Otkrivanje neželjene pošte pomoću neuronskih mreža

Varga, Luka

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Diplomski studij

OTKRIVANJE NEŽELJENE POŠTE POMOĆU NEURONSKIH MREŽA

Diplomski rad

Luka Varga

Osijek, 2022.

SADRŽAJ

1. UVOD

U današnje vrijeme, velik dio naših života odvija se online. Podaci o bankovnim karticama, fotografije na društvenim mrežama te razni drugi osobni podaci koje dajemo raznim tvrtkama. Spam mailovi šalju se ogromnom broju osoba, a takvi mailovi ponekad imaju i zlokobne namjere. Statistika za prošlu godinu govori kako je udio neželjene elektroničke pošte u odnosu na ukupni promet čak 84%. Prosječni radnik, koji na dnevnoj bazi koristi službeni račun za razmjenjivanje komunikacije putem elektroničke pošte u prosjeku dobije preko 100 (riječima: stotinu) (točnije 144.7) pisama, odnosno *mailova* [1]. Ako se uzme u obzir da je od spomenutih 144 mailova njih čak 122 neželjeno, može se doći do zaključka da se radi o ozbiljnom uznemiravanju korisnika. Velike korporacije poput Googlea, Yahooa i ostalih, taj problem su riješile korištenjem umjetne inteligencije s velikim postotkom uspješnosti. Oni koriste, kako sami kažu: „masivnu bazu podataka zlonamjernih veza“ [2] kako bi filtrirali mailove, jer uz sami sadržaj maila, obraćaju pozornost i na IP adresu, domenu pošiljatelja, koje protokole koristi za slanje, te autentifikaciju računa korisnika. Osim toga, uspješno su razvili sustav koji provjerava povratnu informaciju (engl. *feedback*) o svakom korisniku te se na taj način prepoznaju korisnici po reputaciji, koja ih zatim povezuje s njihovim osobnim računom. To se naziva *whitelist*. To su već unaprijed osmišljeni algoritmi s nazivom nekih od poznatih imena u tom svijetu poput Bayesa, Portera i ostalih. U nastavku ovog diplomskog rada opisano je kako taj proces izgleda u nekomercijalnoj primjeni koristeći ručno dizajnirani model neuronske mreže. Diplomski je rad strukturiran tako da je prvo opisana teorijska podloga rješenja. Zatim slijedi detaljan opis programskog okruženja koje je korišteno. Definirane su i vrste, modeli te optimizacija same mreže, a u konačnici su prikazani rezultati testiranja koji predočavaju raznovrsne dosege određenih pristupa problemu s kojim se susrećemo.

1.1. Zadatak diplomskog rada

Zadatak diplomskog rada je napraviti model neuronske mreže koja će biti u mogućnosti prepoznati elektroničku poštu koja je poznata kao *spam* odnosno neželjena pošta.

2. TEORIJSKA PODLOGA

Kako bi ovaj zadatak bio što bolje riješen, korištene su najnovije tehnologije koje su u posljednje vrijeme jako napredovale, te i dalje konstantno napreduju u dijelu raspoznavanja obrazaca i uzoraka [3]. Radi se o području neuronskih mreža i umjetne inteligencije, gdje mreža uči na predanim primjerima i testira svoje znanje na setu testnih podataka.

2.1. Umjetne neuronske mreže

Ideja iza umjetnih neuronskih mreža bazira se na ljudskom živčanom sustavu u kojem za donošenje nekih od najjednostavnijih pa do najkompleksnijih odluka, sudjeluje nekoliko milijuna neurona odnosno živčanih stanica. Ljudski mozak pokazuje prisutnost ogromnih neuronskih mreža sposobnih za obavljanje kognitivnih, perceptivnih i kontrolnih zadataka u kojima su ljudi izvrsni. Mozak može obavljati računalno zahtjevne perceptivne radnje (kao što su prepoznavanje lica i govor), kao i kontrolne zadatke (primjerice, pokreti tijela i tjelesne funkcije).

Prednost mozga je njegova učinkovita upotreba ogromnog paralelizma, visoko paralelne računalne arhitekture i nesavršene sposobnosti obrade informacija. Ljudski mozak se sastoji od preko 10 milijardi neurona koji su svi međusobno povezani. Svaki neuron je stanica koja prima, obrađuje i prenosi informacije putem biokemijskih aktivnosti. Dendriti su protoplazmatski razgranati produžeci neurona, koji povezuju tijelo stanice, ili somu, s jezgrom stanice. Jedno dugo vlakno zvano akson proteže se od tijela stanice, na kraju se grana u niti te se povezuje sa susjednim neuronima preko sinaptičkih terminala ili sinapsi.

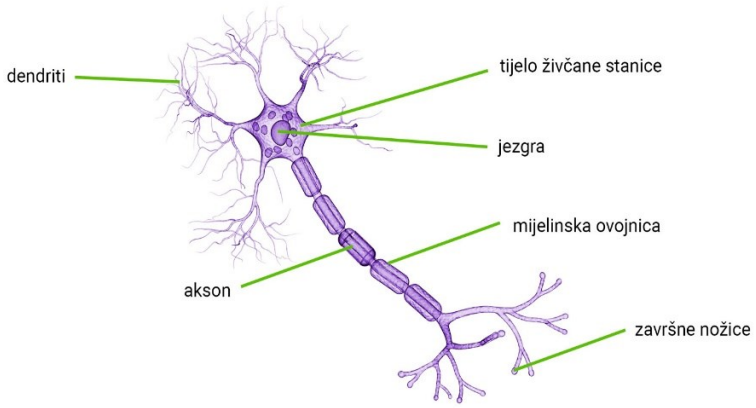
Prijenos poruka s jednog neurona na drugi u sinapsama složen je kemijski proces koji uključuje oslobađanje specifičnih odašiljačkih kemikalija s kraja koji šalje spoj. Rezultat je promjena električnog potencijala unutar tijela prijemne stanice. Puls se isporučuje niz akson ako napon dosegne prag, a stanica se „ispaljuje“.

2.2. Dijelovi neuronske mreže

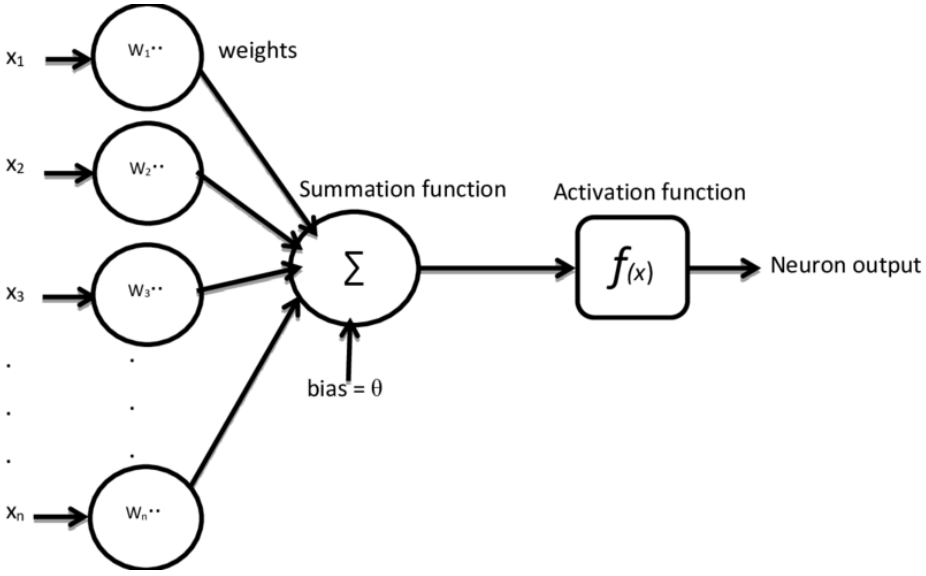
Neuronska mreža sastoji se od osnovnih jedinica neurona koji su podijeljeni u slojeve. Ovisno o načinu kreiranja modela i vrsti korištene mreže ti neuroni su povezani na određeni način.
Određeni broj slojeva sastavljenih od određenog broja neurona tvori neuronsku mrežu. Umjetne

neuronske mreže (engl. *Artificial Neural Networks* – ANN) su ustvari generalizirani matematički modeli bioloških živčanih sustava. Nakon McCullocha i Pittsovog uvođenja pojednostavljenih neurona [4], pojavio se prvi val interesa za neuronske mreže (također poznate kao konekcionistički modeli ili paralelna distribuirana obrada) (1943.).

Umjetni neuroni, često poznati kao neuroni ili čvorovi, osnovni su dijelovi za obradu neuronskih mreža. Učinci sinapsi predstavljeni su težinama veze koje utječu na učinak povezanih ulaznih signala u pojednostavljenom matematičkom modelu neurona, a nelinearna karakteristika koju pokazuju neuroni predstavljena je prijenosnom funkcijom. Kako izgleda ljudska živčana stanica, a kako njen umjetni prikaz, prikazano je u nastavku:



Slika 2.1. Ljudska živčana stanica [5]

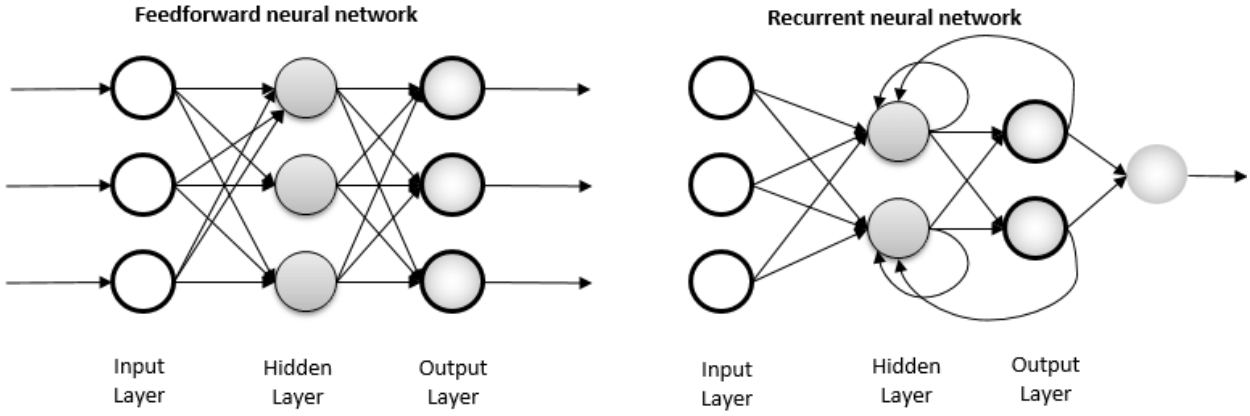


Slika 2.2. Struktura umjetne žičane stanice [6]

2.3. Vrste neuronskih mreža

Postoji nekoliko vrsta neuronskih mreža, od jednoslojnih do višeslojnih, onih s jednim skrivenim slojem do više skrivenih slojeva, povratne ili one koje idu prema naprijed.
U osnovnoj arhitekturi postoje tri vrste neuronskih slojeva: ulazni, skriveni i izlazni slojevi. Tok signala u mrežama s prijenosom prema naprijed je striktno usmjeren prema naprijed, od ulaznih do izlaznih jedinica. Obrada podataka može obuhvatiti mnogo (slojeva) jedinica, ali nema povratnih veza [7]**.**

Veze za povratne informacije vide se u ponavljajućim mrežama. Za razliku od mreža s prijenosom prema naprijed, ovdje su dinamičke značajke mreže ključne. U nekim okolnostima, aktivacijske vrijednosti jedinica prolaze kroz proces opuštanja, što rezultira razvojem mreže u stabilno stanje u kojem se aktivacije ne razlikuju. U nekim aplikacijama, promjene u aktivacijskim vrijednostima izlaznih neurona dovoljno su značajne da je izlaz mreže određen njezinim dinamičkim ponašanjem.



Slika 2.3. Arhitektura i razlike mreža [8]

Postoji nekoliko drugih arhitektura neuronskih mreža kao što su simetrično povezane mreže, perceptroni, konvolucijske neuronske mreže, mreža dugotrajne/kratkoročne memorije itd. U ovom radu najbitnije su upravo povratne mreže te mreža dugotrajne/kratkoročne memorije jer se njih koristilo kako bi se napravio finalni model mreže, a o kojima će se kasnije detaljnije govoriti.

Neuronska mreža mora biti postavljena na takav način da se, kada se primijeni skup ulaza, proizvodi željeni skup izlaza. Postoji nekoliko metoda za određivanje čvrstoće spojeva. Jedna metoda je korištenje a priori informacija za eksplicitno postavljanje težina. Druga metoda je unos obrazaca podučavanja neuronske mreže i da modificiraju svoje težine prema pravilu učenja.

2.4. Treniranje mreže

Postoje tri različite vrste situacija učenja u neuronskim mrežama a to su učenje pod nadzorom, učenje bez nadzora i učenje s pojačanjem. U nadziranom učenju, ulazni vektor i skup očekivanih odgovora – po jedan za svaki čvor – daju se na ulaznim odnosno izlaznim slojevima. Tada je izvršen prolaz prema naprijed, a pogreške ili razlike između željenog i stvarnog odgovora za svaki čvor, otkrivaju se u izlaznom sloju. Oni se zatim koriste za izračunavanje promjena neto težine na temelju trenutnog pravila učenja. Riječ nadzirani odnosi se na činjenicu da vanjski učitelj daje željene signale na određenim izlaznim čvorovima. Algoritam povratnog širenja (engl. *back-propagation*), delta pravilo i pravilo perceptrona najpoznatiji su primjeri ove tehnike [7].

U nenadziranom učenju, jedinica (izlazna) je učena da odgovori na skupove uzoraka unutar ulaza. Sustav bi trebao otkriti statistički značajne aspekte ulazne populacije u ovoj paradigmi. Za razliku od učenja pod nadzorom, ne postoji unaprijed određen skup kategorija u koje bi se uzorci trebali razvrstati; umjesto toga, sustav mora stvoriti vlastitu reprezentaciju dolaznih podražaja.

Učenje s pojačanjem je proces učenja što treba izvesti kako bi se maksimizirao numerički signal nagrade. Za razliku od većine oblika strojnog učenja, učenik se ne podučava koje radnje treba učiniti, umjesto toga, učenik mora odrediti koje aktivnosti proizvode najveću nagradu pokušavajući ih. Kako postići što bolji rezultat ili kako definirati arhitekturu mreže, učenik bi trebao znati poučen vlastitim iskustvom.

Prema nekim modifikacijskim pravilima, težine veza između jedinica prilagođavaju se u gore navedenim paradigmama učenja. Doprinos Hebba (1949), možda najpoznatijeg djela u „povijesti konekcionizma“ [9], uspostavio je teoriju ponašanja utemeljenu, koliko je to moguće, na fiziologiji živčanog sustava.

Hebbov formalni opis (poznat kao Hebbov postulat) o tome kako se učenje može dogoditi bio je najvažniji uvid koji dolazi iz njegovog rada. Promjena sinaptičkih veza između neurona bila je osnova za učenje. Kada je akson stanice A dovoljno blizu da potakne stanicu B i sudjeluje u njenom paljenju opetovano ili dosljedno, neki razvojni proces ili metabolička promjena događa se u jednoj ili obje stanice, povećavajući učinkovitost A kao jedne od stanica koje pokreće B. Načela na kojima se temelje ove izjave postale su poznate kao Hebbianovo učenje (engl. *Hebbian*

Learning). Većina strategija učenja neuronskih mreža može se smatrati varijacijom Hebbianovog pravila učenja. Osnovna premisa je da kada su dva neurona angažirana u isto vrijeme, njihove veze moraju biti ojačane. Jedan od povezanih neurona u jednoslojnoj mreži bit će ulazna jedinica, dok će drugi biti izlazna jedinica. Kada su podaci predstavljeni u bipolarnom obliku, željeno ažuriranje težine može se lako izraziti kao:

$$\omega_i(novi) = \omega_i(stari) + x_i o \tag{2-1}$$

gdje je o željeni izlaz za:

$$i = 1\ do\ n(ulaza) \tag{2-2}$$

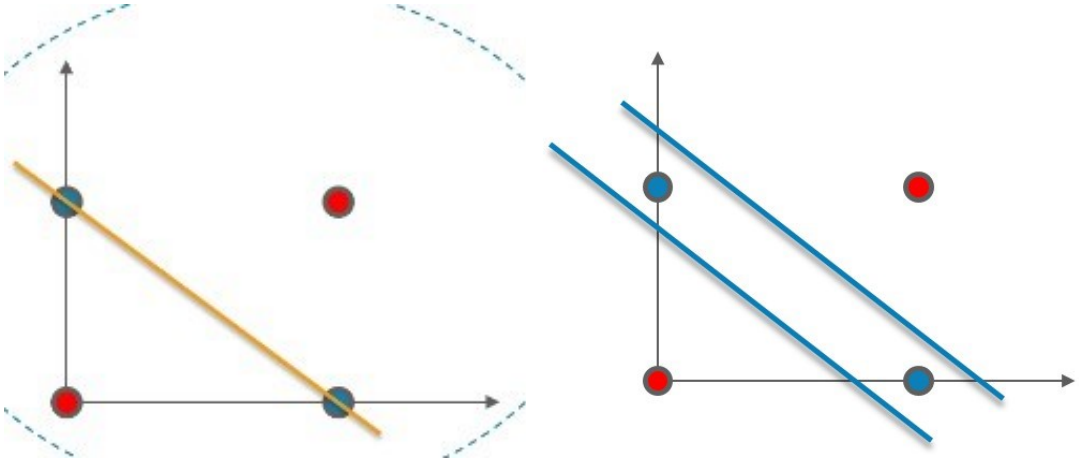
Nažalost, osim ako ulazni podaci nisu prikladno standardizirani, obično Hebbianovo učenje kontinuirano povećava svoje težine bez ograničenja. Sljedeća tehnika treninga koja se koristi zove se pravilo učenja perceptrona. Perceptron je jednoslojna neuronska mreža s težinama i sklonostima (engl. *bias*) koja se može uvježbati za izlaz ispravnog ciljnog vektora kada mu se pridoda odgovarajući ulazni vektor i najprikladniji je za jednostavne probleme kategorizacije uzoraka. Mreža započinje s nasumično odabranim vrijednostima težina koje, tijekom svakog lošeg rezultata, odnosno ako stvarni izlaz ne odgovara pretpostavljenom, prepravlja u druge vrijednosti po formuli:

$$\Delta\omega_{i\,j}(n) = -\eta * \frac{\delta E}{\delta \omega_{ij}} + \alpha * \Delta\omega_{ij}(n-1) \tag{2-3}$$

gdje su η i α brzina učenja i moment.

Proces je prilično sličan Hebbovom pravilu, no jedina razlika je u tome što se težine veze ne mijenjaju kada mreža uspješno reagira. Osim jednoslojnih perceptrona, jedna od najpopularnijih struktura umjetnih neuronskih mreža je višeslojni perceptron. Njegova se struktura sastoji od barem jednog skrivenog sloja neurona između ulaznog i izlaznog sloja gdje je izlaz jednog skrivenog sloja ulaz sljedećeg izlaznog sloja ili ulaz sljedećeg skrivenog sloja. Broj slojeva u neuronskoj mreži određuje sposobnost mreže da nauči specifične obrasce [10]. S više slojeva neuronske mreže ne samo da postaju složenije nego i zahtijevaju dodatne resurse. Neuronska mreža s jednim aktivnim slojem može naučiti samo kako riješiti linearno odvojive probleme. S

dva aktivna sloja, međutim, neuronska mreža može oblikovati konveksne regije u podatkovnom prostoru, što znači da neuronska mreža može odvojiti uzorke podataka s više linija koje tvore različite oblike (poput pravokutnika, kvadrata, trokuta itd.) (Slika 2.4.). Neuronska mreža s tri aktivna sloja može stvoriti bilo koji proizvoljan oblik za odvajanje ulaznih podataka, što znači da bi višeslojni perceptron trebao biti u stanju riješiti bilo koji problem.



Slika 2.4. Razlika u klasifikacijskim mogućnostima između jednoslojnog (lijevo) i višeslojnog (desno) perceptrona [11]

2.5. Algoritam povratnog širenja

Za razliku od jednostavnog perceptrona, koji je samo u stanju rješavati linearno odvojive ili linearno neovisne probleme, uzimanjem djelomične derivacije pogreške mreže s obzirom na svaku težinu, naučit ćemo ponešto o smjeru u kojem se greška mreže kreće. Doista, ako od težine oduzmemo negativ ove derivacije (tj. stopu promjene pogreške kako vrijednost težine raste), pogreška će se smanjivati sve dok ne pronađe lokalni minimum. To je logično jer pozitivna derivacija pokazuje da se pogreška povećava kako se težina povećava. Ako je derivacija negativna, logično je dodati negativan broj težini i obrnuto. Ovaj algoritam je poznat kao algoritam povratnog širenja jer uzima djelomične derivacije i primjenjuje ih na svaku od težina, počevši od težine izlaznog sloja do težine skrivenog sloja, zatim od skrivenog sloja do težine ulaznog sloja (kako se ispostavilo, to je neophodno jer promjena ovog skupa pondera zahtijeva poznate parcijalne

derivacije izračunate u sloju nizvodno).

Neuronska mreža se može trenirati na jedan od dva načina: *online* ili grupno [7]. Za isti broj prezentacija podataka, dva pristupa imaju drastično različit broj ažuriranja težine. Za svaki uzorak ulaznih podataka izračunavaju se ažuriranja težine s *online* metodom, a težine se prilagođavaju nakon svakog uzorka. Druga mogućnost je izračunati ažuriranje težine za svaki ulazni uzorak, ali rezultate pohraniti tijekom jednog pokretanja skupa za obuku, koji se naziva epoha. Svi doprinosi se dodaju na kraju epohe, a zatim se težine ažuriraju složenom vrijednošću. Ova metoda koristi kumulativno ažuriranje težine za prilagodbu težina, dopuštajući joj da bliže prati gradijent, a metoda se zove serijski trening (engl. *batch-training*). Ukratko, trening je unošenje uzoraka za trening u neuronsku mrežu kao ulazne vektore, određivanje pogreške izlaznog sloja, a zatim modificiranje težine mreže kako bi se pogreška svela na najmanju moguću mjeru.

Spuštanje u serijskom treningu, odnosno *batch* metodi, temelji se na gradijentu koji ovisi o parametrima brzine učenja i zamaha (momenta). Utjecaj prijašnjih promjena težine na trenutni smjer kretanja u prostoru težine određen je pomoću zamaha. Za uspjeh treninga i brzinu učenja neuronske mreže neophodan je solidan izbor i jednog i drugog. Pokazalo se da učenje unatrag s dovoljno skrivenih slojeva procjenjuje bilo koju nelinearnu funkciju proizvoljnom preciznošću. Neuronske mreže za učenje unatrag su zbog toga snažan izbor za predviđanje signala i modeliranje sustava.

2.6 Povratna neuronska mreža

Neuronska mreža mora sadržavati memoriju kako bi obrađivala privremene informacije. Postoje dva osnovna načina za ugradnju memorije u neuronske mreže. Prvi je uvesti vremenske odgode u mrežu i prilagoditi njihove parametre tijekom faze učenja. Drugi način je uvođenje pozitivnih povratnih informacija, što znači da se mreža ponavlja. U nastavku su opisani principi obje arhitekture, konačni impulsni odziv (engl. *finite impulse response* - FIR) i povratne neuronske mreže (engl. *Recurrent neural network* – RNN) [12], dok će se ovaj rad detaljnije usredotočiti na drugu arhitekturu: rekurentne, odnosno povratne neuronske mreže.

Funkcionalni ekvivalent neuronske mreže s vremenskom odgodom (engl. *time-delay neural network* - TDNN), FIR neuronska mreža koristi statičku metodu odvijanja u vremenu. Oni nemaju povratne veze između jedinica. TDNN pruža jednostavne oblike dinamike pohranjivanjem

zaostalih ulaznih varijabli na ulaznom sloju i/ili zaostalih izlaza skrivenih jedinica na skrivenom sloju. FIR mreže su mreže s FIR linearnim filtrom koji se može opisati korištenjem prisluškivanih linija odgode umjesto statičkih težina veze između jedinica. Prerastanjem mreže u značajnu ekvivalentnu statičku strukturu nakon upotrebe tehnike odvijanja u vremenu na FIR-u, sva kašnjenja će biti eliminirana. Zatim se za obuku koristi uobičajeni algoritam povratnog širenja. Budući da su vremenska kašnjenja i vremenski okviri formalni ekvivalenti, mogu se smatrati autoregresivnim modelima.

Rekurentna mreža je neuronska mreža s povratnom spregom (veze zatvorene petlje). Primjeri uključuju BAM (engl. *Bidirectional associative memory*), Hopfield, Boltzmannov stroj i mreže povratnog širenja. Arhitektura uključuje višeslojne mreže za prijenos podataka s odvojenim ulaznim i izlaznim slojevima, kao i potpuno i djelomično međusobno povezane mreže. U potpuno povezanim mrežama, svaki čvor prima ulaz od svakog drugog čvora, stoga ne postoje različiti ulazni slojevi čvorova. Povratna informacija samom čvoru je nemoguća.

Višeslojne neuronske mreže mogu uključiti povratnu informaciju u jednu od dvije osnovne metode. Elman je uveo povratnu informaciju sa skrivenog sloja u odjeljku konteksta ulaznog sloja. Ova metoda daje više pažnje na redoslijed ulaznih varijabli. Jordanove rekurentne neuronske mreže naglašavaju slijed izlaznih vrijednosti korištenjem povratnih informacija od izlaznog sloja do kontekstnih čvorova ulaznog sloja.

Neuronske mreže s povratnim vezama i elementima dinamičke obrade nalaze sve više primjena u različitim područjima. Iako je propagaciju unatrag vrlo lako izgraditi, može naići na brojne probleme kada se koristi u stvarnim scenarijima, uključujući izazov izbjegavanja hvatanja u lokalne minimume, čiji je problem opisan u sljedećem podnaslovu. Vremenski odgođeno ažuriranje ulaznih podataka ponavljajućih neuronskih mreža dodaje složenost dinamičkoj obradi, zahtijevajući kompliciranije algoritme za predstavljanje učenja.

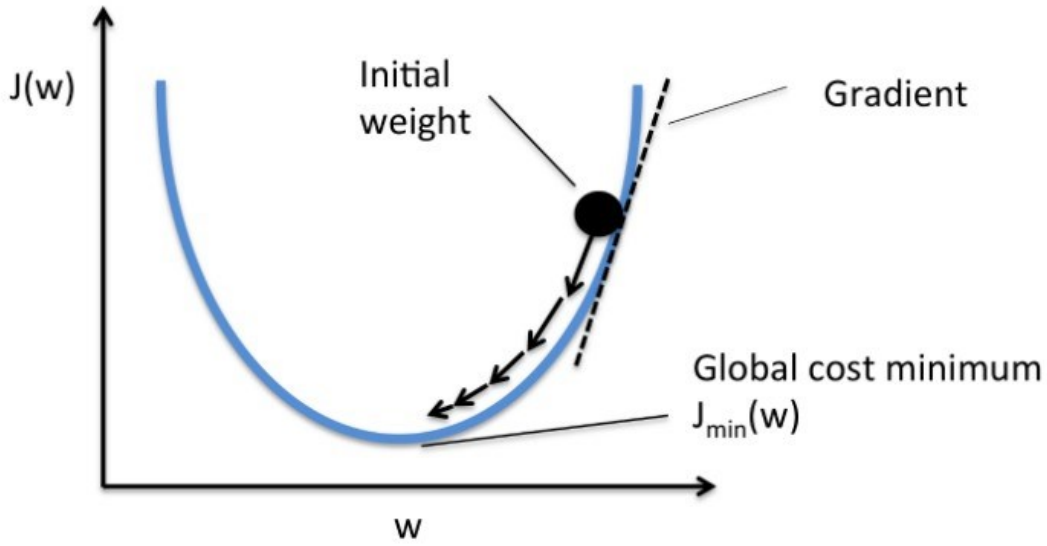
Hebbianovo učenje i učenje s gradijentom uspona ključni su koncepti na kojima su utemeljene tehnike neuronske mreže.

2.7. Optimizacija mreže

Najlakši način za učenje je prikupiti veliki broj primjera (potrebno je više primjera za teže probleme) koji pokazuju sve različite aspekte problema.

Kako bi se izgradila otporna i pouzdana mreža, neki šum ili druga slučajnost se ponekad uključuje u podatke obuke kako bi se mreža upoznala s bukom i prirodnom varijabilnosti u stvarnim podacima. Loši podaci o obuci uvijek rezultiraju mrežom koja je nepouzdana i nepredvidiva. Mreža se obično trenira za unaprijed određeni broj epoha ili dok izlazna pogreška ne padne ispod određenog praga.
Kako bi greška bila što manja, postoje algoritmi koji pomažu optimizirati rezultate a nazivaju se algoritmi gradijenta pada (Slika 2.5). Zbog simetrije, neuronska mreža počinje sa slučajnim vrijednostima parametara, jer da su sve vrijednosti iste, svi neuroni u skrivenom sloju bi imali iste izlazne vrijednosti, što bi rezultiralo da se isti gradijent izračunava tijekom algoritma, a prilagodbe težine bi bile iste za svaku mrežu. Funkcija pogreške omogućuje kvantificiranje kvalitete bilo kojeg skupa težina *W*. Cilj optimizacije je otkriti skup težina koji ima najmanju funkciju pogreške. Pronalaženje najboljeg *W* je nedostižno odmah, ali pronalaženje boljeg *W* iterativno je zamislivo.

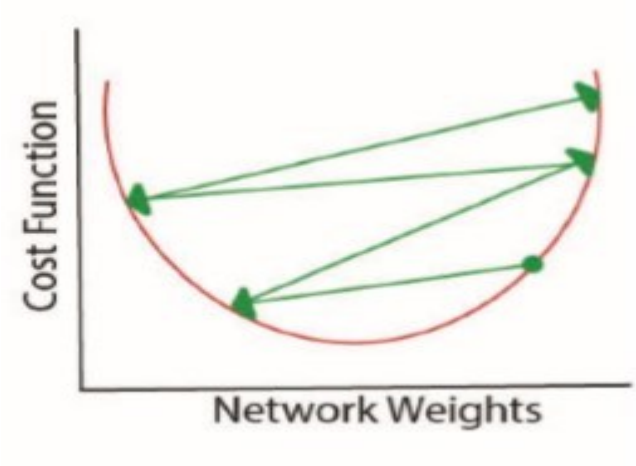
Zbog gore spomenutog obrazloženja, mreža počinje sa slučajnim vrijednostima parametara i mijenja ih iterativno tako da se vrijednost pogreške smanjuje sa svakom iteracijom. Algoritmi koji nam služe kako bi uravnotežili preciznost su: gradijent pada skupa (engl. *Batch gradient descent*), stohastički gradijent pada (engl. *Stochastic gradient descent*) te gradijent pada podskupa (engl. *Mini-batch gradient descent*). Razlike između algoritama su u količini podataka koje koriste za svoje operacije i vrijeme koje im je potrebno za prilagodbu parametara.



Slika 2.5. Funkcioniranje gradijentnog spusta [13]

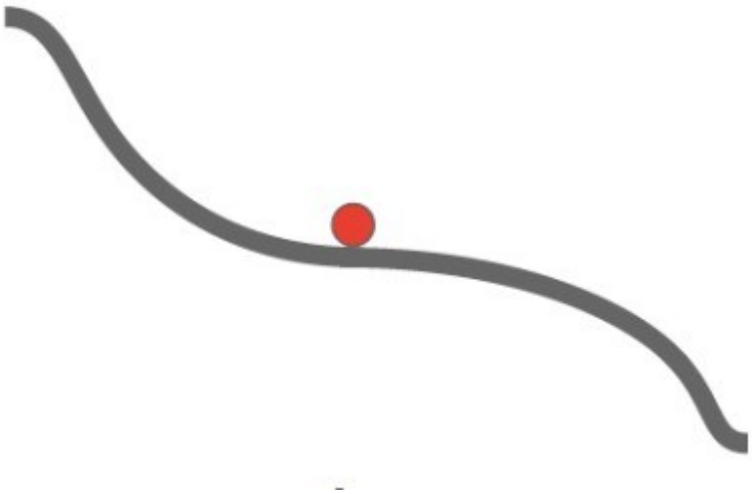
Smanjivanje pogreške ovim metodama vuče za sobom i nove probleme, a neki od najvažnijih su:

a) Gradijent koji nestaje (engl. *Vanishing Gradient*): scenarij u procesu učenja neuronskih mreža gdje model uopće ne uči. To je zbog toga što kada gradijent postane premali, gotovo nestaje te dovodi do zaglavljivanja utega i nikada ne dostiže optimalnu vrijednost za minimalni gubitak (globalni minimumi). Stoga mreža nije u stanju učiti i konvergirati.
b) Eksplodirajući gradijent: upravo suprotno od nestajanja gradijenta. Kada model nastavlja učiti, težine se nastavljaju ažurirati na jako velike vrijednosti, ali model se nikada ne konvergira. Izračunava gradijent (gubitak) u odnosu na težine koji postaje iznimno velik u ranijim slojevima na takav način da eksplodira. Nastavlja oscilirati, uzimajući velike korake kao što je prikazano u nastavku na slici 2.6 i odstupi od točke konvergencije dok se udaljava od nje.



Slika 2.6. Preskakanje lokalnog minimuma [14]

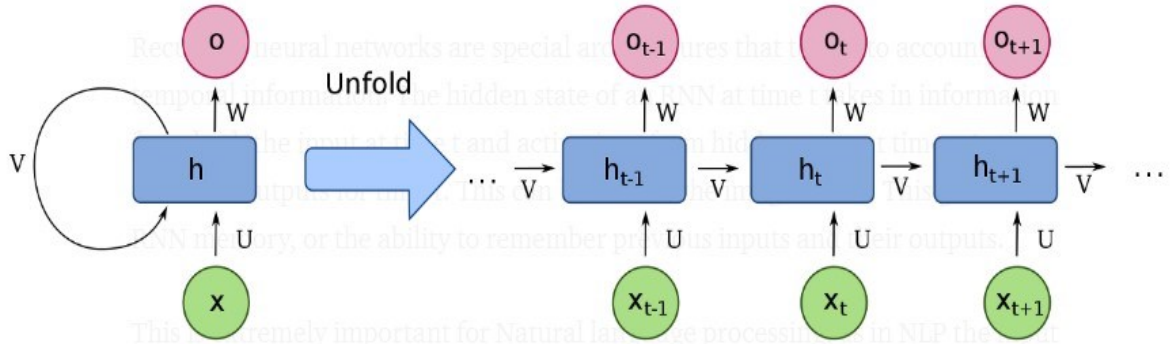
c) Točka sedla (eng: *Saddle Point - MiniMax Point*): Na površini funkcije gubitka, točka sedla je diplomatska točka u kojoj se, iz jedne dimenzije, čini da je kritična točka najmanja, dok se iz druge dimenzije čini da je najveća, što se može vidjeti u nastavku (Slika 2.7).



Slika 2.7. Grafički prikaz točke sedla [13]

2.8. Obrada prirodnog jezika

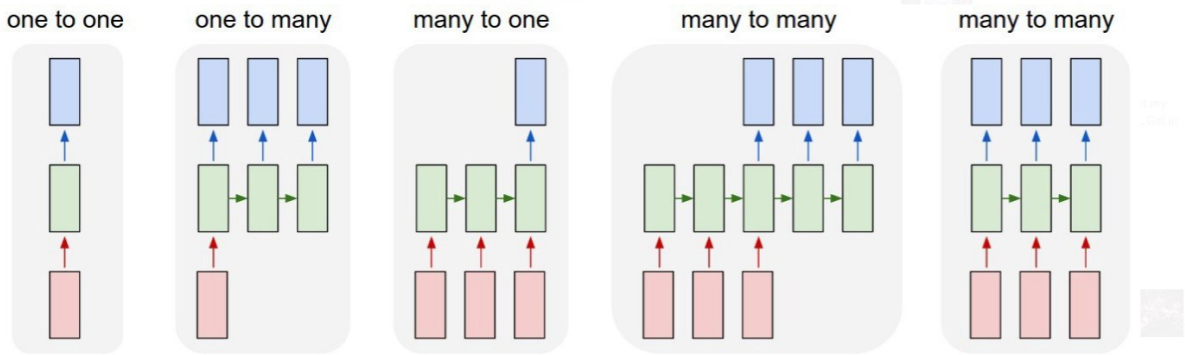
Činjenica da rekurentne neuronske mreže uzimaju u obzir vremenske informacije čini ih izuzetno važnim za obradu prirodnog jezika, budući da u NLP-u (engl. *Natural language processing*) ulazni podaci nemaju fiksnu veličinu, a sljedeća riječ jako ovisi o prethodnim riječima.



Slika 2.8. Način pamćenja RNN-a [15]

Skriveno stanje RNN-a u trenutku t uzima informacije iz oba ulaza u vrijeme t i aktivacija od skrivenih jedinica u trenutku t-1, za izračunavanje izlaza za vrijeme t. To se može vidjeti na slici 2.8. To daje RNN memoriju, odnosno mogućnost pamćenja prethodnih ulaza i njihovih izlaza. U NLP-u je kontekst ključan. Svaka rečenica ima različitu duljinu, a svaka rečenica daje drugačiji rezultat. Također je iznimno korisno za RNN-ove da mogu izračunati izlaze različitih veličina iz promjenjivih veličina ulaza. RNN-ovi mogu zadržati kontekstualne informacije jer imaju i sjećanja.

RNN arhitekture mogu proizvesti više vrsta ulaznih i izlaznih oblika, prema slici 2.9.



Slika 2.9. Moguće RNN arhitekture [16]

a) U arhitekturi jedan na jedan (engl. *one to one*), trenutna riječ u rečenici služi kao ulaz, dok sljedeća riječ služi kao izlaz. I ulaz i izlaz dugi su jednu riječ. Jezični model nastaje kada se prediktor riječi uzastopno spaja kako bi se proizvele rečenice, pa čak i tekst.
b) Ako želimo generirati natpis rečenice za jednu sliku, potreban nam je model jedan prema više. Označavanje slika izvrstan je primjer jednog ulaza, a višestrukih izlaza.
c) Za predviđanje tona izjave ili recenzije može se koristiti model više-prema-jedan. Možemo unijeti nekoliko riječi iz naše evaluacije i proizvesti jedan ishod (osjećaj: pozitivan ili negativan).
d) Strojno prevođenje može koristiti model mnogo na mnogo. Ako imamo rečenicu na engleskom i želimo je prevesti na npr. hrvatski, cilj je da se više riječi na engleskom prevede u više riječi na hrvatski, ali količina riječi ili redoslijed možda nisu isti.

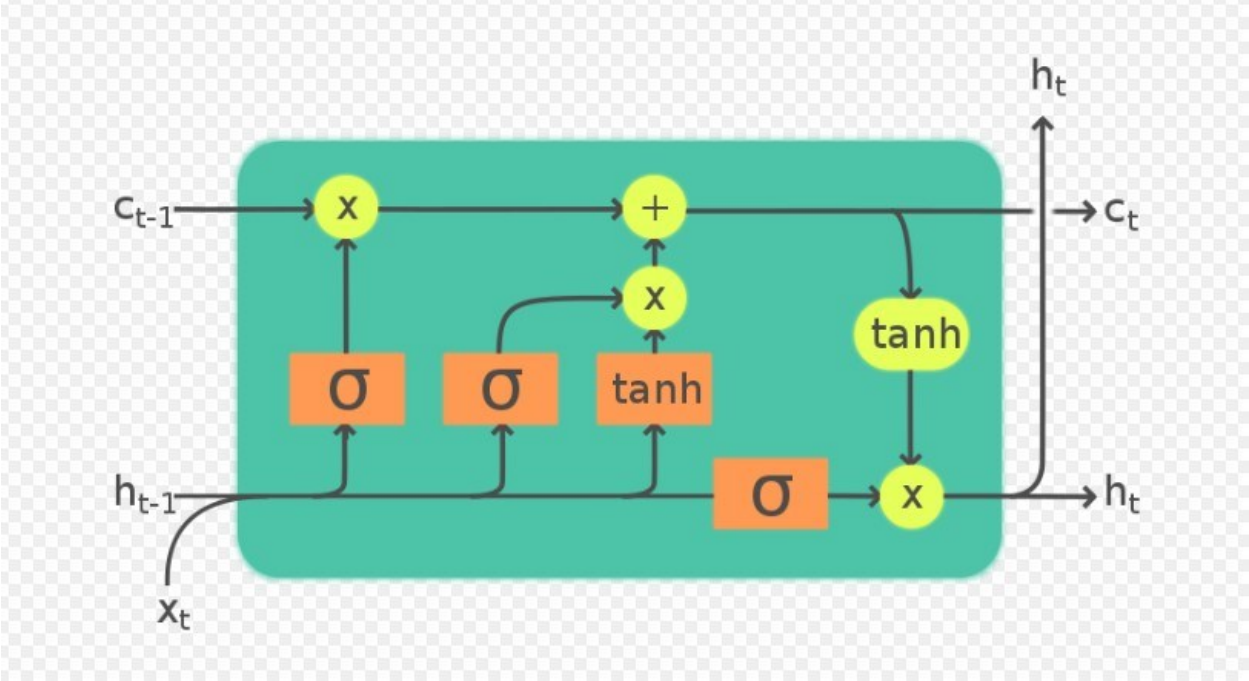
2.9. Mreža dugotrajne/kratkoročne memorije

RNN-ovi su imali jednu očiglednu lošu stranu zbog koje je bilo gotovo nemoguće trenirati veće verzije, bili su iznimno skloni eksploziji gradijenta i problemima s nestajućim gradijentom.

Ukratko, ponavljajuća množenja su osnovni uzrok problema s pucanjem i nestajanjem gradijenta (u slučaju ovog rada, množenje matrice ulaza i težine). Kada se pomnože sami sa sobom, brojevi samo veći od 1 eksplodiraju u beskonačnost, dok kada se pomnože sami sa sobom, brojevi manji od 1 nestaju na nulu. Gradijent izbija i nestaje u RNN-u jer se ista matrica težine množi s ulazima i prethodnim izlazima. Dodatno, udaljenost između posljednje iteracije RNN-a i prve vrlo je velika za gradijentni protok. To ukazuje da RNN ima ograničen kapacitet za kontekstualne informacije prije nego što gradijenti izbiju ili nestanu. Zbog toga, jednostavni RNN-ovi imaju ograničenu memoriju i kratak referentni prozor, što je broj riječi ispred trenutne riječi iz koje mogu izdvojiti kontekstualne informacije.

Mreža dugotrajne/kratkoročne memorije (engl. *Long Short-Term Memory* - LSTM) rješava probleme RNN-a, a rješenje su predstavili njemački istraživači Sepp Hochreiter i Jürgen Schmidhuber u svom radu "Long short-term memory" 1997. godine [17], iako se originalna verzija nakon mnogo iteracija dosta promijenila kako bi postigla popularnost kakvu danas ima.

LSTM rješenje problema postiže očuvanjem stanja stanice, što je trenutno stanje. U svakom vremenskom koraku, stanje ove ćelije ažurira se relevantnim podacima. Ulaz, prethodni izlaz i ažurirano stanje ćelije koriste se za određivanje izlaza u svakom vremenskom koraku.



Slika 2.10. Arhitektura stanice LSTM mreže [15]

Tri vrata - ulazna vrata, vrata zaborava i vrata kandidata - koriste se za ažuriranje stanja ćelije (koja se ponekad naziva i vrata vrata). Ulaz koji je obrađen izlaznim vratima i najnovije stanje ćelije koriste se za izračunavanje izlaza. Jednostavno, *sigmoid* ili *tanh* ulaza i prethodnog izlaza, nakon čega slijedi umnožak ili zbroj, sve je što je potrebno za svaka vrata (Slika 2.10.).

Temeljna prednost usvajanja stanja ćelije je u tome što se može lako ažurirati i održavati, a djeluje kao gradijentna autocesta, dopuštajući gradijentima da se propagiraju unatrag i sprječavaju probleme nestajanja i pucanja gradijenta.

3. KORIŠTENE TEHNOLOGIJE

3.1. Python

Python je objektno orijentirani programski jezik koji je interpretiran i interaktivan. Uključuje strukture podataka visoke razine kao što su popisni i asocijativni nizovi (također poznati kao rječnici), dinamičko tipkanje i dinamičko uvezivanje, module, klase, iznimke i automatsko upravljanje memorijom, između ostalog. To je moćan programski jezik opće namjene sa zapanjujuće jednostavnom i elegantnom sintaksom. Dizajnirao ga je Guido van Rossum 1990. godine [18].

Besplatan je, čak i za komercijalnu upotrebu, i može se pokrenuti na gotovo svakom modernom računalu, kao i mnogi drugi skriptni jezici. Interpretator automatski kompilira python program u bajt kod neovisan o platformi, koji se zatim interpretira. Python komponente mogu se pokrenuti na Linuxu, Windows NT, 98, 95, IRIX, SunOS i OSF operacijskim sustavima.

Python je dizajniran tako da bude modularan. Kernel je jednostavan program koji se može proširiti učitavanjem modula proširenja. Python distribucija uključuje veliku biblioteku standardnih proširenja (neke su napisane u Pythonu, druge u C ili C++ programskom jeziku) za širok raspon zadataka, uključujući manipulacije nizovima i regularne izraze slične Perlu, kao i uslužne programe povezane s webom, usluge operacijskog sustava, alati za otklanjanje pogrešaka i profiliranje i tako dalje. Za proširenje jezika novim ili povijesnim kodom, mogu se napisati novi moduli proširenja. Članovi Python korisničke zajednice napisali su i objavili značajan broj modula proširenja. GADFLY, SQL upravitelj baze podataka napisan na Pythonu, PIL (Pythonova biblioteka slika), FNORB i OmniBorker, CORBA kompatibilni posrednik zahtjeva za objekte (ORB) napisani na Pythonu, Gendoc, automatizirani dokumentacijski alat; i Numeric Python samo su nekoliko primjera ovih modula proširenja, također poznatih kao "paketi" ili komponente.

3.2. TensorFlow

Među mnoštvom biblioteka dubokog učenja, TensorFlow [19], kojeg su dizajnirali Googleovi istraživači, najpopularniji je. Neuronske mreže postigle su veliki uspjeh u području dubokog učenja i stekle su veliku pažnju u raznim područjima. Zbog svoje svestranosti i skalabilnosti, ova obitelj modela nudi puno potencijala za promicanje analize podataka i modeliranja za mnoge

izazove u obrazovnim i bihevioralnim znanostima. Pruža pregled osnova modela neuronskih mreža kao što su višeslojni perceptron, konvolucijska neuronska mreža i stohastički gradijentni silazak, koji je najčešće korišteni pristup optimizaciji za modele neuronskih mreža. Međutim, korištenje i stavljanje ovih modela i tehnika optimizacije u praksu može zahtijevati vrijeme i podložno je pogreškama, ali TensorFlow, srećom, znatno pojednostavljuje i ubrzava razvoj i korištenje modela neuronskih mreža. Omogućeni su ključni principi TensorFlowa, uključujući funkcije za stvaranje grafa, alati za izvršavanje grafa i TensorBoard, TensorFlow-ov alat za vizualizaciju. Koristi se za stvaranje i treniranje modela konvolucijske neuronske mreže koja može klasificirati rukom napisane znamenke. Evaluacija završava usporedbom između sučelja za programiranje aplikacija niske i visoke razine, kao i raspravom o podršci knjižnice TensorFlow Probability za jedinice za obradu grafike, distribuiranu obuku i vjerojatnosno modeliranje.

3.3. Keras

Keras je u početku razvijen kao dio istraživačkog rada projekta ONEIROS (*Open-ended Neuro-Electronic Intelligent Robot Operating System*).

Povrh okvira za strojno učenje TensorFlow, Keras je aplikacijsko sučelje za duboko učenje temeljen na Pythonu [20]. Stvoren je s ciljem olakšavanja brzog eksperimentiranja. Tajna provođenja učinkovitog istraživanja je u mogućnosti brzog prijelaza od začeća do zaključka. Jednostavan i fleksibilan, kako bi smanjio kognitivni pritisak na programere tako da se mogu koncentrirati na ključne aspekte problema. Uz to, pridržava se pojma progresivnog otkrivanja složenosti, koji kaže da bi proizvoljni komplicirani tijekovi rada trebali biti izvedivi putem jasnog puta koji se temelji na onome što je već naučeno. Jednostavni tijekovi rada trebaju biti brzi i laki. Snažan jer nudi performanse i skalabilnost koji su konkurenti najboljima u poslu; NASA, YouTube i Waymo samo su neke od korporacija koje ga koriste. *User-friendly*, odnosno prilagođen korisniku, vrlo učinkovito sučelje za rješavanje problema strojnog učenja s naglaskom na suvremeno duboko učenje. Nudi ključne građevne elemente i apstrakcije za brzo stvaranje i isporuku rješenja za strojno učenje. Inženjeri i istraživači mogu pokrenuti Keras na TPU-u (Googleovi prilagođeni razvijeni integrirani sklopovi) koji se koriste za ubrzavanje radnih opterećenja strojnog učenja) ili na masivnim skupinama GPU-a (engl. *graphics processing unit*), a Keras modeli mogu se izvesti za rad u pregledniku ili na mobilnom uređaju, omogućujući da se u potpunosti iskoriste skalabilnost TensorFlow 2 i međuplatformske značajke.

3.4. Google Colaboratory

Colaboratory, ili skraćeno "Colab", proizvod je Google Researcha. Colab je web-bazirani Python uređivač koji svakome omogućuje pisanje i pokretanje proizvoljnog Python koda. Posebno je koristan za strojno učenje, analizu podataka i obrazovanje. Colab je, tehnički rečeno, „domaćinska“ Jupyter usluga prijenosnih računala koja ne zahtijeva instalaciju i pruža besplatan pristup računalnim resursima, uključujući GPU-ove [21].

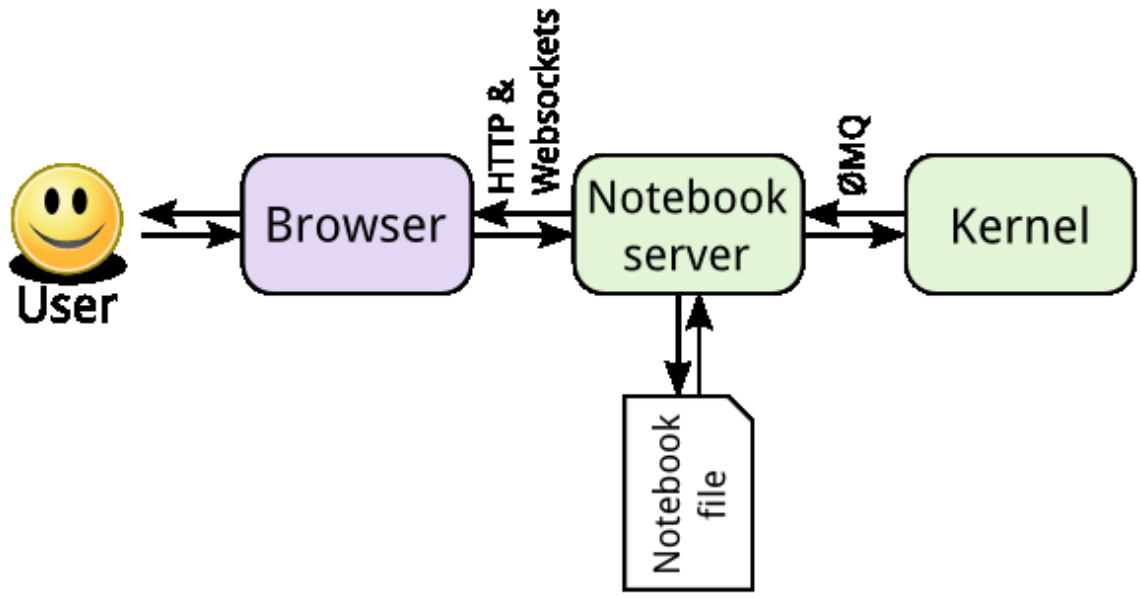
Nevjerojatno moćni grafički procesori Tesla K80 grafičkih kartica, na kojima se izvodi implementacija i pokretanje programa, upravo su bili razlog za odabir ovog okruženja. Zbog toga je vrijeme evaluacije modela uvelike smanjeno. Colab bilježnice mogu se preuzeti s GitHuba ili čuvati na Google disku. Colab bilježnice mogu se dijeliti na isti način kao i Google dokumenti i tablice, što zbog svoje jednostavnosti i sigurnosti, uvelike pomaže prilikom suradnje na projektu.

3.5 Jupyter

JupyterLab je najnovije interaktivno razvojno okruženje za bilješke, kod i podatke koji su dostupni na webu. Korisnici mogu kreirati i organizirati tijekove rada u znanosti o podacima, znanstvenom računarstvu, računskom novinarstvu i strojnom učenju koristeći njegovo svestrano sučelje (Slika 3.1) [22]. Modularni dizajn potiče proširenja za poboljšanje i obogaćivanje funkcionalnosti. Izvorna web aplikacija za izradu i dijeljenje računskih dokumenata je Jupyter Notebook. Jupyter Notebook i njegovo prilagodljivo sučelje nadilaze kod i uključuju vizualizaciju, multimediju, suradnju i druge značajke. Ne samo da pokreće kod, već ga i sprema, zajedno sa svim bilješkama, u dokument koji se može uređivati pod nazivom bilježnica. Kada se spremi, preglednik ga šalje poslužitelju prijenosnog računala, koji ga sprema kao JSON datoteku s ekstenzijom .ipynb na disku.

Budući da je poslužitelj prijenosnog računala, a ne kernel, zadužen za pohranjivanje i učitavanje bilježnica, prilagodljiv je čak i ako ne postoji instalirana ljuska (engl. *kernel*) za taj jezik. Omogućuje jednostavno, pojednostavljeno okruženje usredotočeno na dokumente, a Python, R, Julia i Scala su među više od 40 programskih jezika koje podržava. E-pošta, Dropbox, GitHub i Jupyter Notebook Viewer mogu se koristiti za razmjenu zabilješki. HTML, slike, videozapisi, LaTeX i prilagođeni MIME tipovi su svi primjeri bogatog, interaktivnog izlaza koji kod može

generirati. Od Pythona, R i Scale koristite tehnologije velikih podataka kao što je Apache Spark. Moguće je koristiti pandas, scikit-learn, ggplot2 i TensorFlow kako bi se bolje mogli pogledati isti podaci.



Slika 3.1. Komponente Jupyter Notebooka [22]

4. PROGRAMSKO RJEŠENJE

Za rješenje je bilo potrebno proučiti optimalan dizajn slojeva same neuronske mreže, odrediti kojim će se programskim jezikom implementirati spomenuta neuronska mreža, koji će se skupovi podataka dati mreži kao ulazni podaci, te na koji način će se odrediti jesu li uvjeti zadovoljeni ili ne [23].

4.1. Odabir modela

Kako bi mreža bila što efikasnija, potrebno je smisliti od koliko slojeva će se umjetna neuronska mreža sastojati, koliko neurona će svaki sloj obuhvaćati te pomoću kojih aktivacijskih funkcija će rezultati biti određeni na izlazu. Vrlo je važno ne pretrenirati mrežu. Mreža može postati pretjerano prilagodljiva u učenju uzoraka iz skupa za treniranje kao rezultat pretreniranosti [24], te stoga možda neće moći učinkovito identificirati podatke izvan skupa za obuku.

Osnovni princip funkcioniranja umjetnih neuronskih mreža je davanje dovoljno velikog broja primjerenih podataka, kako bi mreža mogla kreirati obrasce po kojima kategorizira rješenja te odrediti prag koji zadovoljava inicijalno postavljene kriterije uspješnog rješenja.

Odabir broja neurona - koliko uspješno mreža može odvojiti podatke ovisi o tome koliko ima skrivenih neurona. Mreža može ispravno predvidjeti podatke na kojima je obučena, a veliki broj skrivenih neurona osigurat će točno učenje, no njezina izvedba na novim podacima i sposobnost generalizacije su otežani. Mreža možda neće moći razumjeti veze između podataka ako nema dovoljno skrivenih neurona, a pogreška se neće moći smanjiti. Stoga je odabir broja skrivenih neurona važan izbor.

Početne težine - algoritam učenja koristi tehniku spuštanja koja se stalno spušta u prostoru težine sve dok ne dođe do prve doline. Zbog toga je ključno odabrati početnu točku u višedimenzionalnom prostoru težine. Međutim, osim eksperimentiranja s različitim početnim vrijednostima težine kako bi se utvrdilo jesu li rezultati mreže poboljšani, ne postoje predloženi kriteriji za ovaj odabir. Tako da model koristi nasumične težine.

Brzina učenja - kada se svaka težina ažurira, brzina učenja učinkovito regulira količinu koraka koji se poduzima u višedimenzionalnom prostoru težine. Lokalni minimum može biti kontinuirano

prekoračen ako je odabrana stopa učenja previsoka, što dovodi do oscilacija i spore konvergencije prema nižem stanju pogreške. Spora izvedba mogla bi proizaći iz velikog broja iteracija ako je stopa učenja preniska. Korištena brzina je zadana vrijednost brzine učenja tijekom korištenja Adam optimizacijskog algoritma [25], a iznosi 0.001.

Slojevi - slojevi su osnovni gradivni blokovi neuronskih mreža u Kerasu. Sloj se sastoji od metoda poziva sloja i nekog stanja koje se drži u varijablama TensorFlow (težine sloja). Instanca sloja se može pozvati, slično kao funkcija, to je vidljivo na slici 4.1.

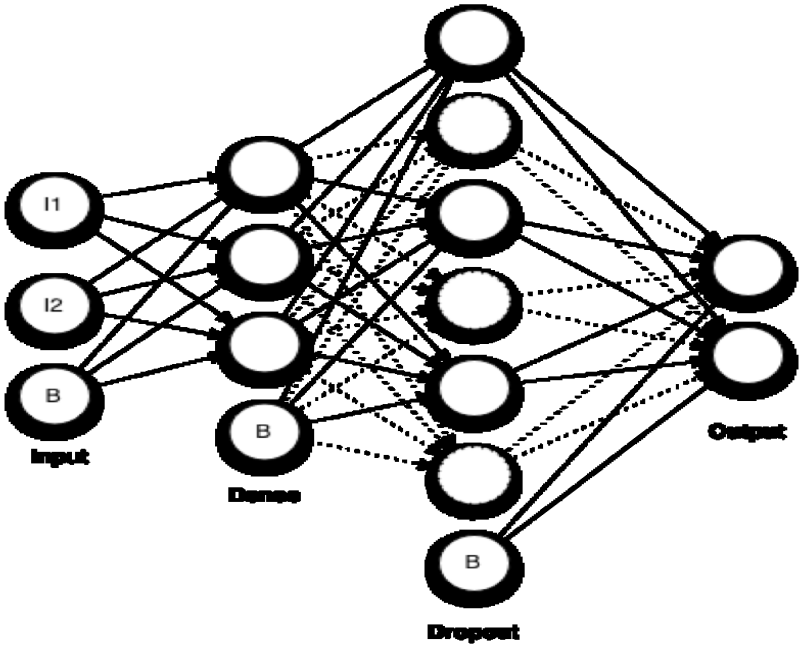
```
from tensorflow.keras import layers

layer = layers.Dense(32, activation='relu')
inputs = tf.random.uniform(shape=(10, 20))
outputs = layer(inputs)
```

Slika 4.1. Primjer dodavanja sloja

Svi parametri koji nisu eksplicitno navedi, a funkcija ih zahtjeva, imaju predefinirane vrijednosti od strane Kerasa.

Vrste korištenih slojeva:

a) *Embedding* sloj - Ovo se uglavnom koristi u aplikacijama vezanim za obradu prirodnog jezika kao što je modeliranje jezika, ali se također može koristiti s drugim zadacima koji uključuju neuronske mreže [26, 27].
b) LSTM sloj - jedno od glavnih svojstava LSTM-a je pamćenje i prepoznavanje informacija koje dolaze unutar mreže, kao i odbacivanje informacija koje nisu potrebne mreži za učenje podataka i predviđanja [28, 29, 30].
c) *Dropout* sloj - rješava problem pretreniranja dubokih neuronskih mreža s velikim brojem parametara, kako se ne bi prekomjerno prilagodile kombiniranju tijekom predviđanja za vrijeme testiranja [31] (Slika 4.2.).



Slika 4.2. Funkcioniranje *dropouta* [32]

d) *Dense* sloj - ovaj sloj pomaže u promjeni dimenzionalnosti izlaza iz prethodnog sloja tako da model može lako definirati odnos između vrijednosti podataka u kojima model radi [33].

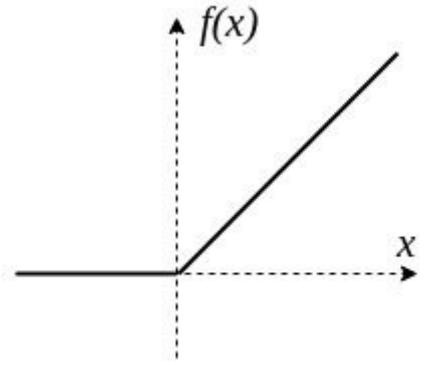
Broj slojeva mreže - tijekom testiranja mreže, broj slojeva se mijenjao ovisno o rezultatima testiranja, koji je u konačnici ostao na pet skrivenih slojeva (Slika 4.3).

```
[11] model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 40, 32)            640000

 lstm (LSTM)                 (None, 64)                24832

 dropout (Dropout)           (None, 64)                0

 dense (Dense)               (None, 6)                 390

 dense_1 (Dense)             (None, 1)                 7

=================================================================
Total params: 665,229
Trainable params: 665,229
Non-trainable params: 0
_________________________________________________________________
```

Slika 4.3. Broj i vrsta korištenih slojeva u modelu

Aktivacijske funkcije - aktivacijska funkcija u neuronskoj mreži definira kako se težinski zbroj ulaza pretvara u izlaz iz čvora ili čvorova u sloju mreže. Postoji nekoliko različitih vrsta aktivacijskih funkcija [34]. Funkcije koje se koriste za skrivene slojeve i funkcije koje se koriste za izlazni sloj. U nastavku se može vidjeti definicija nekolicine:

1) Ispravljena linearna aktivacija (ReLU – engl. *Rectified Linear Unit*) - najčešća funkcija koja se koristi za skrivene slojeve. Uobičajena je jer je jednostavna za implementaciju i učinkovita u prevladavanju ograničenja drugih aktivacijskih funkcija, kao što su *Sigmoid* i *Tanh*. Točnije, manje je osjetljiva na nestajuće gradijente koji sprječavaju obuku dubokih modela, iako može patiti od drugih problema poput zasićenih ili „mrtvih“ jedinica. ReLU funkcija se izračunava na sljedeći način, slika 4.4.



Slika 4.4. ReLu aktivacijska funkcija [35]

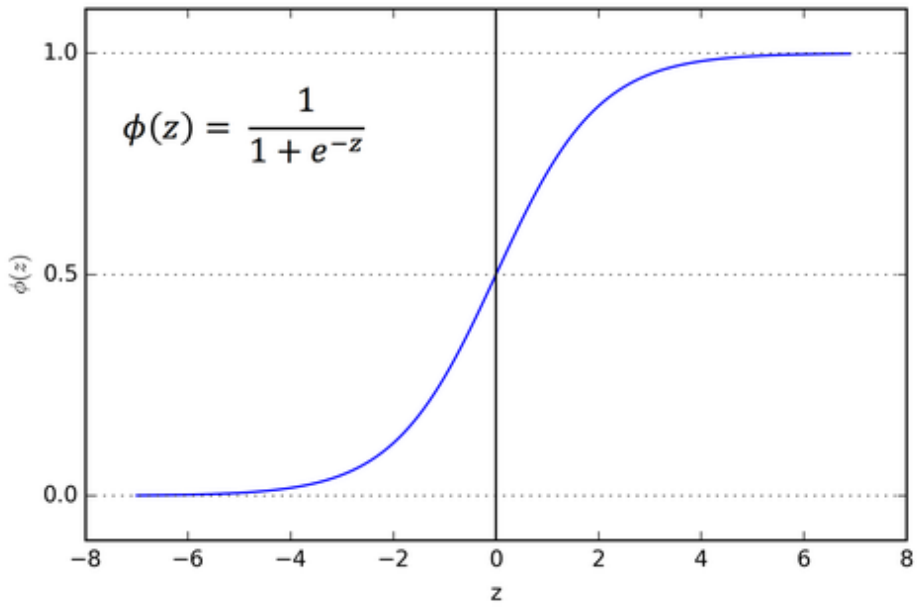
max(0.0, x) ,to znači da ako je ulazna vrijednost (x) negativna, tada se vraća vrijednost 0,0, u suprotnom se vraća vrijednost(x).

2) Hiperbolički tangent (*Tanh*) aktivacijska funkcija - vrlo je slična sigmoidnoj aktivacijskoj funkciji i čak ima isti S-oblik. Funkcija uzima bilo koju stvarnu vrijednost kao ulaz i daje vrijednosti u rasponu od -1 do 1. Što je veći ulaz (pozitivniji), to će izlazna vrijednost biti bliža 1,0, dok što je manji ulaz (negativniji), to će izlaz biti bliže -1,0. *Tanh* aktivacijska funkcija izračunava se prema formuli (4-1).

$$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad (4\text{-}1)$$

Gdje je e matematička konstanta koja je baza prirodnog logaritma.

3) *Sigmoid* aktivacijska funkcija - funkcija koja se koristi u algoritmu klasifikacije logističke regresije. Funkcija uzima bilo koju stvarnu vrijednost kao ulaz i daje vrijednosti u rasponu od 0 do 1 (formula 4-2). Što je veći ulaz (pozitivniji), to će izlazna vrijednost biti bliža 1.0, dok što je manji ulaz (negativniji), to će izlaz biti bliži 0.0. *Sigmoid* aktivacijska funkcija prikazana je na Slici 4.5.



Slika 4.5. Grafički prikaz sigmoidne aktivacijske funkcije [36]

$$\frac{1}{1+e^{-z}} \tag{4-2}$$

Gdje je e matematička konstanta, koja je baza prirodnog logaritma.

Keras u svojim slojevima ima predefinirane aktivacijske funkcije za LSTM sloj kao i za *Dense* sloj, gdje je za LSTM podrazumijevana funkcija *tanh*, dok je u skrivenom *Dense* sloju eksplicitno korištena Relu funkcija a u izlaznom sloju sigmoidna funkcija.

Model – korišten je Keras za izgradnju modela, te njegova predefinirana klasa *Sequential*() koja je prikazana na Slici 4.6.

```
tf.keras.Sequential(layers=None, name=None)
```

Slika 4.6. Keras *Sequential* klasa

Kao što samo ime kaže, dozvoljava sekvencijalno grupiranje snopa slojeva u model i pruža značajke obuke i zaključivanja na modelu, prikazano na Slici 4.7.

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8))
model.add(tf.keras.layers.Dense(1))
model.compile(optimizer='sgd', loss='mse')
# This builds the model for the first time:
model.fit(x, y, batch_size=32, epochs=10)
```

Slika 4.7. Kreiranje nasumičnog modela

4.2. Kreiranje modela

Dok radimo s tekstualnim podacima, moramo ih pretvoriti u brojeve prije nego što ih unesemo u bilo koji model strojnog učenja, uključujući neuronske mreže.

Da bi se riječi mogle poistovjetiti s brojevima, potrebno je svaku riječ iz seta podataka povezati s određenom vrijednosti. Za to služi Keras-ova klasa za predobradu *Tokenizer*. Ona omogućuje vektorizaciju strukturiranog skupa tekstova pretvaranjem svakog teksta u niz cijelih brojeva (sekvenci) (svaki cijeli broj je indeks oznake u rječniku) ili u vektor. Prema zadanim postavkama, sva interpunkcija se briše iz tekstova, ostavljajući samo razmak između riječi, od kojih neke mogu uključivati ' znak. Zatim se iz tih sekvenci kreiraju popisi tokena, koji će biti vektorizirani ili indeksirani. Koeficijent za svaki token može biti binarni, na temelju broja riječi i ostalo (Slika 4.8).

```
tokenizer = Tokenizer(num_words = vocabulary_size)
tokenizer.fit_on_texts(train_data)
word_index = tokenizer.word_index

print(word_index)
```

```
{'i': 1, 'to': 2, 'you': 3, 'a': 4, 'the': 5, 'u': 6, 'and': 7, 'is': 8, 'in': 9, 'me': 10
```

Slika 4.8. Dodavanje numeričke vrijednosti svakoj riječi

Kako bi se postiglo da svaka rečenica koja ulazi u *embedding* sloj sadrži isti broj riječi, koristi se funkcija *padding*, a funkcionira na način da razliku između rečenica popuni određenom vrijednosti, u ovom slučaju (Slika 4.9), nulama.

```
sequences = tokenizer.texts_to_sequences(train_data)
test_sequences = tokenizer.texts_to_sequences(test_data)
print(sequences[0])

padded = pad_sequences(sequences,maxlen=sentence, padding='post')
test_padded = pad_sequences(test_sequences,maxlen=sentence, padding='post')
print(padded[0])

print(len(sequences[0]), len(sequences[1]))
print(len(padded[0]), len(padded[1]))
```

```
[52, 424, 3837, 812, 813, 572, 67, 9, 1260, 84, 128, 331, 1436, 143, 2587, 1123, 65, 59, 3838, 141]
[  52  424 3837  812  813  572   67    9 1260   84  128  331 1436  143
 2587 1123   65   59 3838  141    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0]
20 6
40 40
```

Slika 4.9. Prikaz izjednačavanja dužina rečenica

Sloj za ugrađivanje (engl. *Embedding*) omogućuje nam pretvaranje svake ulazne riječi u vektor fiksne duljine definirane veličine. Radi jednostavnosti rečenice se mogu usporediti s kategoričkim varijablama.Za to se koristi jednokratno kodiranje kako bi se kategoričke značajke pretvorile u brojeve. Kako je u ovom radu tražena klasifikacija neželjene pošte, radi se o tome da su regularni mailovi označeni brojem 0, dok je neželjena pošta označena brojem 1. Rezultirajući vektor je gust i ne sadrži samo 0 i 1, već i stvarne vrijednosti. Fiksna duljina vektora riječi i smanjene dimenzije omogućuju nam učinkovitije izražavanje riječi (Slika 4.10).

```
(1, 2)
[[[ 0.04502351  0.00151128  0.01764284 -0.0089057 ]
  [-0.04007018  0.02874336  0.02772436  0.00842067]]]
```

Slika 4.10. Prikaz dvije riječi provedene kroz Embedding sloj

Prilikom kreiranja *Embedding* sloja, postoje tri parametra, a to su ulazna veličina, odnosno veličina rječnika, izlazna veličina, odnosno duljina svakog vektora te maksimalna duljina niza koji se obrađuje odnosno duljina rečenice. Nakon istraživanja, prvi parametar je postavljen na 20 000, jer se radi o setu podataka koji sadrži 5 tisuća rečenica. Drugi parametar je postavljen na 32 jer manji broj ne daje dovoljno informacija, dok veći brojevi mogu davati nepotrebne informacije te mogu zbuniti mrežu. Treći parametar je postavljen na 40 jer po provedenom istraživanju [37], poželjna prosječna dužina rečenice u mailu iznosi između 28 i 50 riječi.

LSTM arhitektura omogućuje mreži da pronalazi korelacije između riječi i rečenica jer je sposobna analizirati podatke i naučiti dugoročne ovisnosti, posebno u problemima predviđanja slijeda. Radi tako da eliminira neiskorištene informacije i pamti slijed informacija pomoću svojih ćelija. Ulazna vrata ocjenjuju točnost i značaj podataka za izradu predviđanja te odlučuju o važnosti informacija ažuriranjem stanja ćelije. Vrata zaboravljanja određuju mogu li se podaci kretati kroz različite razine mreže. Informacije koje nije potrebno naučiti o prognozama odbacuje.

Dropout sloj služi kako se mreža prilikom treniranja ne bi prekomjerno prilagodila podacima te na testnim primjerima dala loše rezultate.

Svaki neuron u *Dense* sloju prima ulaz od svakog neurona u sloju prije njega, što je poznato kao „duboka veza“. On množi matrice s vektorima, a vrijednosti matrice su parametri koji služe kako bi se mreža mogla trenirati i poboljšati uz pomoć povratnog širenja. Na izlazu je još jedan sloj koji se sastoji od samo jednog neurona i sigmoidne aktivacijske funkcije, koji služi da prezentira binarni rezultat 1 ili 0, odnosno kaže je li poruka prepoznata kao spam ili ne.

Prilikom sastavljanja mreže postavljen je Adam optimizacijski algoritam te binarna križna entropija koja uspoređuje stvarne rezultate s predviđenim te prikazuje koliko su udaljeni rezultat i stvarna vrijednost.

Kreiranje modela se završava predajom parametara i određivanjem težina (Slika 4.11).

```
weights= model.get_weights()
print(weights)
```

```
[array([[-0.04436097, -0.01054129,  0.01997608, ..., -0.00912695,
         -0.00591241, -0.03568421],
        [-0.06112846, -0.08427896,  0.06286675, ..., -0.03732788,
         -0.05644974, -0.07379684],
        [-0.03476398, -0.0206798 ,  0.00249491, ...,  0.00584932,
          0.02446158, -0.00596397],
        ...,
        [ 0.01490085,  0.00223935,  0.03425306, ..., -0.04175204,
         -0.00177326,  0.03674146],
        [ 0.04623813,  0.04360092, -0.00313727, ...,  0.00180254,
         -0.03318089,  0.02727078],
        [ 0.00425329,  0.01262427, -0.00031185, ..., -0.00295241,
          0.0046641 ,  0.02609501]], dtype=float32), array([[ 0.03132696, -0.00920295,  0.04075328, ...,  0.00821998,
          0.11030928,  0.03106965],
        [-0.09619351, -0.03933043,  0.01275695, ..., -0.07520334,
          0.0340211 ,  0.03793649],
        [ 0.02227843, -0.06805266, -0.14797468, ..., -0.02471679,
         -0.00423211,  0.04906042],
        ...,
```

Slika 4.11. Odabrane nasumične težine

4.3. Rezultati

Početni podaci koji su bili korišteni u istraživanju su kolekcija neželjene pošte „Spam Text Message Classification“ [38], a postoje dvije kategorije rezultata dobivenih provlačenjem spomenutih podataka kroz mrežu. To su rezultati prvog dijela primjera mreže gdje mreža uči, te rezultati stvarnih zadanih primjera. Točnost, odnosno preciznost rezultata uspoređuje se s postavljenim pragom te optimalnim rješenjima koja se mogu pronaći kod velikih korporacija koje se bave sličnim problemom.

Hiperparametri poput broja epoha te broja uzoraka podataka u epohi, odnosno jednom prolasku kroz podatke su određeni pomoću iskustva i *keras.callbacks.EarlyStopping()* klase koja ne dopušta povećanje razlike između stvarnih rezultata i predviđenih, već zaustavlja model ako primijeti pogoršanje. Optimalni hiperparametri su na kraju iznosili 512 broj uzoraka po epohi, te 15 epoha. Rezultati će biti vidljivi dalje u radu (Tablica 4.2.).

Model je prvotno napravljen kao mreža s GlobalAveragePooling1D slojem, koji je sažimao vrijednosti *Embedding* sloja, te na temelju sličnosti vrijednosti vektora pokušavao pronaći korelaciju između riječi (Slika 4.12.).

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, 40, 32)            640000

 global_average_pooling1d (G  (None, 32)                0
 lobalAveragePooling1D)

 dropout_1 (Dropout)          (None, 32)                0

 dense_2 (Dense)              (None, 6)                 198

 dense_3 (Dense)              (None, 1)                 7

=================================================================
Total params: 640,205
Trainable params: 640,205
Non-trainable params: 0
_________________________________________________________________
```

Slika 4.12. Slojevi modela koristeći *GlobalAveragePooling1D()*

Testiranje je bilo izvršeno koristeći ručno unesene regularne mailove kao i spam mailove, što se može vidjeti na jednom od primjera na slici 4.13.

```
[30] text= [" Guaranteed CASH MONEY only if you call us at - 0800 555 7475!! ",
            "Did you see what happened yesterday with Melissa?",
            "You just won FREE holiday in our best hotel on shore!",
            "Can you help me with moving to  place today?",
            "Book your favorite holiday places for discount price in July!"]
     CheckRandomInput(text)

      Guaranteed CASH MONEY only if you call us at - 0800 555 7475!!
     [0.25199008]


     Did you see what happened yesterday with Melissa?
     [0.223349]


     You just won FREE holiday in our best hotel on shore!
     [0.25929904]


     Can you help me with moving to  place today?
     [0.23130766]


     Book your favorite holiday places for discount price in July!
     [0.25112814]
```

Slika 4.13 Testni primjeri te rezultat

Iako je mreža uspjela razlikovati regularne i spam mailove, postotak od blizu 25% kojim je prepoznavala pogreške bio je nizak te stoga neprihvatljiv.

Sljedeća Tablica 4.1., prikazuje nekolicinu ostalih rezultata validacijskog seta podataka i točnost procjene istog modela, ali s manipulacijom hiperparametara.

Tablica 4.1. Rezultati prvih testiranja

Broj epoha	Broj uzoraka u epohi	Točnost treniranja	Točnost validacije	Točnost testiranja

12	256	0.9562	0.9561	0.16643623, 0.09225941, 0.19087997, 0.10783309, 0.16247934
15	256	0.9742	0.9695	0.12867874, 0.04559031, 0.15629417, 0.06403208, 0.12069222
20	256	0.9670	0.9578	0.05748004, 0.01734382, 0.07110634, 0.02773315, 0.05634502
15	512	0.8811	0.8834	0.11502412, 0.06855655, 0.12340584, 0.079319, 0.11181253
20	512	0.945	0.9327	0.11938, 0.06035, 0.13221, 0.07635, 0.11707

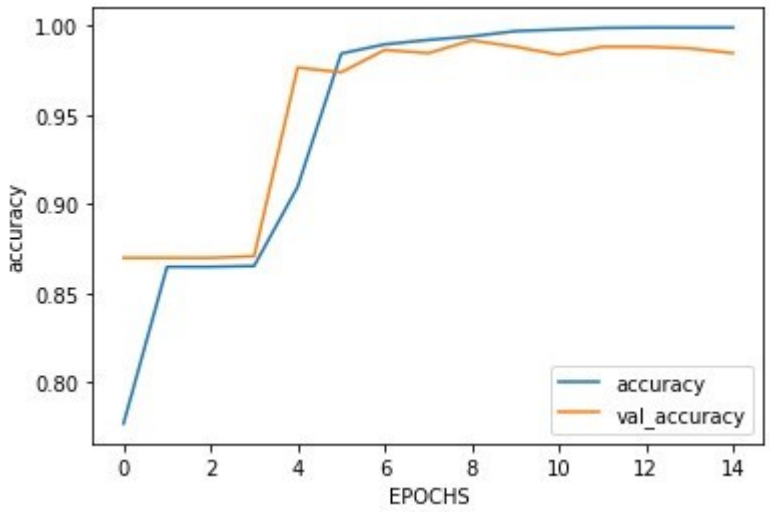
Iako su rezultati treniranja i validacije bili točni, nisu bili dovoljno izraženi te je kod klasifikacije puno regularnih mailova bilo prepoznato kao spam, što je nedopustivo.

Sljedeći korak je bila primjena *dropouta*, te LSTM arhitekture što je pokazalo velika poboljšanja. Neki od rezultata su vidljivi u tablici 4.2. u nastavku:

Tablica 4.2. Rezultati daljnjih testova

Broj epoha	Broj uzoraka u	Točnost	Točnost	Točnost

	epohi	treniranja	validacije	testiranja
12	256	0.9998	0.9857	0.89222, 0.00045, 0.99859, 0.00047, 0.47853
20	256	1.0000	0.9865	0.58471, 0.00024, 0.99951, 0.00027, 0.99793
15	512	0.9989	0.9883	0.99413, 0.00093, 0.99500, 0.00194, 0.71460
20	512	1.0000	0.9874	0.99808, 0.00040, 0.99811, 0.00044, 0.48591
Uz predefinirani *dropout*:				
15	512	0.9991	0.9848	0.9980185, 0.0013363, 0.9982241, 0.0013913, 0.9897939

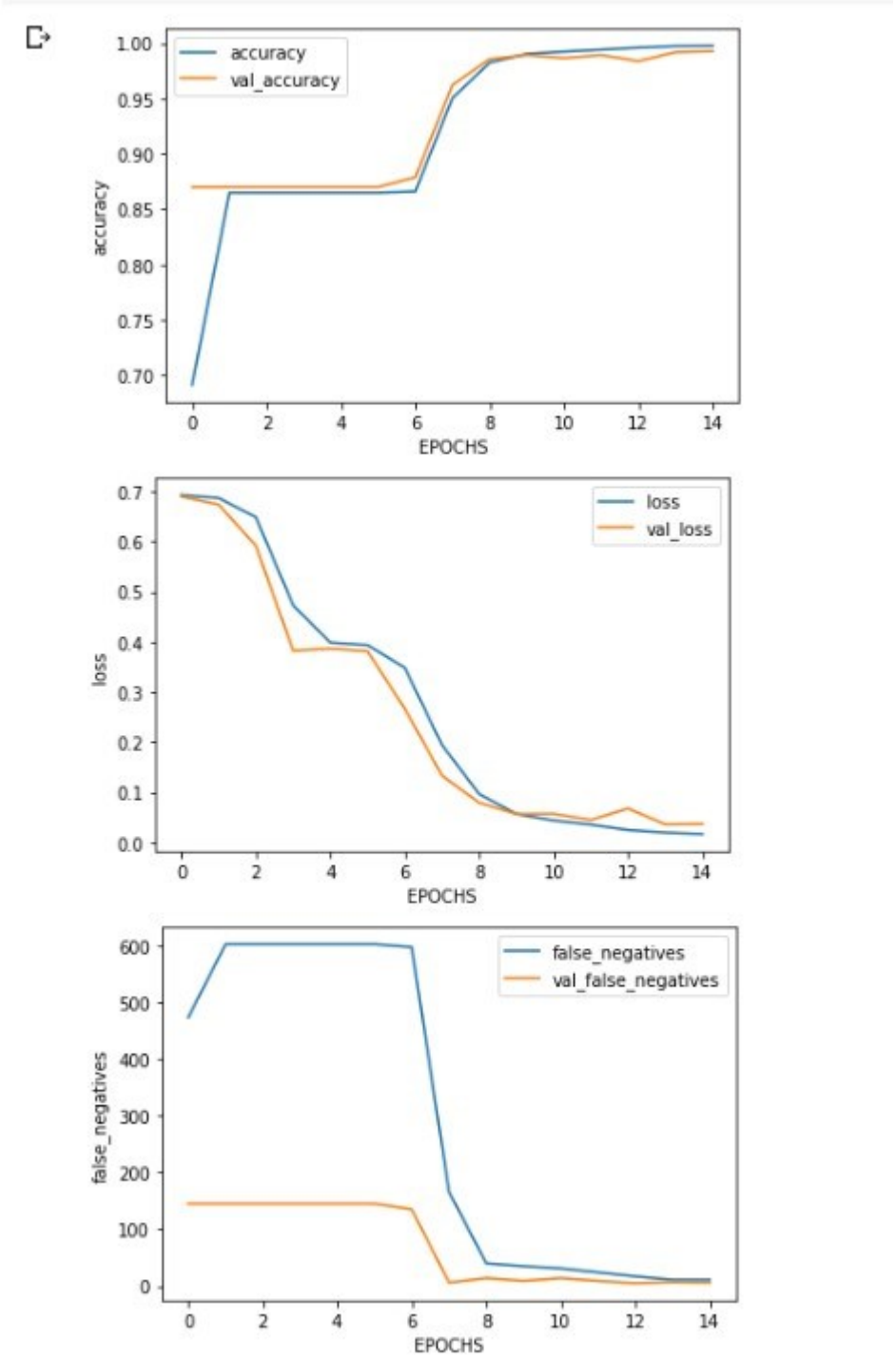


Slika 4.14. Točnost daljnjih rezultata

Na Slici 4.14 se vidi kako točnost mjerenja tijekom prolaska kroz skup podataka za trening (plava linija – „accuracy“) raste te ne pada na krajnjoj točki, što znači da ne dolazi do pretreniranja mreže, a što kod prolaska kroz validacijski skup nije situacija (žuta linija – „val_accuracy“), gdje je došlo do pretreniranja. Međutim, istaknuto je da korištenje točnosti kao jedini indeks uspješnosti nije dovoljna. Osim točnosti modela na testnim primjerima, neki od parametara koji su bitni prilikom zaključivanja uspješnosti modela su funkcija gubitka, te lažno pozitivni i lažno negativni rezultati.

Kad je *spam* poruka pogrešno klasificirana kao regularna pošta to stvara pomalo beznačajan problem, jer jedino što korisnik treba učiniti je izbrisati takvu poruku. Naprotiv, neugodno je kada komunikacija koja nije neželjena pošta bude pogrešno klasificirana kao neželjena pošto povećava rizik od gubitka važnih podataka zbog pogrešne klasifikacije filtra. Stoga je nedovoljno oslanjanje isključivo na točnost klasifikacije za procjenu učinkovitosti algoritama kao filtera neželjene pošte.

Kako bi klasifikacija bila što pouzdanija, uključeni su parametri koji osim točnosti pokazuju i gubitak te prepoznate lažno negativne poruke (Slika 4.15.). Lažno pozitivni uzorci iako pokazuju kvalitetu mreže, ne predstavljaju ozbiljan problem pa nisu prikazani.



Slika 4.15. Grafički prikaz indikatora kvalitete mreže

Kako je prikazano na slici iznad, na grafu točnosti najuspješnijeg modela, može se vidjeti da plava linija koja predstavlja trening set raste i ne pada, što implicira da ne dolazi do pretreniranja, a gdje se sada ni kod žute linije koja predstavlja validacijski set, ne pojavljuje pad na kraju validacije. Također vidimo drastično poboljšanje funkcije gubitka kroz epohe, zajedno s lažno negativnim prepoznatim uzorcima.

Gubitak najučinkovitijeg modela, lažno negativne procjene te točnost procjene može se vidjeti u Tablici 4.3:

Tablica 4.3. Gubitak, lažno negativni i lažno pozitivni rezultati

Redni broj epohe:	Gubitak:	Lažno negativni uzorci:	Točnost:
1/15	0.5160	145 uzoraka	0.8700
4/15	0.2478	145 uzoraka	0.8700
8/15	0.0746	34 uzorka	0.9677
11/15	0.0531	9 uzoraka	0.9874
13/15	0.0602	8 uzoraka	0.9883
15/15	0.0683	11 uzoraka	0.9857

Od 1000 uzoraka koji su pruženi za validaciju modela, broj od 8-11 uzoraka prepoznatih kao lažno negativnih, predstavlja postotak od samo 0.01% pogreške, što mrežu čini poprilično pouzdanom za primjenu.

Zaključak

Tehnologija u modernom svijetu napreduje iznimnom brzinom te se svakodnevno pojavljuju nove prilike koje olakšavaju život i rad svima na planeti, a posebno onima koji imaju pristup internetu. Uz sve pogodnosti i prednosti koje pristup internetu donosi čovječanstvu, tu se nalaze i problemi kojima se ovaj radi bavi. Radi se o korištenju jedne od najpoznatijih internetskih usluga, a to je elektronska pošta. Mnogobrojne reklame, ponude i prevare pokušavaju doći do svojih meta upravo pomoću e-pošte. Shodno tome, ne iznenađuje potreba za izradom ovog jednostavnog, ali iznimno korisnog rješenja za spomenuti problem. U sklopu ovog rada osmišljena je i izgrađena duboka neuronska mreža koja pomaže korisnicima prepoznati upravo tu neželjenu poštu. Proces detektiranja neželjene pošte u potpunosti obavlja mreža, a potrebno joj je samo predati sadržaj pošte. Nakon što je sadržaj predan, mreža daje rezultat iskazan u postocima koji definiraju kolika je vjerojatnost da je sadržaj prepoznat kao neželjen. Model je izrađen pomoću Python programskog jezika, uz korištenje TensorFlow i Keras biblioteka. Baza ulaznih podataka korištena je kao kolekcija neželjene i regularne pošte koja je unaprijed napisana. Neuronska mreža kreirana je i trenirana u sklopu Google programskog okruženja Colaboratory radi lakšeg rada sa samom mrežom. Nakon kreiranja, mreža je testirana na validacijskom setu podataka, a kasnije i na osobnim primjerima kako bi potvrdili njenu pouzdanost i uspješnost. Točnost koju mreža pokazuje kreće se blizu 99%, što je vidljivo i u tablicama rezultata (Tablica 4.3.). Budući je područje umjetne inteligencije poprilično veliko i kompleksno, uvijek postoji prostor za napredak, a mijenjajući kombinacije slojeva i parametara u različite strukture, postoji mogućnost da bi se taj zadatak mogao obavljati još uspješnije, kako je to učinio Google. Iako se radi o umjetnoj inteligenciji koja konstantno napreduje i uči, nažalost, ona nikada neće biti točna 100% te neće moći detektirati svaki neželjeni mail, jer se oni istovremeno prilagođavaju. Osim toga, složenije neuronske mreže mogu zahtijevati puno veću alokaciju resursa, a to potencijalno također može predstavljati problem.

Literatura

[1] S. Dixon, Apr 28, 2022, Daily spam volume worldwide 2020-2021: https://www.statista.com/statistics/1270424/daily-spam-volume-global/

[2] Gmail Spam Filter: When It Is Not Enough to Stop Spam: https://clean.email/gmail-spam-filter

[3] Christopher M. Bishop, Neural Networks for Pattern Recognition: http://people.sabanciuniv.edu/berrin/cs512/lectures/Book-Bishop-Neural%20Networks%20for%20Pattern%20Recognition.pdf

[4] McCulloch, W.S. and Pitts, W.H. (1943) A Logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics, 5, 115–133.

[5] Edutorij e-Škole, https://edutorij.e-skole.hr/share/proxy/alfresco-noauth/edutorij/api/proxy-guest/3b8a4b4e-84b0-4580-aa6f-e38efe028ed9/content/uploads/biologija-8/m03/j01/Biologija-8.-razred-3.-modul-1.-jedinica-1-1.jpg

[6] Douw Boshoff, The-structure-of-the-artificial-neuron: https://www.researchgate.net/profile/Douw-Boshoff/publication/328733599/figure/fig2/AS:734165188218886@1552050029499/The-structure-of-the-artificial-neuron.png

[7] Ajith Abraham, Artificial Neural Networks: http://www.softcomputing.net/ann_chapter.pdf

[8] Engin Pekel, Feed-forward-and-recurrent-ANN-architecture: https://www.researchgate.net/profile/Engin-Pekel/publication/315111480/figure/fig1/AS:472548166115333@1489675670530/Feed-forward-and-recurrent-ANN-architecture.png

[9] Hebb, D.O. (1949) The Organization of Behavior, John Wiley, New York.

[10] Yoichi Hayashi, Masateru Sakata and Stephen I. Gallant, Multi-Layer Versus Single-Layer Neural Networks and an Application to Reading Hand-Stamped Characters: https://www.researchgate.net/publication/312689257_Multi-Layer_Versus_Single-Layer_Neural_Networks_and_an_Application_to_Reading_Hand-Stamped_Characters

[11] Saurabh, Neural Network Tutorial – Multi Layer Perceptron: https://www.edureka.co/blog/neural-network-tutorial/

[12] Milos Miljanovic (Feb–Mar 2012). "Comparative analysis of Recurrent and Finite Impulse Response Neural Networks in Time Series Prediction", Indian Journal of Computer and Engineering. 3 : http://www.ijcse.com/docs/INDJCSE12-03-01-028.pdf

[13] Shaci Kaul, Mar 12,2020 Gradient Descent Problems and Solutions in Neural Networks: https://medium.com/analytics-vidhya/gradient-descent-problems-and-solutions-in-deep-learning-8002bbac09d5

[14] Chayakrit Krittanawong, Kipp Johnson, Robert S Rosenson, Deep learning for cardiovascularmedicine: A practical primer: https://miro.medium.com/max/807/1*hvUGQ7VkUGHuAjNqYLE_pQ.png

[15] Derrick Mwiti, Apr 22, Getting Started with Recurrent Neural Network (RNNs): https://towardsdatascience.com/getting-started-with-recurrent-neural-network-rnns-ad1791206412

[16] Andrej Karpathy, May 21, 2015, The Unreasonable Effectiveness of Recurrent Neural Networks: https://www.western-neuralnets.ca/week12/lab12.html

[17] Sepp Hochreiter, Jürgen Schmidhuber, LONG SHORT-TERM MEMORY: https://www.bioinf.jku.at/publications/older/2604.pdf

[18] Diego de Sousa Miranda, Python: a programming language for software integration and development J Mol Graph Model, 1999: https://www.academia.edu/1831560/Python_a_programming_language_for_software_integration_and_development?from=cover_page

[19] Peter Goldsborough, Oct 2016, A Tour of TensorFlow: https://arxiv.org/abs/1610.01178

[20] Antonio Gulli and Sujit Pal, Deep Learning with Keras: https://books.google.hr/books?hl=hr&lr=&id=20EwDwAAQBAJ&oi=fnd&pg=PP1&dq=keras&ots=lIdAcpdUX0&sig=tCiw7G7fvI-nMtqCPBSu3gcePl4&redir_esc=y#v=onepage&q=keras&f=false

[21] Google Colaboratory: https://research.google.com/colaboratory/faq.html

[22] Copyright 2015, Jupyter Team: https://docs.jupyter.org/en/latest/projects/architecture/content-architecture.html

[23] Emmanuel Gbenga Dada, Joseph Stephen Bassi, Haruna Chiroma, Shafi'i Muhammad Abdulhamid, Adebayo Olusola Adetunmbi, Opeyemi Emmanuel Ajibuwa, Machine learning for email spam filtering: review, approaches and open research problems ,Heliyon

Volume 5, Issue 6, June 2019, e01802 : https://www.sciencedirect.com/science/article/pii/S2405844018353404

[24] Tom Dietterich, September 1995, Overfitting and Undercomputing in Machine Learning: https://dl.acm.org/doi/pdf/10.1145/212094.212114

[25] Diederik P. Kingma, Jimmy Ba, 22 Dec 2014, Adam: A Method for Stochastic Optimization: https://arxiv.org/abs/1412.6980

[26] Sawan Saxena, listopad 3, 2020, Published in Analytics Vidhya, Understanding Embedding Layer in Keras: https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce

[27] Jeff Heaton, What are Embedding Layers in Keras (11.3): https://www.youtube.com/watch?v=OuNH5kT-aD0

[28] Christopher Olah, Understanding LSTM Networks, kolovoz 27,2015: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[29] Yugesh Verma, rujan 10, 2021 DEVELOPERS CORNER, A Complete Guide to LSTM Architecture and its Use in Text Classification: https://analyticsindiamag.com/a-complete-guide-to-lstm-architecture-and-its-use-in-text-classification/

[30] Tensorflow, 28. svibanj 2022., LSTM layer: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; 15(56):1929−1958, 2014., Dropout: A Simple Way to Prevent Neural Networks from Overfitting:

https://jmlr.org/papers/v15/srivastava14a.html

[32] Dropout: https://colab.research.google.com/github/jeffheaton/t81_558_deep_learning/blob/master/t81_558_class_05_4_dropout.ipynb

[33] Yugesh Verma, rujan 19, 2021 DEVELOPERS CORNER, A Complete Understanding of Dense Layers in Neural Networks: https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/

[34] Jason Brownlee on sječanj 18, 2021 in Deep Learning, How to Choose an Activation Function for Deep Learning: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

[35] Ihab S. Mohamed, September 2017, Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques: https://www.researchgate.net/publication/324165524_Detection_and_Tracking_of_Pallets_using_a_Laser_Rangefinder_and_Machine_Learning_Techniques

[36] Sagar Sharma, Sep 6, 2017, Activation Functions in Neural Networks: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[37] Campaign Monitor - Feb 27, 2020, Email Length Best Practices for Email Marketers and Email Newbies: https://www.campaignmonitor.com/blog/email-marketing/email-length-best-practices-for-email-marketers-and-email-newbies/

[38] Kaggle TEAM AI, Spam Text Message Classification: https://www.kaggle.com/datasets/team-ai/spam-text-message-classification

Sažetak

Problem svakodnevnog dobivanja neželjene pošte stvara puno neugodnosti. Približno 84% mailova svaki dan bude prepoznato kao nepoželjna pošta. Ovaj rad bavi se rješenjem tog problema. Rješenje je predstavljeno u obliku neuronske mreže koja je sposobna prepoznati i klasificirati potencijalnu nepoželjnu poštu. Kako bi se napravila mreža, korišteni su Python, TensorFlow, Keras, Google Colaboratory te Jupyter. Rezultati mreže su zadovoljavajući, iznose približno 99% te se mogu usporediti s velikim firmama poput Googlea, Yahooa i sličnih.
Ključne riječi: klasifikacija, neuronske mreže, neželjena pošta, strojno učenje

Summary

The problem of getting spam on a daily basis creates a lot of inconvenience. Approximately 84% of emails are recognized as spam each day. This paper deals with the solution of this problem. The solution is presented in the form of a neural network capable of identifying and classifying potential spam. Python, TensorFlow, Keras, Google Colaboratory and Jupyter were used to create the network. The results of the network are satisfactory, amounting to approximately 99% and can be compared to large companies such as Google, Yahoo !, and others.
Keywords: classification, neural networks, spam, machine learning