.019455
Root Mean Squared Error:133195.35871801033
Training time (sec): 70.2124502658844
```

Slika 123 Rezultati testiranja RMSProp

Vrijednost funkcije gubitka prosječno pada do oko 100.epohe nakon koje prosječno stagnira sve do iznad 120.epohe kada kreće s rastom.



Slika 124 Točnost i vrijednost funkcije gubitka po epohama RMSProp

10.5.10. Zaključak na performanse algoritama

Za kraj će se usporediti svi algoritmi.

	Algoritam	R2 score	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	Training time (sec)	Test time (sec)	Epochs
0	Adam	0.893101	71906.881673	1.599080e+10	126454.739081	55.086866	0.067354	100
1	Adamax	0.887889	72513.390382	1.677035e+10	129500.396157	80.184302	0.070063	150
2	Adagrad	0.774931	101552.511745	3.366747e+10	183486.988111	518.199940	0.071065	1000
3	Nadam	0.887923	71154.423980	1.676540e+10	129481.261429	57.218452	0.068061	100
4	SGD	0.887644	71059.261005	1.680705e+10	129642.011737	277.108889	0.068061	500
5	RMSprop	0.881401	72629.128088	1.774100e+10	133195.358718	70.212450	0.074067	110

Slika 125 Ukupne performanse algoritama

Kod ovog skupa podataka možemo vidjeti vrlo slične rezultate kod svih algoritama što se tiče točnosti/greške. Jedini algoritam koji se odvojio od ostatka (u negativnom smislu), jest *Adagrad* koji ponovno uz najdulje vrijeme treniranja ili i znatno slabije rezultate od ostatka. Ponovno bi mogli zaključiti da najbolje performanse imaju *Adam* i *NAdam* algoritmi zbog najmanjih grešaka uz najkraće vrijeme treniranja. Osim *Adagrad*-a svi algoritmi imali su vrlo male razlike u grešci, no ponovno zbog brzine treniranja *Adam* i *NAdam* su se odvojili od ostatka.

10.6. Real estate value prediction

10.6.1. Opis domene

Kao i prošli skup podataka, ovaj se odnosi na regresijski problem, odnosno predviđanje vrijednosti kontinuirane numeričke varijable. I u ovom slučaju radi se o predviđanju cijene nekretnina, no ovaj put predviđa se prosječna cijena nekretnina po predgrađima Bostona. [11]

10.6.2. Analiza i čišćenje skupa podataka

Za početak će se opisati osnovne karakteristike skupa podataka. Ovaj skup se sastoji od 14 atributa od kojih je 13 kontinuiranih varijabli, a jedna binarna kategorijska. Zavisna varijabla jest kontinuirana numerička varijabla. Skup je vrlo malen te sadrži svega 511 instanci.

Atributi u skupu daju različite informacije o nekretninama po predgrađima. Varijabla „*MEDV*“, čiju vrijednost želimo predvidjeti, govori kolika je medijalna cijena nekretnina u određenom predgrađu u tisućama dolara. Varijabla „CRIM“ označava stopu kriminala prema predgrađu, „ZN“ postotak zemljišta u vlasništvu države, „INDUS“ udio prostora na kojem se nalaze maloprodajna poduzeća u ukupnoj količini zemljišta u predgrađu, „NOX“ koncentraciju dušikovih oksida u zraku, „RM“ prosječni broj soba u stanu, „AGE“ udio naseljenih nekretnina izgrađenih prije 1940. godine, „DIS“ prosječnu udaljenost od centara za zapošljavanje, „RAD“ indeks pristupačnosti autocestama, „TAX“ stopu poreza na imovinu u tisućama dolara,

[11] Poveznica na skup podataka: https://www.kaggle.com/datasets/arslanali4343/real-estate-dataset

„PTRATIO“ omjer učenika i učitelja u predgrađu, „B“ udio afroameričkog stanovništva u ukupnom stanovništvu te „LSTAT“ postotak populacije slabijeg imovinskog statusa u ukupnoj populaciji.

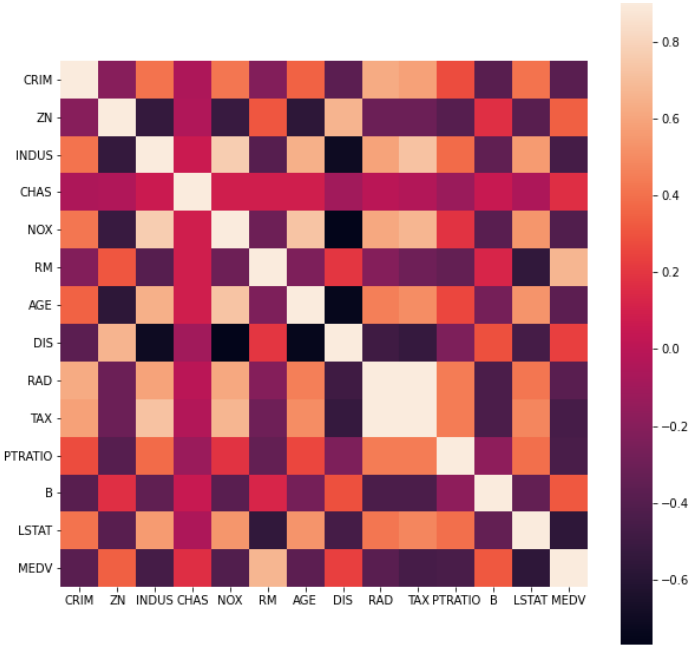
Možemo ponovno detaljnije analizirati deskriptivnu statistiku zavisne varijable „MEDV“.

count	mean	std	min	25%	50%	75%	max
511.0	22.682192	9.484262	5.0	17.05	21.2	25.0	67.0

Slika 126 Deskriptivna statistika varijable "MEDV"

Vidimo da je ukupan broj instanci 511. Prosjek vrijednosti ove varijable iznosi 22.682, odnosno prosječno u skupu podataka nekretnine u predgrađima koštaju 22 682 dolara. Najmanja medijalna vrijednost nekretnina u nekom predgrađu iznosi 5, odnosno 5000 dolara, a najviša 67 000 dolara. Medijan „MEDV“ varijable iznosi 21.2.

Slijedi analiza matrice korelacija. Možemo vidjeti veći broj varijabli s pozitivnim i negativnim korelacijama između nezavisnih varijabli, ali i nezavisnih varijabli s zavisnom.



Slika 127 Korelacijska matrica

Detaljnije će se analizirati korelacija nezavisnih varijabli s zavisnom.

```
print(korelacijska_matrica['MEDV'].drop(['MEDV']).sort_values())
✓ 0.5s
LSTAT     -0.562960
INDUS     -0.463269
TAX       -0.459274
PTRATIO   -0.447464
NOX       -0.411486
CRIM      -0.380072
RAD       -0.379016
AGE       -0.368203
CHAS       0.164782
DIS        0.233469
B          0.317941
ZN         0.339767
RM         0.667695
```

Slika 128 Korelacije između nezavisnih varijabli i zavisne varijable

Može se vidjeti da postoji više varijabli s negativnom korelacijom s varijablom „MEDV“. Najveću pozitivnu korelaciju s zavisnom varijablom ima varijabla „RM“, odnosno prosječni broj soba u stanu. To znači da što je više soba u stanu to je veća vrijednost stana što je naravno i za očekivati. Najveću negativnu korelaciju s zavisnom varijablom ima „LSTAT“, odnosno postotak populacije slabijeg imovinskog statusa u ukupnoj populaciji. To znači što je veći udio populacije slabijeg imovinskog statusa u ukupnoj populaciji to su manje vrijednosti nekretnina u tom predgrađu, što je ponovno i za očekivati.

Što se tiče šuma u podacima analizirane su nedostajuće vrijednosti. Za varijablu„*RM*“ (prosječni broj soba u stanu) postoje 5 instanci sa vrijednostima koje nedostaju.

```
#5 null vrijednosti za RM
dataset.isna().sum()

CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         5
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

Slika 129 Broj nedostajućih vrijednosti po atributima

Kako je to numerička varijabla na mjesto „null“ kod navedenih instanci stavit će se prosječna vrijednost ove varijable.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(dataset[['RM']])
dataset[['RM']] = imputer.transform(dataset[['RM']])
```

Slika 130 Zamjena nedostajućih vrijednosti

Što se tiče outlier vrijednosti nisu uočene instance sa ekstremnim vrijednostima koje ne bi spadale unutar granica dopuštenog. Sljedeći korak jest podjela skupa podataka na skup za treniranje i testiranje. U testni skup spada 25% originalnog skupa podataka. Na kraju je provedena standardizacija vrijednosti skupova.

Na kraju možemo zaključiti da je ovaj skup podataka vrlo malene veličine prema broju instanci. U odnosu na ostale opisane skupove podataka ovaj je nešto manje dimenzije od 13 atributa pretežito numeričkih tipova. Ni ovaj skup podataka nije imao previše šuma, a također smo vidjeli da postoji nekoliko nezavisnih varijabli koje su u znatnoj korelaciji sa zavisnom. Ovaj skup podataka moći će se usporediti sa prethodnim skupom u pogledu točnosti i vremena treniranja zbog istog tipa problema koji se rješava, a to je regresijski problem.

10.6.3. Postavke i treniranje

Struktura koja se pokazala najoptimalnijom kod ovog skupa podataka sastoji se od dva skrivena sloja sa po 64 neurona ponovno sa „ReLU“ aktivacijskom funkcijom. Kako imamo regresijski problem ponovno je na izlaznom sloju jedan neuron sa linearnom aktivacijskom funkcijom. Funkcija gubitka jest prosječna kvadrirana greška.

```
ann_adam = tf.keras.models.Sequential()
#2 skirvena sloja sa 64 neurona
ann_adam.add(tf.keras.layers.Dense(units=64, activation='relu'))
ann_adam.add(tf.keras.layers.Dense(units=64, activation='relu'))
# Dodavanje izlaznog sloja
ann_adam.add(tf.keras.layers.Dense(units=1, activation='linear'))
# kompajliranje mreže
ann_adam.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics = ['mean_squared_error'])
```

Slika 131 Struktura neuronske mreže

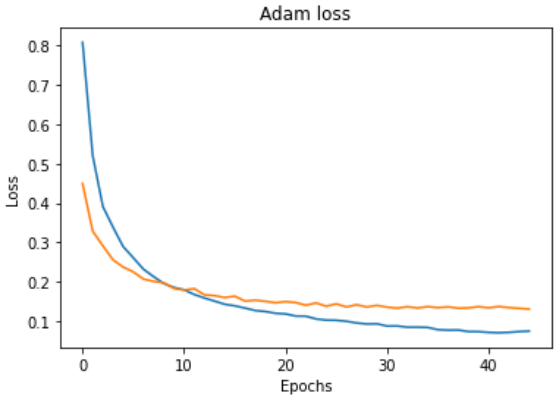
10.6.4. Algoritam Adam

Kao i svaki put do sada najbrži algoritam bio je *Adam*. Nakon 45 epoha i 1.15 sekundi treniranja model treniran ovim algoritmom došao je do vrijednosti koeficijenta determinacije od 0.857 uz prosječnu apsolutnu grešku (MAE) od 2.377 odnosno 2377 dolara. Gledajući prosječnu vrijednost nekretnina od oko 23 000 dolara, mogli bi reći da ovo i nije prevelika greška.

```
Algoritam: Adam
R2 score:0.8569733515245412
Mean Absolute Error:2.3766602881252763
Mean Squared Error:12.012187963548623
Root Mean Squared Error:3.465860349689327
Training time (sec): 1.1540501117706299
```

Slika 132 Rezultati testiranja Adam

Praćen je pad funkcije gubitka te se pokazalo da povećanje broja epoha treniranja od 45 dovodi do „prenaučenosti“.



Slika 133 Točnost i vrijednost funkcije gubitka po epohama Adam

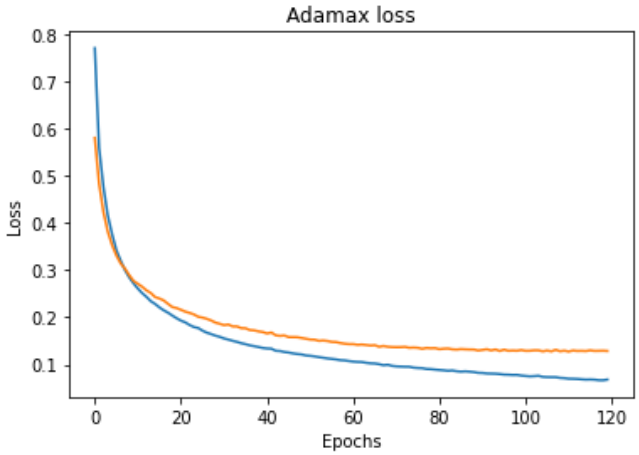
10.6.5. Algoritam Adamax

Neznatno bolje rezultate od *Adam* algoritma pokazao je *Adamax* uz dosta dulje vrijeme treniranja od 120 epoha što je rezultiralo vremenom treniranja od 2.6 sekundi. *Adamaxov* model je u odnosu na *Adamov* imao malo veću prosječnu apsolutnu grešku od 2.382, no nešto manju korjenovanu prosječnu kvadriranu grešku.

```
Algoritam: Adamax
R2 score:0.8595625195871406
Mean Absolute Error:2.381599391251802
Mean Squared Error:11.794734966022098
Root Mean Squared Error:3.4343463666354475
Training time (sec): 2.5518393516540527
```

Slika 134 Rezultati testiranja Adamax

Funkcija gubitka padala je do 120. epoha nakon koje je uočen rast vrijednosti validacijske funkcije gubitka.



Slika 135 Točnost i vrijednost funkcije gubitka po epohama Adagrad

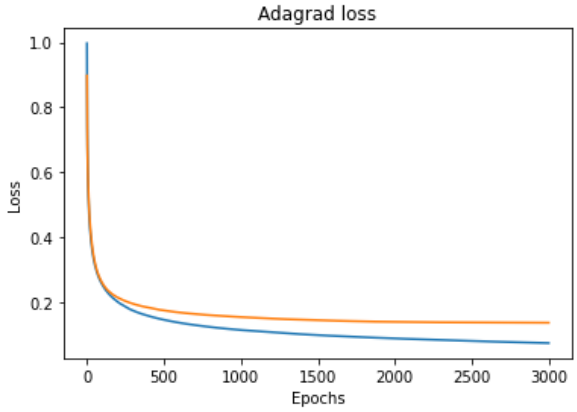
10.6.6. Algoritam Adagrad

Najslabiji se ponovno pokazao *Adagrad* algoritam. Iako se uspjelo doći sličnih točnosti kao i kod ostalih algoritama, vrijeme treniranja bilo je daleko dulje. Treniranje modela Adagrad algoritmom trajalo je čak 3000 epoha, a vrijeme treniranja bilo je preko minute, točnije 66 sekundi. Koeficijent determinacije iznosio je najnižih 0.851, no MAE vrijednost bila je nešto niža od opisanih algoritama od 2.367. S druge strane model kod ovog algoritma imao je najveći RMSE od 3.53.

```
Algoritam: Adagrad
R2 score:0.8517514947715383
Mean Absolute Error:2.367070063203573
Mean Squared Error:12.450749067401645
Root Mean Squared Error:3.5285618979127524
Training time (sec): 66.21861553192139
```

Slika 136 Rezultati testiranja Adagrad

Kao i gotovo uvijek do sad *Adagrad* ima vrlo stabilnu funkciju gubitka bez oscilacija te je njezina vrijednost u svakoj epohi niža ili jednaka nego u prethodnoj epohi. Zadnjih 100 epoha treniranja nije se više smanjivala.



Slika 137 Točnost i vrijednost funkcije gubitka po epohama Adagrad

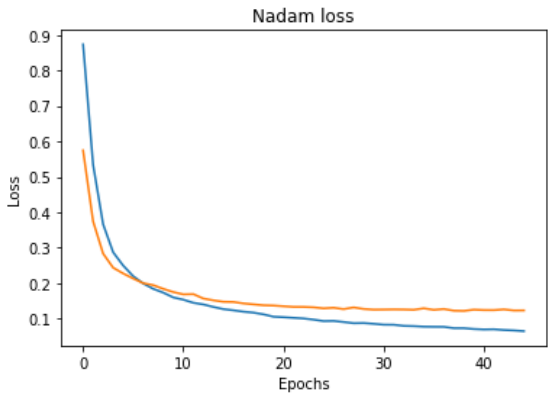
10.6.7. Algoritam NAdam

NAdam, kao i *Adam* najbrži su algoritmi te je i ovim algoritmom model treniran 45 epoha, to jest 1.3 sekunde. U ovom slučaju koeficijent determinacije iznosio je 0.865 uz MAE od 2.367. RMSE iznosi, od sad najnižih, 3.36.

```
Algoritam: Nadam
R2 score:0.8653734304299537
Mean Absolute Error:2.367273465543985
Mean Squared Error:11.306701763627155
Root Mean Squared Error:3.362543942259663
Training time (sec): 1.307190179824829
```

Slika 138 Rezultati testiranja NAdam

Vrijednost validacijske funkcije gubitka padala je do 45 epoha nakon koje je završeno treniranje.



Slika 139 Točnost i vrijednost funkcije gubitka po epohama NAdam

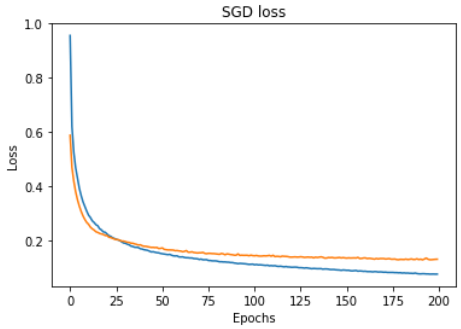
10.6.8. Algoritam SGD

Slične rezultate kao i ostali algoritmi imao je i *SGD* algoritam. Kao i do sad ovaj algoritam ukupno je drugi najsporiji te je model treniran do 200. epohe što je vrijeme oko 4 sekunde. Koeficijent determinacije na razini je ostalih te iznosi 0.859 uz MAE od 2.328.

```
Algoritam: SGD
R2 score:0.8588754452898038
Mean Absolute Error:2.32794943973422
Mean Squared Error:11.85243935672485
Root Mean Squared Error:3.442737189610158
Training time (sec): 3.803467035293579
```

Slika 140 Rezultati testiranja SGD

Validacijska funkcija ponovno je nešto stabilnije te pada do oko 200. epohe.



Slika 141 Točnost i vrijednost funkcije gubitka po epohama SGD

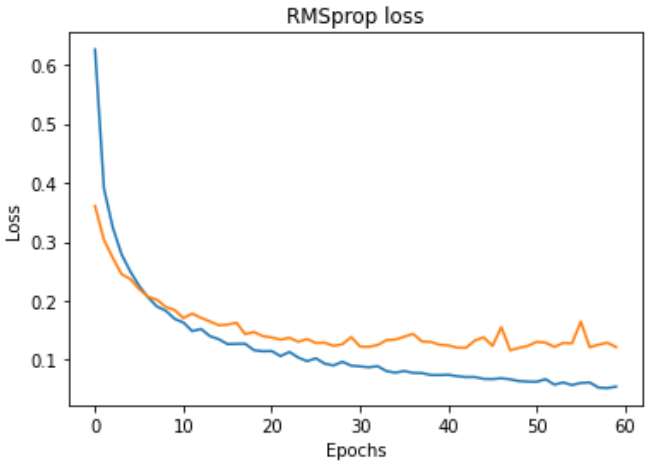
10.6.9. Algoritam RMSProp

Posljednji algoritam, ali i onaj koji je pokazao na ovom skupu podataka najbolje rezultate, jest *RMSProp*. Ovim algoritmom model je treniran 60 epoha što je također dosta niskih 1.5 sekundi. Koeficijent determinacije nešto je viši nego kod *NAdam* algoritma te iznosi 0.867, no u odnosu na ostale algoritme koji su imale vrijednost MAE oko 2.3, ovdje ona iznosi 2.15, to jest model je napravio prosječno grešku manju za 200 dolara. Također ovaj model radi i najmanje većih grešaka pošto RMSE iznosi također najmanjih 3.34.

```
Algoritam: RMSprop
R2 score:0.8670465423543414
Mean Absolute Error:2.150323849916458
Mean Squared Error:11.166184348627748
Root Mean Squared Error:3.3415841076692576
Training time (sec): 1.5003647804260254
```

Slika 142 Rezultati testiranja RMSProp

Ovaj algoritam imao je nešto nestabilniju funkciju gubitka gdje vidimo nešto veće oscilacije, a najoptimalnije vrijeme treniranja iznosilo je 60 epoha.



Slika 143 Točnost i vrijednost funkcije gubitka po epohama RMSProp

10.6.10. Zaključak na performanse algoritama

Algoritam	R2 score	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	Training time (sec)	Test time (sec)	Epochs
Adam	0.856973	2.376660	12.012188	3.465860	1.154050	0.025022	45
Adamax	0.859563	2.381599	11.794735	3.434346	2.551839	0.021019	120
Adagrad	0.851751	2.367070	12.450749	3.528562	66.218616	0.026023	3000
Nadam	0.865373	2.367273	11.306702	3.362544	1.307190	0.019017	45
SGD	0.858875	2.327949	11.852439	3.442737	3.803467	0.019017	200
RMSprop	0.867047	2.150324	11.166184	3.341584	1.500365	0.022018	60

Slika 144 Ukupne performanse algoritama

Kako je u ovom slučaju skup podataka bio izrazito malen, vrijeme treniranja nije predstavljalo veliki problem kod nijednog algoritma osim Adagrad-a. Svi algoritmi završili su treniranje unutar 4 sekundi, osim Adagrad-a kojem je trebalo 3000 epoha da vrijednost validacijske funkcije gubitka počne stagnirati. U ovom slučaju svi algoritmi imali su vrlo slične vrijednosti metrika. Kod ovako malenih skupova podataka svakako bi bilo poželjno isprobati sve algoritme te vidjeti koji daje najbolje točnosti pošto vrijeme treniranja, u ovom slučaju, nije presudno.

11. Zaključak

Kroz ovaj rad opisana je teorijska podloga neuronskih mreža, no naglasak je bio na ispitivanju performansi njihovih optimizacijskih algoritama. Korišteni skupovi podataka bili su različitih karakteristika uključujući različite veličine skupova, problema koji se rješavaju, šuma u podacima te tipova podataka. Ono što se moglo uočiti jesu vrlo konzistentne performanse svih algoritama, odnosno, svi algoritmi su se kod različitih tipova skupova podataka ponašali veoma slično, od vremena treniranja pa do točnosti u predikciji.

Mogli smo uočiti da svi algoritmi imaju sposobnosti pronalaska minimuma funkcije gubitka kod svih skupova, no najveća razlika jest u vremenu istog. Što se tiče ukupne točnosti kod gotovo svakog skupa podataka mogli smo vidjeti da se ona razlikuje oko 1% između algoritama. To bi značilo da svi algoritmi dolaze do minimuma funkcije gubitka, no čini se da zbog oscilacija blizu minimuma nemaju svi algoritmi identične točnosti.

Algoritam koji se najviše razlikovao od ostalih svakako je „*Adagrad*“. U teorijskoj obradi mogli smo uočiti da je ovo najjednostavnija modifikacija osnovnog algoritma gradijentnog spusta. Za očekivati je bilo da bi ovaj algoritam trebao imati nešto bolje performanse od osnovnog algoritma gradijentnog spusta zbog varijabilne stope učenja, no to nije bio slučaj. Zbog prevelike varijabilnosti, odnosno smanjenja stope učenja kroz epohe, algoritam prebrzo snizi stopu učenja do razine da već nakon prvih nekoliko epoha ona postaje premala što dovodi do toga da na višim epohama učenje presporo napreduje zbog čega su sama vremena učenja jako duga. To je osobito problematično kod kompleksnih skupova podataka gdje je vrijeme učenja i kod bržih algoritma nešto dulje. U tim slučajevima ovaj algoritam jednostavno ne može u razumnom vremenu postići svoj maksimum. Upravo taj problem vidjeli smo kod detekcije mačaka i pasa u konvolucijskoj neuronskoj mreži i predviđanja cijene nekretnina u okružju King gdje je treniranje zaustavljeno zbog ekstremno dugog vremena treniranja. Ovaj algoritam mogli bi preporučiti samo kod nekompleksnih i vrlo malih skupova podataka, odnosno kod skupova podataka gdje se vrlo brzo dolazi do minimuma funkcije gubitka gdje se može iskoristiti prednost ovoj algoritma, a to je vrlo stabilna funkcija gubitka koja ne oscilira kao što je slučaj kod ostalih algoritama te vrlo niska stopa učenja pri minimumu funkcije gubitka sprečava „prenaučenost“.

Osnovni algoritam gradijentnog spusta (kroz ispitivanja „*Mini-Batch Gradient Descent*“ verzija) pokazao se i više nego dobar. Uz bržu konvergenciju prema minimumu funkcije gubitka pokazao je vrlo visoke točnosti vrlo često na razini ostalih algoritama, a također zbog često stabilnije funkcije gubitka, u nekoliko slučajeva imao je malo veće točnosti zbog boljeg „pogađanja“ minimuma funkcije gubitka. U odnosu na preostale algoritme dosta je sporiji pošto

je stopa učenja nevarijabilna. Zbog toga, kod kompleksnijih skupova podataka ima presporo učenje, što se najviše pokazalo kod detekcije mačaka i pasa gdje je, kao i kod slučaja „*Adagrad*“ algoritma treniranje prekinuto zbog prespore konvergencije prema minimumu. U slučaju vrlo jednostavnog skupa podataka, kao što je predviđanje raka dojke kod žena, algoritam je imao svakim ponavljanjem treniranja najviše točnosti gdje je puno bolje konvergirao prema minimumu funkcije gubitka od ostalih algoritama. Zbog toga ovaj algoritam također bi preporučili kod manjih i jednostavnijih skupova podataka ili kada nam vrijeme treniranja nije bitno.

Preostali algoritmi su algoritmi sa varijabilnom funkcijom gubitka koja kod početka treniranja ubrzava treniranje te vrlo brzo dolaze do minimuma. U svim slučajevima ukupne točnosti ovih algoritama razlikovale su se unutar 1%. Algoritmi „*RMSProp“* i „*Adamax*“ u većini slučajeva imali su slične točnosti uz nešto dulje vrijeme treniranja od najbržih algoritama „*Adam*“ i „*NAdam*“ s tim da je gotovo uvijek „*RMSProp*“ bio brži od „*Adamax*“ algoritma. Upravo brzina konvergencije prema minimumu velika je prednost ovih algoritama pred „*SGD*“ i „*Adagrad*“ gdje kod kompleksnijih problema i velikih skupova podataka svakako imaju prednosti u korištenju. Iako se „*Adamax*“ smatra nadogradnjom na „*Adam“* algoritam, to se nije vidjelo u ovom ispitivanju. Što se tiče vremena treniranja „*Adam*“ je zajedno sa „*NAdam*“ algoritmom bio najbrži. Oba algoritma su gotovo uvijek nakon istog broja epoha bila blizu minimuma funkcije gubitka zbog čega ne bi mogli izdvojiti nijedan od navedenih kao bolji, ali svakako bi ih zajedno mogli izdvojiti kao najbolje u vidu brzine konvergiranja prema minimumu zbog čega je preporuka korištenje ovih algoritama kod bilo kojeg skupa podataka.

Nedostatak koji je uočen kod svih ovih četiri brzih algoritama jest problem sa potencijalnom „prenaučenosti“. Zbog izrazito brzog dolaska do minimuma funkcije gubitka, vrlo lako se može dogoditi da vrijednost funkcije gubitka nad validacijskim skupom počinje rasti nakon određenog broja epoha, često vrlo naglo upravo zbog njihove velike brzine treniranja. Preporuka je zbog toga praćenje grafa funkcije gubitka za trening i testni skup te potencijalno smanjivanje i povećanje broja epoha treniranja. Iako „*Tensorflow/Keras*“ ima mogućnosti uključenja takozvane „EarlyStopping“ callback funkcije koja automatski zaustavlja treniranje ako nekoliko epoha za redom vrijednost validacijske funkcije gubitka raste, zna se dogoditi da se ne detektira rast kod oscilacija, što je bio slučaj u ovom ispitivanju, gdje funkcija „prosječno“ raste. Kod jednostavnih skupova možemo isprobati „*Adagrad*“ ili „*SGD*“ algoritme kod kojih, zbog duljeg treniranja, znatno teže dolazi do prenaučenosti .

Konačni zaključak ovog istraživanja bio bi da su algoritmi „*Adam“* i „*NAdam“* neizostavni kod gradnje neuronskih mreža u razvojnom okviru „*Tensorflow*“ kod svih skupova podataka, no isprobavanje ostalih algoritama u vidu potencijalno boljih točnosti također nije naodmet.

Neuronske mreže jako se razlikuju od ostalih algoritama strojnog učenja zbog velikog broja podesivih parametara kao što je struktura mreže, tipovi aktivacijskih funkcija u skrivenim slojevima, funkcija gubitka, veličina uzorka za propagaciju te naravno optimizacijskih algoritama. Podešavanje navedenih parametara kako bi maksimizirale performanse izrađenih modela može biti izrazito naporan posao. Kako bi malo olakšali taj posao preporučuje se u svakom slučaju korištenje algoritama „*Adam*“ i „*NAdam“*. Osobno smatram da bi se daljnja istraživanja trebala baviti analizom utjecaja različitih struktura neuronskih mreža (u vidu broja skrivenih slojeva i neurona u slojevima) na performanse modela koristeći skupove podataka različitih karakteristika.

Popis literature

Agarwal, R. (Towards D. S. (2019). *The 5 Classification Evaluation metrics every Data Scientist must know*. https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226

Althnian, A., AlSaeed, D., Al-Baity, H., Samha, A., Dris, A. Bin, Alzakari, N., Abou Elwafa, A., & Kurdi, H. (2021). Impact of dataset size on classification performance: An empirical evaluation in the medical domain. *Applied Sciences (Switzerland)*, *11*(2), 1–18. https://doi.org/10.3390/app11020796

Baheti, P. (V7 L. (2022). *12 Types of Neural Networks Activation Functions: How to Choose?* https://www.v7labs.com/blog/neural-networks-activation-functions

Brownlee, J. (Machine L. M. (2018). *Train Neural Networks With Noise to Reduce Overfitting*. https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/

Brownlee, J. (Machine L. M. (2021a). *Code Adam Optimization Algorithm From Scratch*. https://machinelearningmastery.com/adam-optimization-from-scratch/

Brownlee, J. (Machine L. M. (2021b). *Gradient Descent With AdaGrad From Scratch*. https://machinelearningmastery.com/gradient-descent-with-adagrad-from-scratch/

Brownlee, J. (Machine L. M. (2021c). *Gradient Descent With Momentum from Scratch*. https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch/

Brownlee, J. (Machine L. M. (2021d). *Gradient Descent With RMSProp from Scratch*. https://machinelearningmastery.com/gradient-descent-with-rmsprop-from-scratch/

Brownlee, J. (Machine L. M. (2021e). *How to Choose an Activation Function for Deep Learning*. https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

Brownlee, J. (Machine L. M. (2021f). *Regression Metrics for Machine Learning*. https://machinelearningmastery.com/regression-metrics-for-machine-learning/

Chen, B. (Towards D. S. (2021). *7 popular activation functions you should know in Deep Learning and how to use them with Keras and TensorFlow 2*. https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6

Cheng, C. (Towards D. S. (2022). *Principal Component Analysis (PCA) Explained Visually with Zero Math*. https://towardsdatascience.com/principal-component-analysis-pca-explained-visually-with-zero-math-1cbf392b9e7d

Cornelisse, D. (freeCodeCamp. org. (2018). *An intuitive guide to Convolutional Neural Networks*. https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/#:~:text=The term convolution refers to,then produce a feature map.

Dobša, J. (2021). Regresijska analiza. In *FOI SMI Moodle materijali 2020./2021.* https://elfarchive2021.foi.hr/mod/resource/view.php?id=9570

Dozat, T. (2016). *Incorporating Nesterov Momentum into Adam*. https://openreview.net/pdf/OM0jvwB8jIp57ZJjtNEZ.pdf

Dr. Woodruff, A. (The U. O. Q. (n.d.). *What is a neuron?* https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron

Gebel, Ł. (Towards D. S. (2020). *Why We Need Bias in Neural Networks*. https://towardsdatascience.com/why-we-need-bias-in-neural-networks-db8f7e07cb98

Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow. In *O'Reilly Media*.

Hill, T. (Towards D. S. (2018). *Part 2: Gradient descent and backpropagation*. https://towardsdatascience.com/part-2-gradient-descent-and-backpropagation-bf90932c066a

Kavlakoglu, E. (IBM). (2020). *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's*

the Difference? https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks

Kwon, O., & Sim, J. M. (2013). Effects of data set features on the performances of classification algorithms. *Expert Systems with Applications*, *40*(5), 1847–1857. https://doi.org/10.1016/j.eswa.2012.09.017

Mulla, M. Z. (medium. com. (2020). *Cost, Activation, Loss Function|| Neural Network|| Deep Learning. What are these?* https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de

Ng, R. (n.d.). *Evaluating a Classification Model*. https://www.ritchieng.com/machine-learning-evaluate-classification-model/

Rivolli, A. (Universidade T. F. do P., Garcia, L. P. F., Soares, C., Vanschoren, J., & de Carvalho, A. C. P. L. F. (2022). *Meta-features for meta-learning*. 1–51.

Ronaghan, S. (Towards D. S. (2018). *Deep Learning: Which Loss and Activation Functions should I use?* https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8

Ruder, S. (ruder. io. (2016). *An overview of gradient descent optimization algorithms*. https://ruder.io/optimizing-gradient-descent/index.html

Russell, S., & Norvig, P. (2010). Artificial Intelligence A Modern Approach Third Edition. In *2010 The 2nd International Conference on Computer and Automation Engineering, ICCAE 2010* (Vol. 4). https://doi.org/10.1109/ICCAE.2010.5451578

Techopedia. (2018). *Hidden Layer*. https://www.techopedia.com/definition/33264/hidden-layer-neural-networks

Wang, C.-F. (Towards D. S. (2019). *The Vanishing Gradient Problem*. https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

Popis slika

Popis tablica

Prilozi

Modificirani skupovi podataka: https://drive.google.com/file/d/1IEM2Ps_SL0IMUzaCDerI-JxHuuq6_oeK/view?usp=sharing

GitHub repozitorij s PyMFE analizom i Jupyter Notebookovima u kojima se odvija proces izgradnje modela: https://github.com/bpilosta/diplomski-rad

Breast Cancer skup podataka: https://archive-beta.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+diagnostic

Credit Card Fraud skup podataka: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Rain Tomorrow skup podataka: https://www.kaggle.com/jsphyg/weather-dataset-rattle-package

Cats&Dogs skup podataka: https://www.microsoft.com/en-us/download/details.aspx?id=54765

King County house sales skup podataka: https://www.kaggle.com/datasets/harlfoxem/housesalesprediction

Boston Real Estate skup podataka: https://www.kaggle.com/datasets/arslanali4343/real-estate-dataset