LSTM model za predikciju koncetracije peludi u zraku

Šarlija, Dino

Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet primijenjene matematike i informatike

Sveučilišni diplomski studij matematike
smjer: Matematika i računarstvo

LSTM model za predikciju koncentracije peludi u zraku

Diplomski rad

Mentor:

izv. prof. dr. sc.
Domagoj Matijević

Kandidat:

Dino Šarlija

Osijek, 2023

Sadržaj

1 | Uvod

U današnje vrijeme, sve veći broj ljudi suočava se s alergijskim reakcijama uzrokovanima peludom, što može znatno utjecati na njihovo zdravlje i kvalitetu života. Stoga je predviđanje koncentracije peludi postalo važnim zadatkom za istraživače kako bi omogućili pravovremeno reagiranje na alergijske simptome i poboljšali svakodnevni život osoba koje boluju od peludnih alergija. U posljednjih nekoliko godina, duboke neuronske mreže su doživjele značajan napredak i primjenu u različitim aspektima života. Jedan od tih modela dubokih neuronskih mreža je LSTM (Long Short-Term Memory), koji se koristi u različitim područjima, uključujući predviđanje vremena, kretanja dionica, rezultata sportskih događaja, ali i koncentracije peludi u zraku.

Cilj ovog rada je razviti i primijeniti LSTM model za predviđanje koncentracije peludi u zraku na određenom geografskom području. LSTM modeli posebno su prikladni za obradu vremenskih podataka koji su prikupljeni tijekom određenog vremenskog razdoblja. Upotrebom LSTM modela možemo stvoriti prognoze koncentracije peludi kako bismo pružili informacije osobama koje su osjetljive na pelud i omogućili im pripremu za nadolazeći dan.

U ovom radu ćemo razmotriti neuronske mreže, njihovu povijest, način funkcioniranja i komponente. Također ćemo istražiti različite vrste neuronskih mreža, s naglaskom na rekurentne neuronske mreže, te objasniti kako se razlikuju od ostalih tipova neuronskih mreža i koje su njihove prednosti. Na kraju ćemo pružiti analizu LSTM mreža i njihovih karakteristika, uz predstavljanje vlastitog razvijenog LSTM modela za predviđanje koncentracije peludi. Opisat ćemo proces prikupljanja i obrade podataka, treniranje i testiranje modela te usporediti rezultate prognoze ovisno o različitim parametrima modela. Na temelju tih rezultata izvući ćemo zaključke o kvaliteti predikcija.

2 | Neuronske mreže

U početku ovog istraživanja, važno je istaknuti da će se često koristiti izraz neuronske mreže, što se odnosi na umjetne neuronske mreže, a ne biološke neuronske mreže prisutne u ljudskom živčanom sustavu. Ovaj pojačani naglasak na umjetnim neuronskim mrežama ne isključuje postojanje veza između bioloških i umjetnih neuronskih mreža. U ljudskom živčanom sustavu, postoje specifične stanice poznate kao neuroni. Neuroni su međusobno povezani putem aksona i dendrita, a područja njihove povezanosti nazivaju se sinapsama.

Koncept neuronskih mreža počinje se pojavljivati između 1940-ih i 1960-ih pod imenom kibernetika [6]. U to vrijeme, neuronske mreže bile su jednostavni modeli temeljeni na linearnim funkcijama koje su koristile ulazne varijable i težine za generiranje brojčanih rezultata. Sljedeći važan razvojni trenutak za neuronske mreže dogodio se tijekom 1980-ih, kada su se počele pojavljivati po nazivom "konektivizam" [6]. Tada su počele razvijati značajke ulaznih podataka, dodatne vrijednosti koje opisuju ulazne varijable, te su se razvijali načini za bolju obradu podataka, kao što je povratna propagacija. S vremenom, tehnologija se razvijala, a neuronske mreže postajale sve prisutnije. Digitalizacija društva igrala je ključnu ulogu u njihovom razvoju, jer su računala postajala sve moćnija i omogućila pohranu i obradu velikih količina podataka u digitalnom obliku. Oko 2006. godine, kada su računala postala dovoljno snažna za obradu ogromnih količina podataka, počinje se koristiti izraz "duboke neuronske mreže" kako bi se označio treći ključni period u razvoju ovog područja.

Neuronske mreže su računalni modeli koji obrađuju ulazne podatke pomoću težina i slojeva neurona. Sličnost s biološkim neuronima nalazimo u načinu na koji ulazni podaci prolaze kroz mrežu, slično kao što impuls prolazi kroz dendrite i sinapse u biološkim neuronima. Ova arhitektura omogućava neuronskim mrežama da detektiraju složene obrasce i prilagode se različitim ulazima. Nakon što se obrade ulazne vrijednosti, neuronska mreža generira izlaznu vrijednost koja može biti rezultat klasifikacije, predviđanja ili drugih vrsta informacija. Impuls, nakon što prođe kroz dendrite i sinapse, nastavlja putovati do kraja aksona, što označava završetak procesa prijenosa signala.

2.1 Arhitektura neuronske mreže

Arhitekturu neuronske mreže ćemo objasniti kroz nekoliko primjera. Imat ćemo probleme koje ćemo pokušati riješiti primjenom neuronskih mreža. Treba naglasiti da ovi primjeri namjerno neće imati najbolje rješenje, jer ćemo se preko njih

upoznati s različitim elementima i pristupima.

Primjer 1. *Neka je zadan niz realnih brojeva brojeva (x_n). Prva četiri člana tog niza su 1, 3, 5 i 7. Treba napraviti model ili funkciju koja će predvidjeti sljedeći broj u nizu.*

Rješenje. U prvom pristupu koristit ćemo funkciju $f : \mathbb{R}^4 \to \mathbb{R}$ tako da je $f(x) = wx$, pri čemu je vektoru $x \in \mathbb{R}^4$. U ovom slučaju, vektor težina $w \in \mathbb{R}^4$ je zadan kao $w = [0.2, 0.4, 0.6, 0.8]$. Ulazna vrijednost za funkciju f je vektor x dok će izlazna vrijednost ili predikcija biti označena s $y = f(x)$. Sada ćemo detaljnije razmotriti skalarni umnožak između vektora w i x tako da ćemo element iz vektora x označiti s x_i, a elemnt i vektora w s w_i i tako za sve $i = 1, \ldots, 4$.

$$f(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

Nakon što uvrstimo vrijednosti i izračunamo rezultat, rezultat naše funkcije ili predikcija će biti broj 10, što nije točno jer vidimo da imamo niz neparnih prirodnih brojeva pa bi rezultat trebao biti 9. Vidimo da nam ovaj pristup nije dao točnu predikciju pa ćemo sada dodati još par elemenata s kojima možemo poboljšati preciznost. Neka je $b \in \mathbb{R}$ parametar pristranosti, a funkciju f ćemo od sada nazivati aktivacijskom funkcijom, a bit će to funkcija pod. Dodatno ćemo definirati vrijednost $z \in \mathbb{R}$, gdje je z rezultat umnoška težina i vektora x, uz dodatak parametra pristranosti.

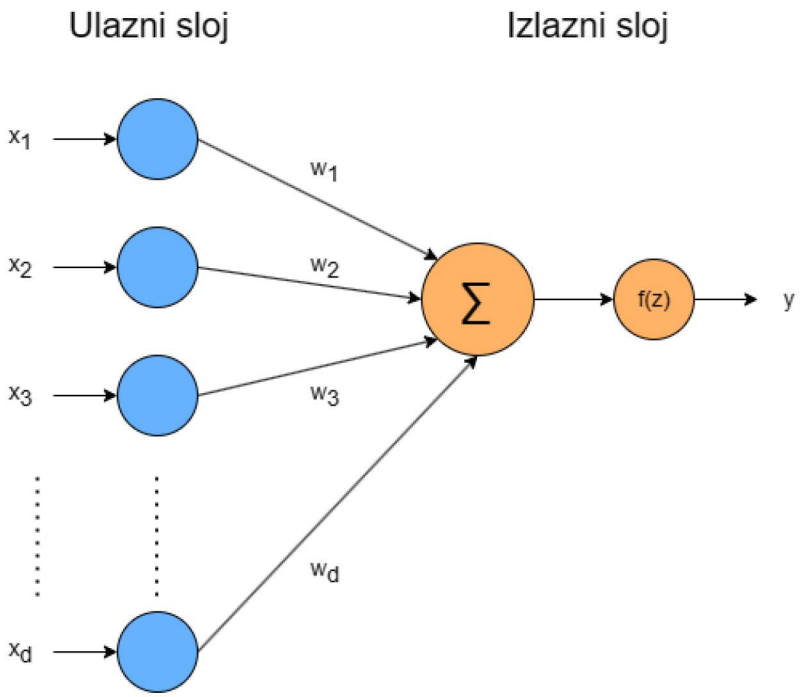
$$z = \sum_{i=1}^{n} w_i x_i + b \quad , \quad b = -0.1 \quad , \quad f(z) = \lfloor z \rfloor$$

Kad uvrstimo našave vrijednosti za x i w dobijemo da je $z = 9.9$, a onda je $f(z) = 9$.

U ovome primjeru smo se upoznali s osnovnim komponentama jednostavne neuronske mreže, a to su ulazne vrijednosti, težine, parametar pristranosti, aktivacijska funkcija i izlazne vrijednosti. Ovako jednostavna neuronska mreža može se nazivati jednoslojni perceptron, iako perceptron je algoritam koji se koristi u strojnom učenju za probleme binarne klasifikacije, te mu aktivacijska funkcija izgleda drugačije. Njegova konstrukcija odgovara modelu na slici 2.1, dok mu aktivacijska funkcija izgleda ovako

$$f(x) = \begin{cases} 1 & , x > 0 \\ 0 & , x \leq 0 \end{cases}$$

Perceptron ima jedan ulazni sloj koji sadrži $d \in \mathbb{N}$ čvorova koji prenose d značajki vektora $x \in \mathbb{R}^d, x = [x_1, \ldots, x_d]$ na izlazni čvor. Bridovi koji povezuju ulazne i izlazne čvorove sadrže težine iz vektora težina $w \in \mathbb{R}^d, w = [w_1, \ldots, w_d]$. Ulazni sloj sam po sebi ne obavlja računske operacije. U skrivenom sloju tada pomoću



Slika 2.1: Arhitektura perceptrona

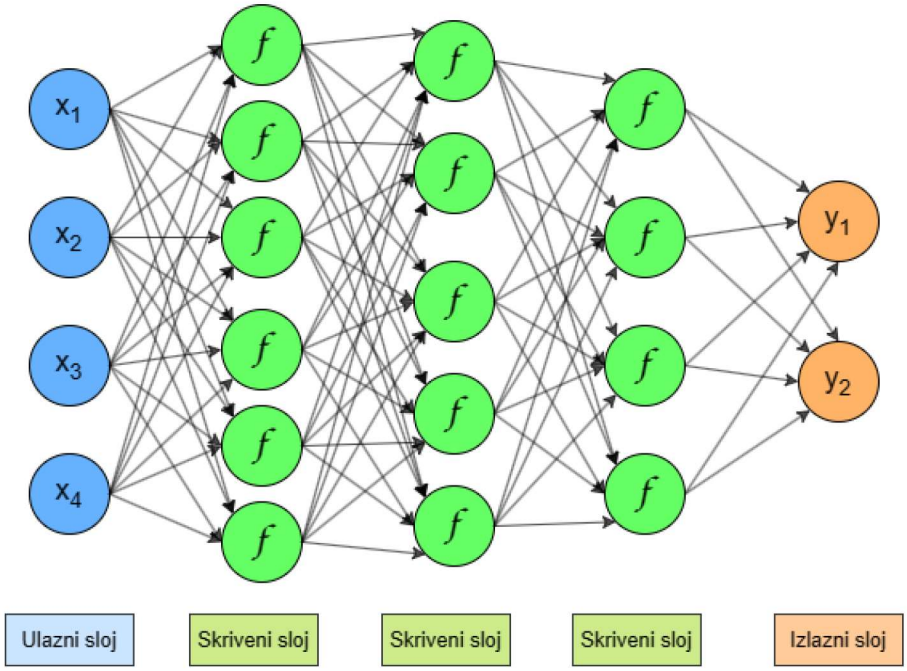
linearne funkcije $w^\intercal \cdot x = \sum_{i=1}^{d} w_i \cdot x_i$ rezultat koji dobijemo proslijeđujemo aktivacijskoj funkcija. Aktivacijska funkcija vraća predikciju $y \in \{0, 1\}$, gdje je $y = f(z)$, $z = \sum_{i=1}^{d} w_i \cdot x_i$.

Iako su pojedinačni neuroni moćniji od linearnih perceptrona, nisu dovoljno izražajni da riješe složene probleme učenja. Postoji razlog zašto naš mozak sastoji se od više od jednog neurona. Na primjer, nemoguće je da pojedinačni neuron razlikuje ručno pisane brojeve [4]. Dakle, kako bismo se nosili s mnogo složenijim zadacima, morat ćemo još dalje razvijati naš model strojnog učenja.

Osim jednoslojnih neuronskih mreža imamo još i višeslojne neuronske mreže poznate kao višeslojni perceptron ili duboke neuronske mreže. Višeslojne neuronske mreže sadrže više računalnih slojeva, dodatni slojevi (između ulaznog i izlaznog sloja) nazivaju se skrivenim slojevima jer izračuni koje obavljaju nisu vidljivi korisniku.

Specifična arhitektura višeslojnih neuronskih mreža naziva se unaprijedne mreže (feed-forward networks) jer se uzastopni slojevi međusobno povezuju u smjeru od ulaza prema izlazu. Zadana arhitektura unaprijednih mreža pretpostavlja da su svi čvorovi u jednom sloju povezani s onima u sljedećem sloju. Stoga je arhitektura neuronske mreže gotovo potpuno definirana nakon što su definirani broj slojeva i broj/vrsta čvorova u svakom sloju. Ne postoji povezanost između neurona u istom sloju, niti veze koje prenose podatke s trenutačnog sloja na prethodni sloj. Ovu mrežu možemo vidjeti na slici 2.2.

Poznavajući ove koncepte, možemo matematički konstruirati unaprijednu neuronsku mrežu. Neka je $x \in \mathbb{R}^d$, $x = [x_1, x_2, \ldots, x_d]$ vektor ulaznih vrijednosti s d značajki, gdje je $d \in \mathbb{N}$, a $y \in \mathbb{R}^m$, gdje je $y = [y_1, y_2, \ldots, y_m]$ vektor izlaznih



Slika 2.2: Arhitektura višeslojne neuronske mreže

vrijednosti, a $m \in \mathbb{N}$ broj izlaznih značajki.Matrice težina za i-ti skriveni sloj su označene s $W^{(i)} \in \mathbb{R}^{d_{i-1}} \times \mathbb{R}^{d_i}$ gdje su d_i dimenzije slojeva za $i = 1, \ldots, k$, gdje je $k \in \mathbb{N}$ broj skrivenih slojeva u neuronskoj mreži. Aktivacijske funkcije mogu se slično definirati po slojevima gdje za i-ti sloj imamo $f^{(i)}$. Svaki sloj može imati zasebnu aktivacijsku funkciju dok u praksi se najčešće koriste dvije aktivacijske funkcije, jedna za sva skrivena stanja i jedna za izlazni sloj. Slično ćemo definirati vektor pristranosti $b^{(i)} \in \mathbb{R}^{d_i}$, gdje je $b^{(i)}$ vektor pristranosti u i-tom sloju s dimenzijom d_i. Dodatno možemo definirati vektor $h^{(i)}$, koji će sadržavati vrijednost svih neurona nakon primjene aktivacijske funkcije u i-tom sloju. Tada arhitekturu neuronske mreže možemo prikazati kao

$$\begin{aligned} h^{(1)} &= f^{(1)}((W^{(1)})^\intercal x + b^{(1)}) \\ h^{(2)} &= f^{(2)}((W^{(2)})^\intercal h^{(1)} + b^{(2)}) \\ &\vdots = \ \vdots \\ h^{(k)} &= f^{(k)}((W^{(k)})^\intercal h^{(k-1)} + b^{(k)}) \\ y &= f^{(k+1)}((W^{(k+1)})^\intercal h^{(k)} + b^{(k+1)}) \end{aligned}$$

pri čemu nam $\intercal$ predstavlja operaciju transponiranja matrice. Sada ćemo kroz primjer pokazati proces kako neuronska mreža računa vrijednosti dok ne dođe do riješenja.

Primjer 2. *Zamislimo da se bavimo uzgojem cvijeća te imamo pokretnu traku pomoću koje želimo razvrstati cvijeće koje uberemo. Uzgajamo dvije vrste tulipana, tupilane Fosteriana i tulipane Triumph te želmio automatizirazi proces razvrstavanja tako da imamo uređaj koji će moći prepoznati svaki cvijet po njegovim specifičnostima i točno ga razvrstati. Karakteristike po kojima se ova dva cijeta razlikuju su duljina lista, širina lista, duljina latice,*

širina latice i duljina stabljike. Napravimo neuronsku mrežu koja će nam pomoći riješiti ovaj problem.

Rješenje. Prvo trebamo primijetiti da imamo klasifikacijski problem, to jest trebamo naučiti naš model prepoznati koji cvijet pripada kojoj vrsti. Idemo dodati numeričku vrijednost našim cvjetovima tako da Tupilan Fosteriana bude reprezentiran s vrijednošću 0, a Tulipan Triumph bude reprezentiran s vrijednošću 1. Nakon što smo odredili klase(0,1) za naš problem trebamo napraviti konstrukciju naše neuronske mreže. Za cvjetove znamo da imaju 5 značajki, a one su duljina i širina lista, duljina i širina latice te duljina stabljike pa nam onda ulazni sloj može sadržavati 5 neurona ili imati dimenziju 5. Model će imati i dva skrivena sloja pri čemu će prvi skriveni sloj imati dimenziju 5, a drugi dimenziju 4 pri čemu će svaki neuron primiti vrijednost iz svih pet neurona iz prethodnog sloja. Izlazni sloj će imati dimenziju 2 pri čemu podatci koji će se nalaziti tamo biti predstavljeni kao vjerojatnosna distribucija između naša dva cvijeta to jest ako se broj nalazi na 0. poziciji predstavlja vjerojatnost da je on Forsterian, a ako se nalazi na 1. poziciji onda je to vjerojatnost da je Triumph. Ovisno koja vjerojatnost bude veća naš će model odabrati tu vrstu tulipana.

Neka je $x \in \mathbb{R}^5$, $x = [0.2, 0.4, 0.6, 0.8, 1.0]$ i neka su to svojstva tulipana Triumpha, sljedeće što moramo imati su matrice težina, a one su $W_1 \in \mathbb{R}^{5x5}$, $W_2 \in \mathbb{R}^{5x4}$ i $W_3 \in \mathbb{R}^{4x2}$.

$$W_1 = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 \\ 0.6 & 0.7 & 0.8 & 0.9 & 1.0 \\ 1.1 & 1.2 & 1.3 & 1.4 & 1.5 \\ 1.6 & 1.7 & 1.8 & 1.9 & 2.0 \\ 2.1 & 2.2 & 2.3 & 2.4 & 2.5 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \\ 0.5 & 0.6 & 0.7 & 0.8 \\ 0.9 & 1.0 & 1.1 & 1.2 \\ 1.3 & 1.4 & 1.5 & 1.6 \\ 1.7 & 1.8 & 1.9 & 2.0 \end{bmatrix}, \quad W_3 = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \\ 0.7 & 0.8 \end{bmatrix}$$

U ovome primjeru ćemo koristiti ReLU ($f_1(x)$) i softmax ($f_2(x)$) aktivacijsku funkciju tako da će se ReLU primijeniti na skrivenim slojevima, a softmax na izlaznom sloju. Još ćemo dodati vektore pristranosti $b_1 \in \mathbb{R}^5$, $b_2 \in \mathbb{R}^4$ i $b_3 \in \mathbb{R}^2$.

$$b_1 = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \\ 0.5 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix}, \quad b_3 = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

Kako bi mogli pratit postupak računanja, dodatno ćemo definirati vektore $z_1 \in \mathbb{R}^5$, $z_2 \in \mathbb{R}^4$ i $z_3 \in \mathbb{R}^2$ koji će predstavljati rezultat nakon dodavanja težina i vektora pristranosti te vektore $h_1 \in \mathbb{R}^5$, $h_2 \in \mathbb{R}^4$ koji će biti rezultat primjene aktivacijske funkcije i $y \in \mathbb{R}^2$ kao konačni rezultat u izlaznom sloju. Idemo proći kroz korake našeg model

$$z_1 = W_1^\intercal x + b_1, \quad h_1 = f_1(z_1)$$

$$z_1 = \begin{bmatrix} 3.05 \\ 3.65 \\ 4.25 \\ 4.85 \\ 5.45 \end{bmatrix}, \quad h_1 = \begin{bmatrix} 3.05 \\ 3.65 \\ 4.25 \\ 4.85 \\ 5.45 \end{bmatrix}$$

Nakon što smo pomnožili ulazne vrijednosti i težine, dodali vektor pristranosti i primjenili aktivacijsku funkciju dobivamo vrijednosti za prvi skriveni sloj. Ovaj postupak ponovimo još dva puta i dobivamo

$$z_2 = W_2^\intercal h_1 + b_2, \quad h_2 = f_1(z_2), \quad z_3 = W_3^\intercal h_2 + b_3, \quad y = f_2(z_3)$$

$$z_2 = \begin{bmatrix} 26.21 \\ 31.02 \\ 35.83 \\ 40.64 \end{bmatrix}, \quad h_2 = \begin{bmatrix} 26.21 \\ 31.02 \\ 35.83 \\ 40.64 \end{bmatrix}, \quad z_3 = \begin{bmatrix} 14.29 \\ 19.14 \end{bmatrix}, \quad y = \begin{bmatrix} 0.085 \\ 0.915 \end{bmatrix}$$

Rezultat koji smo dobili za y pokazuje vjerojatnost da je naš tulipan Fosteriana 0.085, dok vjerojatnost da je naš tulipan Triumph je 0.915. Vidimo da je naš model dobro prepoznao cvijet.

Iz ovog primjera vidimo da je arhitektura neuronske mreže gotovo potpuno definirana nakon što su definirani broj slojeva i broj neurona u svakom sloju. Dodatno bi se mogli zapitati što ako proslijedimo našem modelu neke druge ulazne parametre, a predikcija ne bude točna. Ostao nam jedan detalj o kojem do sada nismo pričali, a to je funkcija gubitka (loss function) koja govori koliko je naša izlazna vrijednost blizu ili daleko od prave vrijednosti. Sada kad znamo za ovakvu funkciju bilo bi dobro kada bi mogli iskoristiti taj podatak i promijeniti određene parametre u našoj neuronskoj mreži. Do sada smo mogli primijetiti iz samog imena feed forward neuronskih mreža da se veze kreću samo prema naprijed, ali to ne mora biti tako.

2.2 Aktivacijske funkcije

Aktivacijske funkcije su važne komponente neuronskih mreža koje aproksimiraju naša rješenja. Pomoću njih naše mreže uče i povezuju kompleksne odnose između podataka. Različiti izbori aktivacijskih funkcija mogu se koristiti za simuliranje različitih tipova modela koji se koriste u neuronskim mrežama. Izbor aktivacijske funkcije ključan je dio dizajna neuronske mreže. Kad govorimo o aktivacijskim funkcijama, često se pored njih pojavljuju izrazi "linearna" ili "nelinearna

aktivacijska funkcija". Već smo se upoznali s oba ova tipa aktivacijskih funkcija. U primjeru 1, kada smo računali umnožak ulaznih vrijednosti i vektora težina, u početku nismo koristili aktivacijsku funkciju. Mogli smo koristiti linearnu aktivacijsku funkciju tako da bismo i dalje dobili isti rezultat, ako bismo za aktivacijsku funkciju uzeli funkciju identiteta koja izgleda:

$$f(x) = x$$

jer ona kao rezultat vraća isti element koji smo joj proslijedili. Graf ove funkcije možemo vidjeti na slici 2.3. Budući da je funkcija linearna, nije u mogućnosti modelirati kompleksne odnose u podacima. Kao rezultat toga, neuronske mreže koje koriste samo linearne aktivacijske funkcije bile bi ograničene, što ih čini nesposobnima učinkovito rješavati nelinearne probleme. Stoga se ovakve aktivacijske funkcije često koriste u izlaznom sloju.

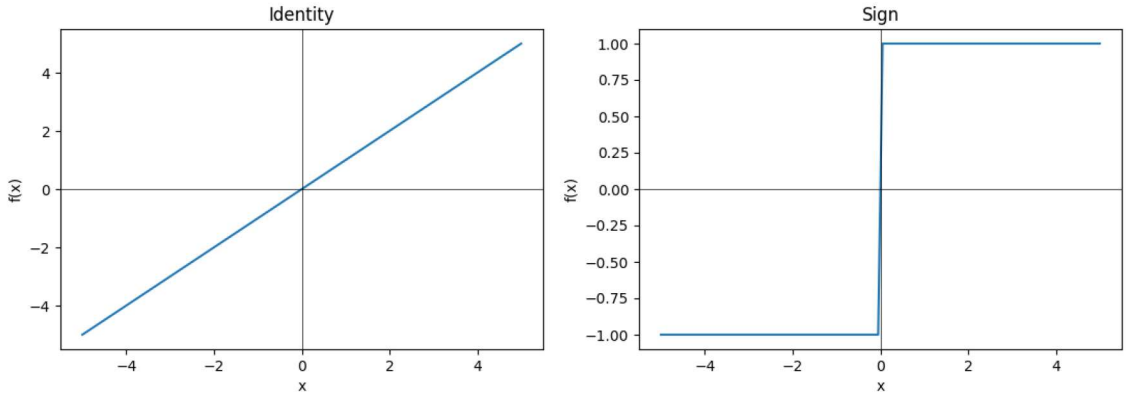
Osim linearnih, imamo i nelinearne aktivacijske funkcije koje imaju veću zastupljenost u neuronskim mrežama. Većina stvarnih podataka i zadataka ima nelinearan karakter, stoga aktivacijske funkcije uvode nelinearnost, omogućavajući neuronskim mrežama da nauče i aproksimiraju kompleksne funkcije, što im omogućava učinkovito rješavanje šireg spektra problema. Ova sposobnost hvatanja složenih značajki ključna je za zadatke poput prepoznavanja slika, obrade prirodnog jezika i drugih izazovnih problema u strojnom učenju. Jednu takvu funkciju smo spomenuli kod perceptrona, a to je ***sign*** funkcija koja izgleda:

$$f(x) = \begin{cases} 1 & , x \geq 0 \\ -1 & , x < 0 \end{cases}$$

a graf funkcije možemo vidjeti na slici 2.3. Ova funkcija se može koristiti za klasifikacijske probleme koji imaju dvije moguće klase. U usporedbi s ostalim nelinearnim aktivacijskim funkcijama ***sign*** funkcija se ne koristi toliko često zbog problema nestajućih gradijenata, o kojem ćemo detaljnije raspravljati u kasnijem poglavlju. Dok se funkcija ***sign*** može koristiti za predviđanje binarne klase, postoji problem kada tijekom treniranja trebamo ažurirati težine jer nam za to treba derivacija funkcije, a kako je derivacija ***sign*** funkcije 0, ne možemo ažurirati težine. Dok se funkcija ***sign*** može koristiti za predviđanje binarne klase, njezin dio koji nije diferencijabilan sprječava njenu upotrebu za stvaranje funkcije gubitka tijekom treniranja.

Sigmoidna aktivacijska funkcija daje vrijednosti u rasponu $(0, 1)$, što je korisno za izvođenje računanja koja se trebaju interpretirati kao vjerojatnosti. Intuitivno, ovo znači da kada je vrijednost neurona jako mala, izlaz ***sigmoid*** funkcije je vrlo blizu 0. Kada je vrijednost neurona jako velika, izlaz ***sigmoid*** fukcije je blizu 1. Između ovih dvaju ekstrema, funkcija poprima oblik slova S, što vidimo u definiciji i na slici 2.4.

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



(a) Identity (b) Sign

Slika 2.3: Grafovi funkcije identiteta (a) i funkcije sign (b)

Sigmoidna funkcija je glatka i diferencijabilna posvuda, što olakšava optimizaciju temeljenu na gradijentu tijekom procesa treniranja. Jedna od mana ***sigmoidne*** funkcije je što njezini izlazi su ograničeni između 0 i 1. Za vrlo pozitivne ili vrlo negativne ulazne vrijednosti, gradijent ***sigmoidne*** funkcije približava se nuli, što može uzrokovati problem nestajućeg gradijenta tijekom postupka povratne propagacije (backpropagation). Zbog toga se više koristi u izlaznim slojevima nego u skrivenim slojevima.

Funkcija hiperbolnog tangensa (***tanh***) ima oblik sličan sigmoidnoj funkciji, osim što je vertikalno proširena u raspon $[-1, 1]$. Funkcija hiperbolnog tangensa je poželjna u odnosu na ***sigmoidnu*** kada želimo da izlazne vrijednosti budu i pozitivne i negativne, što mozežemo vidjeti na slici 2.4.

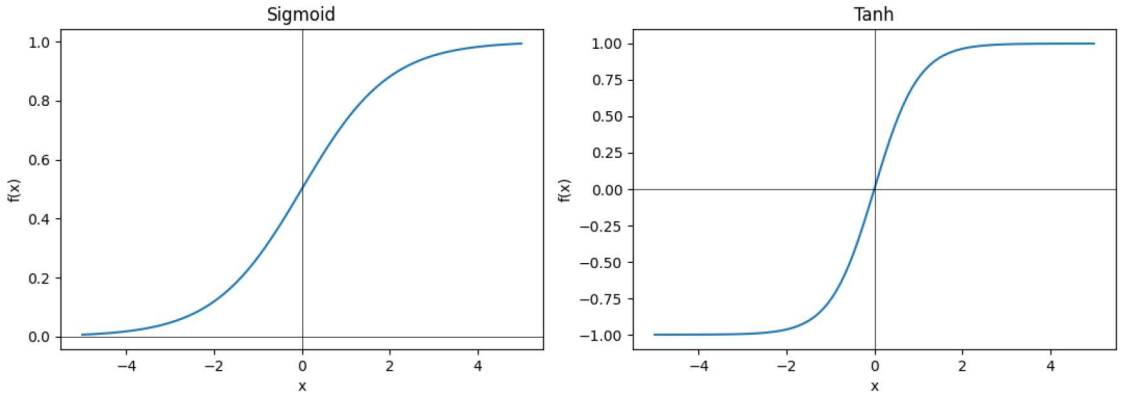
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Osim toga, njezina derivacija je simetrična oko nule što smanjuje problem nestajućeg gradijenta. Zbog toga je pogodnija za učenje dubokih neuronskih mreža. Također i hiperbolni tanges može imati problema s padajućim gradijenotm ako su ulazne vrijednosti jako velike ili jako male.

Polako vidimo da problem nestajućih gradijenata postaje sve veći problem za naše modele pa je tako nastala ispravljena linearna jedinica kao alternativa za rješavanje tog problema, njezin graf možemo vidjeti na slici 2.5. Ispravljeni linearni članovi (Rectified Linear Units - ReLU) koriste aktivacijsku funkciju

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

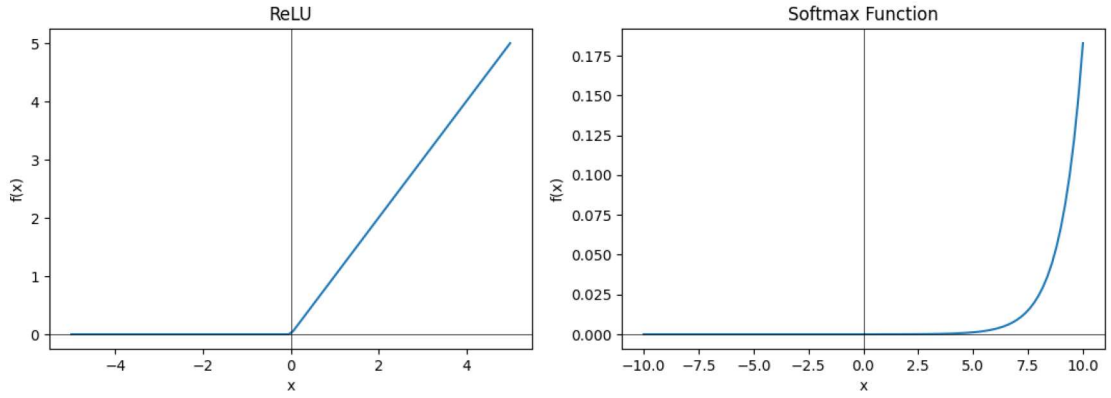
Ova aktivacijska funkcija je jednostavna za optimizaciju jer je vrlo slična linearnim funkcijama. Jedina razlika između linearnih funkcija i ReLU funkcije je što ReLU funkcija za izlazne vrijednosti ima nulu ako su ulazne vrijednosti negativne (nelinearni dio), a u suprotnom rezulta bude jednak ulaznim vrijednostima (linearni



(a) Sigmoid (b) Tanh

Slika 2.4: Grafovi funkcije sigmoid (a) i funkcije hiperbolnog tangensa (b)

dio). Ova aktivacijska funkcija ima manje problema s nestajućim gradijentima i trenira se brže zbog efikasnijeg računanja gradijenta u usporedbi sa ***sigmoidnom*** i ***tanh*** funkcijom [1]. Međutim, može se dogoditi da određeni neuroni "umru" kada vrijednost u neuronima bude 0 i ne može se više promijeniti s nule. Ovdje gradijenti mogu još eksplodirati jer je derivacija funkcije jednaka 1, kada je $x > 0$ pa kada imamo veliku težinu, ne možemo je značajno smanjiti.



(a) ReLU (b) Softmax

Slika 2.5: Grafovi funkcije ReLU (a) i funkcije softmax (b)

Za kraj smo ostavili softmax aktivacijsku funkciju koja se najčešće koristi u klasifikacijskim problemima u izlaznom sloju, a graf funkcije možemo vidjeti na slici 2.5. Ona se može promatrati kao generalizirana verzija sigmoidne funkcije, samo što nju koristimo za predstavljanje vjerojatnostne distribucije K varijabli. Softmax aktivacijska funkcija izgleda

$$f_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

pri čemu je $K \in \mathbb{N}$ broj klasa koje želimo predvidjeti to jest broj vjerojatnosti u vjerojatnosnoj distribuciji. Kao i sigmoidna funkcija, softmax aktivacija može imati problema kada je njen ulaz izuzetno negativan ili izuzetno pozitivan. Kako je softmax funkcija diferencijabilna nema problema s gradijentima pri povratnoj propagaciji.

2.3 Funkcije gubitka

Nakon što smo dizajnirali naš model, odabrali sve dodatne parametre i aktivacijske funkcije te počeli trenirati, često možemo naići na problem da rezultat koji nam naš model izračuna ne bude točan ili precizan kao prava vrijednost koju smo trebali dobiti. Kada dobijemo netočan ili neprecizan rezultat, često nas zanima gdje smo pogriješili, ali to nije lako otkriti, stoga postoji proces kojim možemo podesiti naše parametre kako bismo dobili bolji rezultat. Prvo trebamo izračunati koliko smo pogriješili, a tu nam pomaže funkcija gubitka.

Tijekom treniranja, naši ulazni podatci prolaze kroz neuronsku mrežu i završavaju u izlaznom sloju mreže. Rezultat koji dobijemo u izlaznom sloju je rješenje našeg problema, koje ćemo označiti s $y_{rezultat}$, osim rješenja imamo i originalnu vrijednost za iste ulazne parametre, koju ćemo označiti s $y_{original}$. Te dvije vrijednosti $y_{rezultat}$ i $y_{original}$ prosljeđujemo našoj funkciji gubitka, koju odabiremo tako da najbolje odgovara našem problemu koji rješavamo. Cilj nam je dobiti što manju vrijednost funkcije gubitka, tj. da nam rezultat funkcije bude što bliže nuli.

U slučaju da pokušavamo riješiti problem klasifikacije možemo koristiti unakrsnu entropiju čija bi formula izgleda:

$$H(p,q) = -\sum_{x \in X} p(x) \log q(x)$$

pri čemu je $p(x)$ prava distribucija, a $q(x)$ predviđena distribucija te X skup klasa. Ako bismo uzeli primjer problema binarne klasifikacije onda bi funkcija mogla izgledati:

$$H(p,q) = -(y_{org} \cdot \log y_{pred} + (1 - y_{org}) \cdot \log(1 - y_{pred}))$$

gdje je y_{pred} vjerojatnost predikcije (prve klase), a y_{org} je originalna vrijednost koja može biti ili 0 ili 1. Pored unakrsne entropije postoji još dosta funkcija gubitka koje moežmo koristiti kada imamo klasifikacijski problem, a jedna od njih je i hinge loss koja izgleda:

$$l(y_{pred}) = \max(0, 1 - y_{org} \cdot y_{pred})$$

gdje je $y_{org} \in \{-1, 1\}$, a y_{pred} je izlazna vrijednost ili predikcija rezultat. Osim funkcija gubitka za klasifikacijske probleme, postoje i druge funkcije koje se mogu primjeniti za regresijske probleme od kojih ćemo pokazati njih dvije. Prva funkcija gubitka koje ćemo spomeniti je Mean Squared Error (MSE) Loss čija funckija izlgleda

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^{n} (y_{pred}^{(i)} - y_{org}^{(i)})^2$$

gdje je y_{org} ciljana vrijednsot, a y_{pred} je vrijednost koju je naš model predvidio, pri čemu n označava broj rezultata koje ćemo imati. Druga funkcija gubitka koju ćemo pokazati je Mean Absolute Error (MAE)

$$MAE = \frac{1}{n} \cdot \sum_{i=1}^{n} |y_{pred}^{(i)} - y_{org}^{(i)}|$$

Vidimo da je MAE sličan MSE, samo što ne kvadriramo razliku između y_{pred} i y_{org} nego primjenjujemo apsolutnu vrijednost. Kod ovih funkcija u većini slučajeva želimo dobiti rezultat što bliže nuli ili nulu tamo gdje je to moguće, ali ne smijemo zaboraviti da ovisno o modelu i problemu koji pokušavamo riješiti ne moramo uvijet tražiti takav rezultat.

Sada kada smo se upoznali s funkcijama gubitka možemo reći gdje su nam korisne. Osim što nam govore koliko je vrijednost u izlaznom sloju blizu ili daleko od prave vrijednosti one nam služe kao orijentir fino podešavanje našeg modela. Znamo da prije nego što počnemo trenirati neuronsku mrežu trebamo odabrati broj epoha tj. koliko će puta naš model obraditi iste podatke. Nema smisla raditi iste računske operacije s istim brojevima, jer će rezultat i dalje biti isti. Zato nakon svake epohe naš model optimizira svoje težine i vektore pristranosti u svim slojevima. Nakon što se izračuna funkcija gubitka onda možemo računati njezin gradijent koji možemo primijeniti na težine i vektore pristranosti kako bi ih ažurirali. Ovaj proces se zove gradijentni spust. Zbog gradijentnog spusta izbor aktivacijskih funkcija i funkcija gubitka je važan, jer ako su funkcije derivabilne, lakše i brže ćemo ih moći obraditi. Ovaj proces optimizacije mreže naziva se povratna propagacija, o kojoj ćemo više reći u sljedećem poglavlju.

2.4 Povratna propagacija

Treniranje neuronske mreže obično uključuje prilagodbu njenih parametara putem optimizacije gradijentnog spusta, koja minimizira zadanu funkciju gubitka koja procjenjuje točnost mreže u obavljanju željenog zadatka. Neki poznati primjeri algoritama gradijentnog spusta su Batch Gradient Descent, Stochastic Gradient Descent i Mini-Batch Gradient Descent. Kako bismo mogli računati gradijentni spust potreban nam je novi hiperparametar kojeg do sada nismo spominjali,

a to je stopa učenja $\alpha \in \mathbb{R}^+$. Stopa učenja je broj koji opisuje za koliko ćemo napraviti pomak kako bi naš rezultat što brže konvergirao. Ažuriranje težina možemo napraviti na sljedeći način:

$$W^{(i+1)} = W^{(i)} - \alpha \nabla l(W^{(i)})$$

gdje je $W^{(i)}$ matrica težina u i-toj iteraciji, $i \in \mathbb{N}$ označava broj iteracije u algoritmu gradijentnog spusta. $\nabla l(W^{(i)})$ označava gradijent funkcije gubitka u i-toj iteraciji gradijentnog spusta. Ovisno o vrsti gradijentnog sputa ovaj dio $\nabla l(W^{(i)})$ se računa na različite načine. Gradijentni spust je široko korišten u praksi za treniranje neuronskih mreža jer je vrlo učinkovit i dobro se skalira za velike skupove podataka i složene modele. U povratnoj propagaciji krećemo se s desna na lijevo, tj. od izlazne vrijednosti prema skrivenim stanjima pa sve do ulaznih vrijednosti. Sad možemo opisati kako radi povratna propagacija.

Primjer 3. *Imamo neuronsku mrežu 2.6 u kojoj je svaki neuron povezan sa svim neuronima iz sljedećeg sloja. U ulaznom sloju ima tri neurona koja ćemo zapisati kao* $x_1 = 0.8, x_2 = 0.6, x_3 = 0.4$*, imamo dva skrivena sloja s dva neurona koje ćemo zapisati kao* $h_1^{(1)}, h_2^{(1)}, h_1^{(2)}, h_2^{(2)} \in \mathbb{R}$ *pri čemu su* $h_1^{(1)}$ *i* $h_2^{(1)}$ *vrijednosti neurona u prvom skrivenom sloju, a* $h_1^{(2)}, h_2^{(2)}$ *vrijednosti neurona u drugom skrivenom sloju te za kraj imamo izlazno sloj s jednim neuronom kojeg ćemo zapisati kao* $y_{pred} \in \mathbb{R}$ *pri ćemo će naša originalna vrijednost bit* $y_{org} = 3$*. Za aktivacijsku funkciju* f *ćemo odabrati ReLU funkciju i nju ćemo primijeniti na skrivene slojeve dok u izlaznom sloju nećemo imati aktivacijsku funkciju (ili možemo reći da koristimo funkciju identiteta). Ostaje nam još funkcija gubitka* l*, a ona će biti Mean Squared Error te ćemo za stopu učenja uzeli da je* $\alpha = 0.1$*. Vrijednosti težina će biti* $W_1^{(1)} = 0.1, W_2^{(1)} = 0.2, W_3^{(1)} = 0.3, W_4^{(1)} = 0.4, W_5^{(1)} = 0.5, W_6^{(1)} = 0.6, W_1^{(2)} = 0.7, W_2^{(2)} = 0.8, W_3^{(2)} = 0.9, W_4^{(2)} = 1.0, W_1^{(3)} = 1.1, W_2^{(3)} = 1.2$*,, a parametri pristranosti* $b_1^{(1)} = 0.1, b_2^{(1)} = 0.2, b_1^{(2)} = 0.3, b_2^{(2)} = 0.4, b^{(3)} = 0.5$

Rješenje. Kao što smo to radili i u prethodnim primjerima, sada ćemo izvršiti unaprijednu propagaciju. Prvo ćemo izračunati vrijednosti za prvi skriveni sloj

$$\begin{aligned} z_1^{(1)} &= W_1^{(1)} \cdot x_1 + W_2^{(1)} \cdot x_2 + W_3^{(1)} \cdot x_3 + b_1^{(1)} \\ &= 0.42 \end{aligned}$$

$$\begin{aligned} z_2^{(1)} &= W_4^{(1)} \cdot x_1 + W_5^{(1)} \cdot x_2 + W_6^{(1)} \cdot x_3 + b_2^{(1)} \\ &= 1.06 \end{aligned}$$

$$\begin{aligned} h_1^{(1)} &= max(0, z_1^{(1)}) = 0.42 \\ h_2^{(1)} &= max(0, z_2^{(1)}) = 1.06 \end{aligned}$$

Sada ćemo samo nastaviti za drugi skriveni sloj

$$\begin{aligned} z_1^{(2)} &= W_1^{(2)} \cdot h_1^{(1)} + W_2^{(2)} \cdot h_2^{(1)} + b_1^{(2)} \\ &= 1.442 \end{aligned}$$

$$\begin{aligned} z_2^{(2)} &= W_3^{(2)} \cdot h_1^{(1)} + W_4^{(2)} \cdot h_2^{(1)} + b_2^{(2)} \\ &= 1.838 \end{aligned}$$

$$\begin{aligned} h_1^{(2)} &= max(0, z_1^{(2)}) = 1.442 \\ h_2^{(2)} &= max(0, z_2^{(2)}) = 1.838 \end{aligned}$$

Na kraju računamo izlazni sloj

$$\begin{aligned} z^{(3)} &= W_1^{(3)} \cdot h_1^{(2)} + W_2^{(3)} \cdot h_2^{(2)} + b^{(3)} \\ &= 4.2918 \\ y_{pred} &= z^{(3)} = 4.2918 \end{aligned}$$

S ovime smo završili unaprijednu propagaciju i sada počinjemo s povratnom propagacijom tako da ćemo prvo računati vrijednost funkcije gubitka, a zatim vrijednost gradijenta funkcije gubitka koju ćemo označiti s $\delta^{(i)}$

$$\begin{aligned} l(y_{pred}) &= \frac{1}{2}(y_{org} - y_{pred})^2 \\ &= 0.8344 \\ \delta_3 &= \nabla l(y_{pred}) = 1.2918 \end{aligned}$$

Kako bi mogli dalje računati gradijente bit će nam potrebna derivacija aktivacijske funckije, a ona je

$$f'(x) = \begin{cases} 0 & , x < 0 \\ 1 & , x > 0 \end{cases}$$

Sada treba izračunati ostale gradijente

$$\begin{aligned} \delta_1^{(2)} &= (\delta^{(3)} \cdot W_1^{(3)}) \cdot f'(h_1^{(2)}) = 1.4210 \\ \delta_2^{(2)} &= (\delta^{(3)} \cdot W_2^{(3)}) \cdot f'(h_2^{(2)}) = 1.5502 \\ \delta_1^{(1)} &= (\delta_1^{(2)} \cdot W_1^{(2)} + \delta_2^{(2)} \cdot W_3^{(2)}) \cdot f'(h_1^{(1)}) = 2.3899 \\ \delta_2^{(1)} &= (\delta_1^{(2)} \cdot W_2^{(2)} + \delta_2^{(2)} \cdot W_4^{(2)}) \cdot f'(h_1^{(1)}) = 2.6870 \end{aligned}$$

kada smo to izračunali, sada možemo ažurirati težine i parametre pristranosti

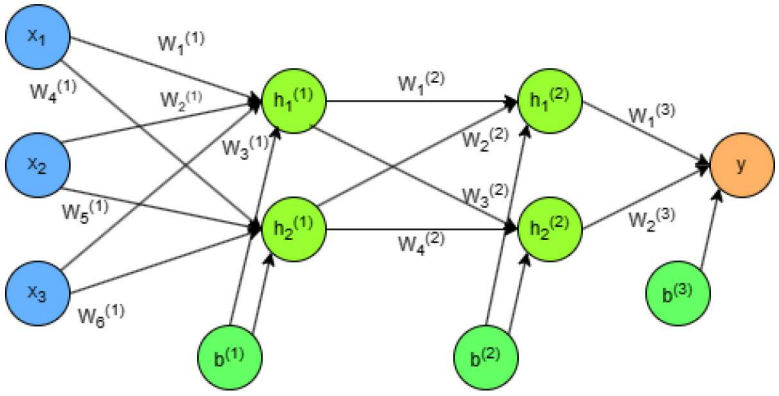
$$
\begin{aligned}
W^{(3)}_{1,novi} &= W^{(3)}_1 - \alpha \cdot \delta^{(3)} = 0.9708 \\
W^{(3)}_{2,novi} &= W^{(3)}_2 - \alpha \cdot \delta^{(3)} = 1.0708 \\
b^{(3)}_{novi} &= b^{(3)} - \alpha \cdot \delta^{(3)} = 0.3708
\end{aligned}
$$

$$
\begin{aligned}
W^{(2)}_{1,novi} &= W^{(2)}_1 - \alpha \cdot \delta^{(2)}_1 = 0.5579 \\
W^{(2)}_{2,novi} &= W^{(2)}_2 - \alpha \cdot \delta^{(2)}_1 = 0.6579 \\
W^{(2)}_{3,novi} &= W^{(2)}_3 - \alpha \cdot \delta^{(2)}_2 = 0.7450 \\
W^{(2)}_{4,novi} &= W^{(2)}_4 - \alpha \cdot \delta^{(2)}_2 = 0.8450 \\
b^{(2)}_{1,novi} &= b^{(2)}_1 - \alpha \cdot \delta^{(2)}_1 = 0.1579 \\
b^{(2)}_{2,novi} &= b^{(2)}_2 - \alpha \cdot \delta^{(2)}_2 = 0.2450
\end{aligned}
$$

$$
\begin{aligned}
W^{(1)}_{1,novi} &= W^{(1)}_1 - \alpha \cdot \delta^{(1)}_1 = -0.1389 \\
W^{(1)}_{2,novi} &= W^{(1)}_2 - \alpha \cdot \delta^{(1)}_1 = -0.0389 \\
W^{(1)}_{3,novi} &= W^{(1)}_3 - \alpha \cdot \delta^{(1)}_1 = 0.0611 \\
W^{(1)}_{4,novi} &= W^{(1)}_4 - \alpha \cdot \delta^{(1)}_2 = 0.1313 \\
W^{(1)}_{5,novi} &= W^{(1)}_5 - \alpha \cdot \delta^{(1)}_2 = 0.2313 \\
W^{(1)}_{6,novi} &= W^{(1)}_6 - \alpha \cdot \delta^{(1)}_2 = 0.3313 \\
b^{(1)}_{1,novi} &= b^{(1)}_1 - \alpha \cdot \delta^{(1)}_1 = -0.1389 \\
b^{(1)}_{2,novi} &= b^{(1)}_2 - \alpha \cdot \delta^{(1)}_2 = -0.0687
\end{aligned}
$$

S ovim smo završili prvu epohu u procesu treniranja i ovaj postupak se ponavlja kroz svaku epohu.

U primjeru smo mogli vidjeti da derivacija aktivacijske funkcije (ReLU) nije definirana za $x = 0$, slično se može dogoditi s ostalim aktivacijskim funkcijama ili funkcijama gubitka. Kada računamo gradijente moramo voditi računa o funkcijama koje koristimo kako naš model ne bi zahvatio jedan od sljedećih problema, kao što su eksplodirajući ili nestajući gradijenti. Nestajući gradijenti najbolje se objašnjavaju na neuronskoj mreži s mnogo slojeva i malo neurona. Na primjer, pretpostavimo da neuronska mreža u svakom sloju ima samo jedan neuron. Kako računamo gradijente unatrag od zadnjeg do prvog sloja, što više idemo prema početnim slojevima, vidimo manje promjene u odnosu na kasnije slojeve, jer je vrijednost gradijenta tako mala da nema značajan utjecaj. Eksplodirajući gradijenti predstavljaju sličan problem, ali s vrijednostima gradijenta koje postaju sve veće i povećavaju vrijednost određenih parametara, što može dovesti do pogrešnog učenja modela. Postoje načini za rješavanje ovih problema, kao što su podrezivanje



Slika 2.6: Primjer neuronske mreže

gradijenta i preskakanje veza. Podrezivanje gradijenta je metoda u kojoj računamo normu gradijenta te je uspoređujemo s pragom koji je unaprijed određen. Ako je norma veća od praga, skaliramo gradijent i nastavljamo s računanjem. Ovo možemo interpretirati kao kretanje u istom smjeru, samo s manjim koracima.

Još dva česta problema koja možemo susresti, ali nisu povezana s gradijentima, već s ulaznim podacima, su Overfitting i Underfitting. Overfitting se javlja kada je model pretreniran, tj. kada se dobro prilagodi podacima za treniranje, ali ne uspijeva dobro generalizirati na nove podatke. Underfitting je suprotno, jer je model jednostavan i ne može dobro naučiti tijekom treniranja. Ova dva problema možemo riješiti pomoću regularizacije, koja ograničava model kako bi izbjegao korištenje manje relevantnih podataka tijekom učenja.

Jedna od tih metoda je i dropout. Funkcija dropout se koristi u unaprijednoj propagaciji kroz neuronsku mrežu i često se spominje pod nazivom dropout sloj. Ideja je nasumično postaviti vrijednosti neurona na nulu u skrivenim slojevima tijekom svake epohe. Tu imamo parametar pomoću kojeg odlučujemo koliko ćemo neurona postaviti na nulu, obično se taj parametar izražava kao postotak. Naglasili smo da se ovaj proces primjenjuje samo tijekom unaprijedne propagacije, ali njegov utjecaj se vidi i tijekom povratne propagacije. Tijekom povratne propagacije gradijenti se neće računati za one neurone za koje je dropout postavio vrijednosti na nulu.

3 | Rekurentne neuronske mreže - RNN

Nakon što smo se upoznali s osnovama neuronskih mreža, polako ćemo početi istraživati specifičnije modele neuronskih mreža, a u ovom poglavlju to će biti rekurentne neuronske mreže (RNN). Rekurentne neuronske mreže su skup neuronskih mreža za obradu sekvencijalnih podataka. Često vremenski periodi, tekstovi i biološki podatci sadrže sekvencijalne ovisnosti među atributima, stoga su RNN-ovi prikladni za takve situacije. Na primjer, u vremenskim periodima, vrijednosti s uzastopnim vremenskim oznakama su blisko povezane jedna s drugom. Ako koristimo vrijednosti tih vremenskih oznaka kao neovisne značajke, ključne informacije o odnosima između tih vrijednosti se gube. Slično i je s obradom teksta jer možemo dobit bolji semantički uvid kada se koristi redoslijed riječi. U takvim slučajevima važno je konstruirati modele koji uzimaju u obzir informacije o slijedu. Također biološki podaci često sadrže nizove, u kojima simboli mogu odgovarati aminokiselinama ili drugim spojevima koje čine gradivne blokove DNA. U ovom poglavlju ćemo se upoznati s arhitekturom rekurentnih mreža i nekim njezinim specifičnim verzijama.

3.1 Arhitektura rekurentnih neuronskih mreža

Kada objašnjavamo arhitekturu ove mreže, razmotrit ćemo problem obrade teksta, gdje ćemo koristiti RNN kao jezični model. U rekurentnoj neuronskoj mreži postoji jedan na jedan odnos između slojeva u mreži i određenih pozicija u sekvenci. Važna napomena o nizovima je da uzastopne riječi ovise jedna o drugoj. Stoga je korisno primiti određeni ulaz $x^{(t)}$ tek nakon što su raniji ulazi već primljeni i obrađeni u skriveno stanje. Ulazne vrijednosti su niz riječi koje se obrađuju, a na njih se primjenjuju težine kako bismo dobili skriveno stanje. U skrivenom sloju, skrivena stanja su međusobno povezana, uz to što su povezana sa sljedećim skrivenim ili izlaznim stanjima, a na tim vezama primjenjuju se težine. Nakon skrivenog stanja dolazimo do izlaznog stanja. Razlika između unaprijed propagiranih mreža i rekurentnih mreža je u povezanosti skrivenih stanja u istome skrivenom sloju.

Sada ćemo detaljnjije pogledati što se točno događa i kako su naši podatci zapisani. Vektor ulaznih parametara $x^{(t)} \in \mathbb{R}^d$ je d dimenzionalni vektor u nekom trenutku t, $t = 1, \ldots, T$. Rekli smo da je naš model temeljen na jeziku pa su ulazne vrijednosti riječi. Da bismo riječi mogli koristiti, možemo ih prilagoditi da nam više odgovaraju pri računanju tako što ćemo ih zapisati kao one-hot vektore, tj.

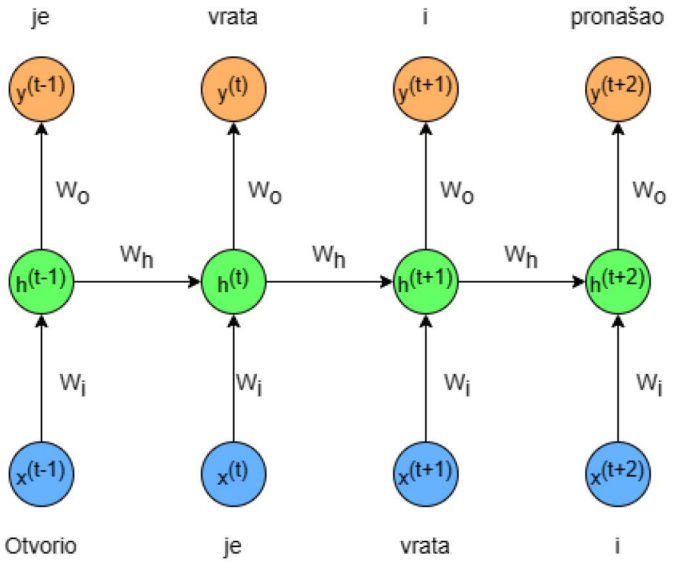
vektor koji ima samo jednu jedinicu na specifičnom mjestu dok su ostale vrijednost jednake nuli. U sljedećem koraku primjenjujemo matricu težina W_i tipa $p \times d$ na ulazne vrijednosti te matricu težina W_h tipa $p \times p$ na prethodno skriveno stanje. Ovo je specifičnost rekurentnih neuronskih mreža, jer skrivena stanja ovise o drugim skrivenim stanjima iz istog sloja. Za kraj, računamo aktivacijksu funkciju hiperbolnog tangensa što se može zapisati

$$h^{(t)} = tanh(W_h h^{(t-1)} + W_i x^{(t)} + b_1)$$

i sada smo izračunali vrijednost skrivenog stanja u trenutku t, pri čemu je $b_1 \in \mathbb{R}^p$ vektor pristranosti. Ovdje smo vidjeli kako neuronske mreže iskorištavaju prethodno izračunate vrijednosti da bi svaka sljedeća vrijednost bila bolje izračunata. Sljedeće što računamo je izlaznu vrijednost u trenutku t gdje možemo koristiti softmax kao aktivacijsku funkciju, pri čemu je $b_2 \in \mathbb{R}^d$ vektor pristranosti.

$$y^{(t)} = softmax(W_o h^{(t)} + b_2)$$

Tu imamo matricu težeina W_o tipa $d \times p$ koju također primjenjujemo na sva skirvena stanja. Ovakvu konstrukciju možemo vidjeti na slici 3.1.



Slika 3.1: Detaljan model rekurentne neuronske mreže.

Ostalo je još nekoliko stvari koje trebamo spomenuti, a to su vektori pristranosti koji mogu i ne moraju biti dio mreže. Naš rezultat će biti distribucija vjerojatnosti da je $y^{(t)}$ određena riječ koja dolazi sljdeća u nizu tj. možemo reći da $y^{(t)}$ treba biti $x^{(t+1)}$. Sada kada smo gotovi s unaprijednom propagacijom možemo prijeći na povratnu propagaciju.

Uzmimo unakrsnu entropiju kao funkciju gubitka, ali budući da naša mreža ima više izlaznih stanja u trenutku t, funkcija gubitka će biti suma unakrsnih entropija.

$$l^{(t)} = H(y_{org}^{(t)}, y_{pred}^{(t)}) = -\log y_{pred,id(x^{(t+1)})}^{(t)}$$

$$l = \frac{1}{T}\sum_{t=1}^{T} l^{(t)} = -\frac{1}{T}\sum_{t=1}^{T} \log y_{pred,id(x^{(t+1)})}^{(t)}$$

Gdje nam je $id(x^{(t)})$ indeks jedinice u one-hot vektoru. Sada idemo računati gradijente od izlaznog, skrivenog i ulaznog sloja

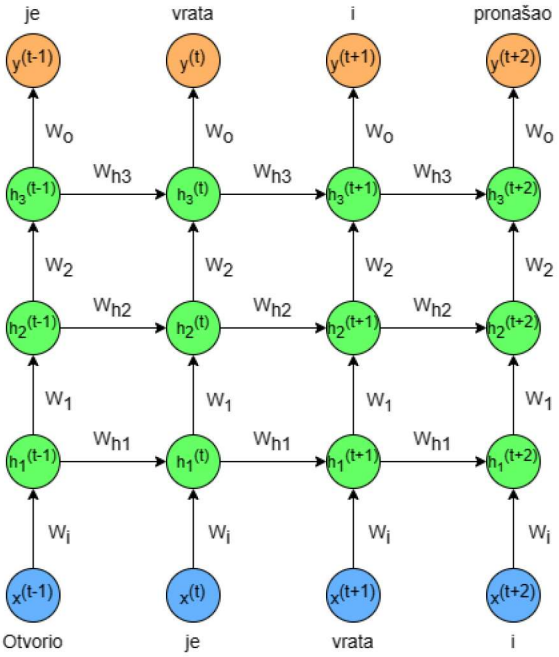
$$\frac{\partial l}{\partial W_o} = \frac{\partial l}{\partial y_j} \cdot \frac{\partial y_j}{\partial (W_o \cdot h_j)} \cdot \frac{\partial (W_o \cdot h)}{\partial W_o}$$
$$\frac{\partial l}{\partial W_h} = \frac{\partial l}{\partial y_j} \cdot \frac{\partial y_j}{\partial (W_o \cdot h_j)} \cdot \frac{\partial (W_o \cdot h_j)}{\partial h_j} \cdot \frac{\partial h_j}{\partial (W_h \cdot h_{j-1})} \cdot \frac{\partial (W_h \cdot h_{j-1})}{\partial W_h}$$
$$\frac{\partial l}{\partial W_i} = \frac{\partial l}{\partial y_j} \cdot \frac{\partial y_j}{\partial (W_o \cdot h_j)} \cdot \frac{\partial (W_o \cdot h_j)}{\partial h_j} \cdot \frac{\partial h_j}{\partial (W_i \cdot x_j)} \cdot \frac{\partial (W_i \cdot x_j)}{\partial W_i}$$

Ovdje smo uveli indeks $j \in \mathbb{N}$ koji predstavlja neuron u jednom od tri sloja koje smo prethodno naveli. S ovime završavamo dio o arhitekturi, ali važno je napomenuti da rekurentne neuronske mreže nisu svaršene, jer ako koristimo povratnu propagaciju postoje značajni izazovi u učenju parametara kao što us problem nestajućeg i eksplodirajućeg gradijenta [2].

3.2 Varijante rekurentnih neuronskih mreža

Do sada smo pričali samo o arhitekturi rekurentne neuronske mreže koja sadrži samo jedan skriveni sloj radi lakšeg razumijevanja. Kako smo se već upoznali višeslojne neuronske mreže, možemo zaključiti da onda postoje i višeslojne rekurentne neuronske mreže. One postoje kako bi se mogli izgradili modeli veće složenosti i time postići bolji rezultati. U arhitekturi se samo mijenja skriveni dio jer dodajemo još skrivenih slojeva. Na slici 3.2 možemo vidjeti kako izgleda arhitektura višeslojne neuronske mreže. Treba naglasiti da su težine W_i između ulaznog sloja i prvog skrivenog sloja, težine W_o su između zadnjeg skrivenog sloja i izlaznog sloja. Težine W_j povezuju skrivene slojeve, gdje $j \in \mathbb{N}$ predstavlja skriveni sloj, a težine W_{hj} se nalaze na povratnim vezama unutar istog sloja. Kada imamo više skrivenih slojeva tada indeks j kod stanja h_j određuje u kojem sloju se nalazi skriveno stanje.

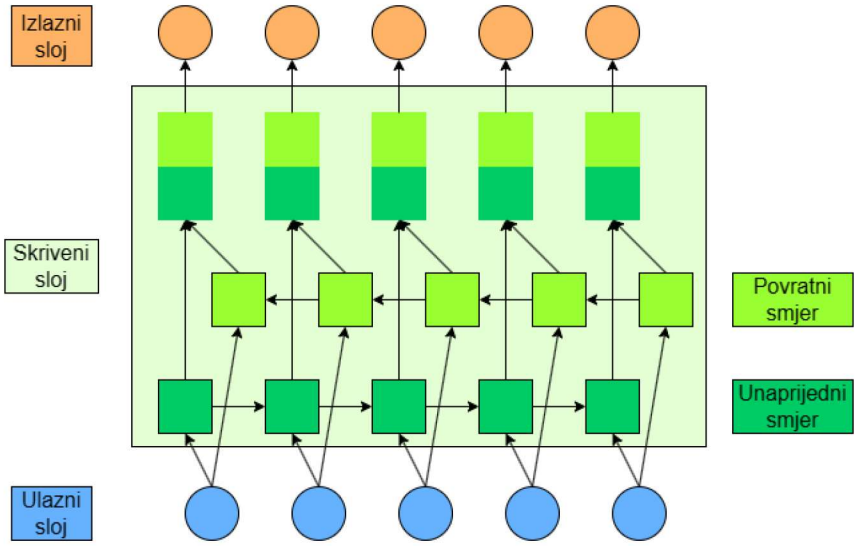
Sljedeća popularna varijanta je dvosmjerna rekurentna neuronska mreža. Ova varijanta sastoji se od dvije rekurentne neuronske mreže, pri čemu svaka mreža ima svoje težine. U njenom skrivenom sloju postoje dva smjera, tj. ima unaprijedni smjer (prva mreža) iz prethodnog stanja u buduće stanje tako da $(t-1)$ ide u t pa onda u $(t+1)$ i tako dalje. Drugi smjer (druga mreža) možemo zvati povratni, jer ide iz budućih stanja u prethodna tako da $(t+1)$ ide u t pa onda u $(t-1)$. Rezultae



Slika 3.2: Arhitektura višeslojne rekurentne neuronske mreže

koje dobijemo iz oba dva smjera spajamo zajedno u jedan rezultat i prosljeđujemo ga u sljedeći skriveni ili izlazni sloj. Ovu arhitekturu možemo vidjeti na slici 3.3.

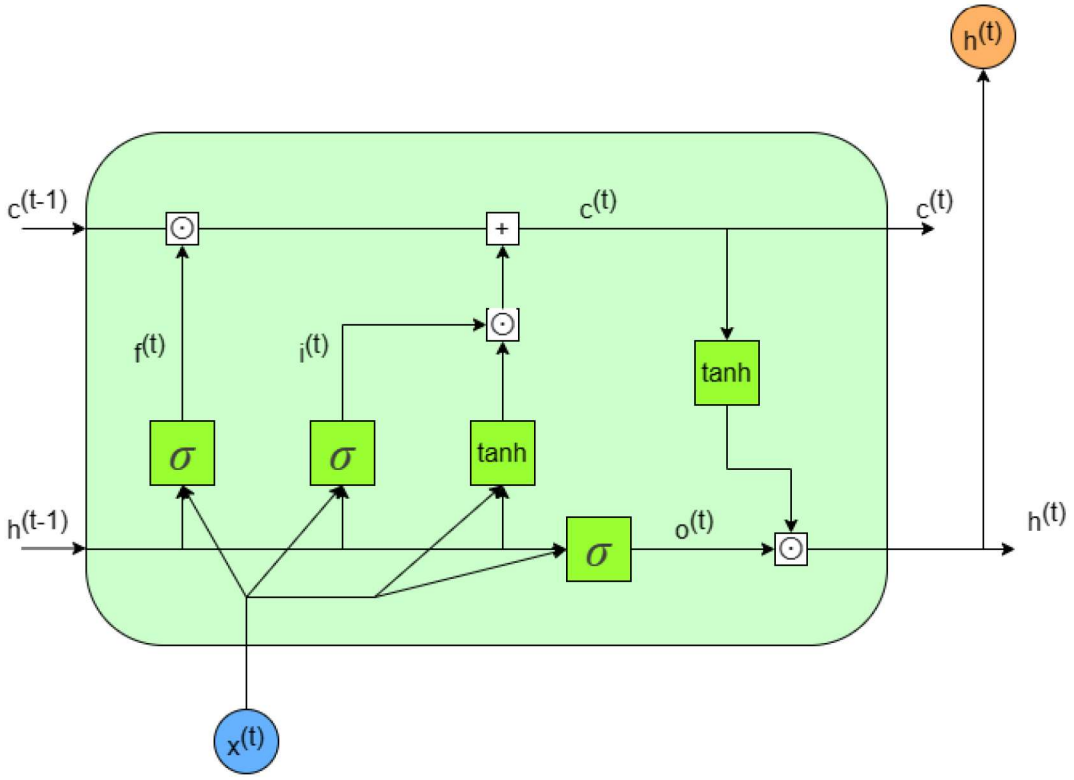
Postoji još mnogo varijanti rekurentnih neuronskih mreža, neke od njih su hijerarhijski RNN, Gated Recurent Unit (GRU), clockwork RNN. Jednu od njih ćemo obraditi u sljedećem poglavlju, a ona se zove Long short term memory(LSTM).



Slika 3.3: Arhitektura dvosmjerne rekurentne neuronske mreže

4 | Long short term memory - LSTM

Već smo napomenuli da rekurentne neuronske mreže imaju probleme povezane s nestajanjem i eksplodiranjem gradijenata, što je jedan od razloga zašto sada pričamo o LSTM-u. Hochreiter i Schmidhuber su 1997. uveli LSTM model prije svega kako bi prevladali problem nestajućih gradijenata [4]. Osnovni princip arhitekture je da će mreža biti dizajnirana s ciljem pouzdanog prijenosa važnih informacija kroz mnogo vremenskih koraka unaprijed, LSTM zato ima mogućnost pamćenja informacija. LSTM se može primjeniti na različite vrste problema, uključujući klasifikacijske probleme, obradu i predviđanje vremenski povezanih podatke, kao što su predviđanje vremena, analize sentimenta, prepoznavanje govora i strojno prevođenje.



Slika 4.1: Arhitektura LSTM ćelije

LSTM model ima drugačiju arhitekturu u usporedbi s običnim rekurentnnim neurnoskim mrežama. Umjesto da koristimo skriveni sloj, kod LSTM-a koristimo ćelije što možemo vidjeti na slici 4.1. Ćelije su međusobno rekurentno povezane kao skrivena stanja u klasičnim rekurentnim nuronskim mrežama. Ove ćelije se

nazivaju LSTM ćelije, a osim njih naša mreža ima ulazni i izlazni sloj kao i svaka druga rekurentna nuronska mreža. LSTM ćelija uvodi dodatne parametre i sistem propusnica pomoću kojih regulira obradu podataka. Ulazna vrijednost ostaje vektor $x^{(t)}$ u trenutku t, pri čemu je $x^{(t)}$ d dimenzionalni vektor, $d \in \mathbb{N}$. Novi bitan parametar koji se pojavljuje je stanje ćelije $c^{(t)}$ u trenutku t koje dimenziju veličine $p \in \mathbb{N}$. Parametar $c^{(t)}$ uvodimo kako bi mogli zadržati i pohraniti dugoročne informacije. Osim stanja ćelije, spomenuli smo i propusnice, prva propusnica je propusnica zaboravljanja (forget gate). Propusnicu zaboravljanja ćemo označiti s $f^{(t)}$,$f^{(t)} \in (0,1)$ u trenutku t, pri čemu joj je cilj joj je kontrolirati koje informacije će se odbaciti, a koje zadržati iz prethodnog stanja.

$$f^{(t)} = \sigma(W_f h^{(t-1)} + U_f x^{(t)} + b_f)$$

Sljedeća propusnica je propusnica ulaza (input gate) $i^{(t)} \in (0,1)$ u trenutku t čija je svrha selektivno propustiti informacije iz ulaza.

$$i^{(t)} = \sigma(W_i h^{(t-1)} + U_i x^{(t)} + b_i)$$

Još imamo propusnicu izlaza (output gate) $o^{(t)} \in (0,1)$ u trenutku t koja također selektivno propušta infomracije iz ćelije prema izlazu.

$$o^{(t)} = \sigma(W_o h^{(t-1)} + U_o x^{(t)} + b_o)$$

Propusnice se mogu tumačiti kao logički sklopovi gdje će npr. za rezultat 1 dodati vrijednosti u stanje ćelije ili za rezultat 0 zaboraviti vrijednost u stanju ćelije ili ako vrijednost bude između 0 i 1 onda propustiti određene podatke u skriveno stanje. Sada kada smo vidjeli kako funkcioniraju propusnice možemo pokazati kako računamo novo stanje ćelije $\tilde{c}^{(t)} \in (-1,1)$ u trenutku t.

$$\tilde{c}^{(t)} = \tanh(W_c h^{(t-1)} + U_c x^{(t)} + b_c)$$

Trebamo još reći da je σ sigmoidna funkcija, tanh je hiperbolni tangens, W_f, W_i, W_o, $W_c \in \mathbb{R}^{p \times p}$ matrice težina koje se primjenjuju na rekurentnim vezama, a U_f, U_i, U_o, $U_c \in \mathbb{R}^{p \times d}$ matrice težina koje se primjenjuju na ulazne vrijednosti. Osim njih imamo još o parametre pristranosti b_f, b_i, b_o, $b_c \in \mathbb{R}^p$.

Nakon što smo sve ovo izračunali možemo nastaviti s računanjem skrivenog stanja $h^{(t)} \in (-1,1)$ i stanja ćelije

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t-1)}$$

$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)})$$

Možemo reći da pri računanju $c^{(t)}$ selektivno zaboravljamo podatke iz prethodnog stanja ćelije i dodajemo nove podatke u našu dugoročnu memoriju. Slično, kod $h^{(t)}$ selektivno propuštamo podatke iz dugoročne memorije u skriveno stanje. Znak $\odot$ predstavlja umnožak po komponentama(Hadamardov umnožak). Trebamo također naglasiti da $h^{(t)}$ možemo tumačiti kao $y^{(t)}$ u trenutku t. Kad smo sve to izračunali onda prosljeđujemo skriveno stanje $h^{(t)}$ i stanje ćelije $c^{(t)}$ u sljedeću ćeliju u trenutku $t+1$.

Ako pogledamo jednadžbu za $c^b(t)$ i pokušamo je derivirati po $c^{(t-1)}$, vidimo da nam ostaje $f^{(t)}$, a znamo da su njezine vrijednosti između $(0,1)$. Već s ovim rezultatom značajno smanjujemo vjerojatnost eksplodirajućeg i nestajućeg gradijenta, slično možemo reći i za $h^{(t)}$ jer smo ga definirali pomoću stanja ćelije $c^{(t)}$ stoga ovdje možemo lakše provesti povratnu propagaciju.

5 | Eksperiment

Sada kad smo se upoznali s LSTM neuronskom mrežom te ostalim svojstvima neuronskih mreža, možemo se fokusirati na istraživanje ovog rada. Željeli smo riješiti problem predviđanja koncentracije peludi u zraku tako da imamo kvalitetnu prognozu pomoću koje bismo mogli na vrijeme obavijestiti osobe koje imaju problema s peludnim alergijama. Podaci koji su korišteni u ovome radu prikupljeni su pomoću uređaja za mjerenje čestica u zraku u sklopu projekta RealForAll [13]. Period koji naši podaci pokrivaju traje 22 godine, od 1.1.2000. do 31.12.2021., što čini ukupno 8036 dana.

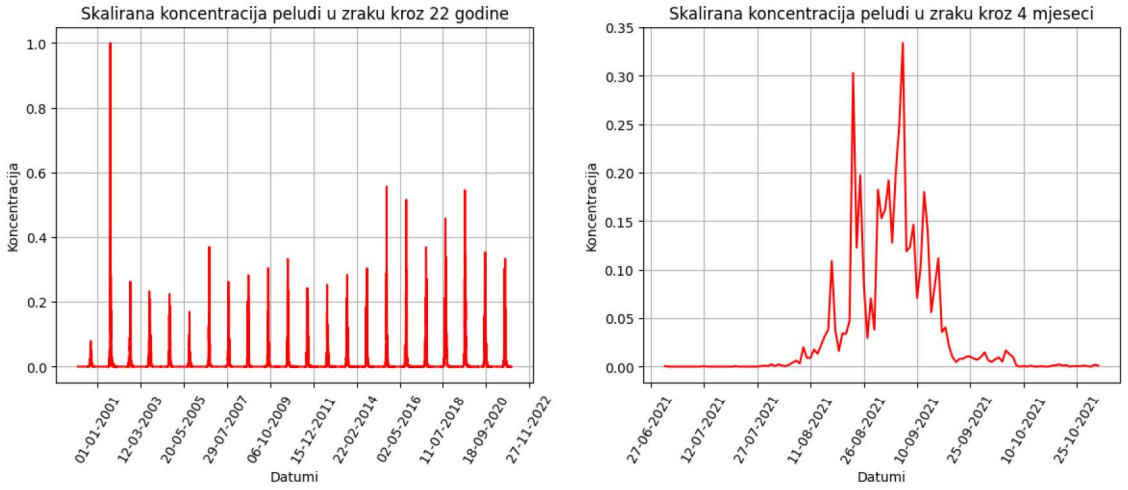
Podaci su spremljeni u Excel tablicu, pri čemu jedan element odgovara jednom retku u tablici, koji ćemo označiti kao x. Naš x sadrži 21 značajku koje ćemo sada opisati. Prve četiri značajke su kiša, snijeg, olujno nevrijeme i magla, a njihove vrijednosti mogu biti 0 ako takvih vremenskih pojava nije bilo ili 1 ako je bilo takvih vremenskih pojava. Sljedećih 9 značajki su brojčane vrijednosti koje reprezentiraju srednju brzinu vjetra, maksimalnu brzinu vjetra, vlažnost zraka, atmosferski tlak, minimalnu dnevnu temperaturu, srednju dnevnu temperaturu i maksimalnu dnevnu temperaturu. Sljedećih 5 parametara obuhvaća godinu u rasponu od 2000. do 2021., mjesec u rasponu od 1. do 12., dan u rasponu od 1. do 31., te redni broj dana u godini u rasponu od 1 do 366 zbog prijestupnih godina. Nakon toga dolazi lokacija, koja ima samo jednu vrijednost jer su podaci dobiveni samo s jednog uređaja. Imamo još 3 parametra koji predstavljaju prosječnu koncentraciju ambrozije, breze i trave. Za zadnju značajku imamo datum kada je napravljen taj unos.

Obradu podataka i izgradnju LSTM modela odlučili smo napraviti u programskom jeziku Python uz biblioteke PyTorch, pandas, NumPy, scikit-learn i Matplotlib. Od 21 značajke, odlučili smo da ćemo za našu ulaznu varijablu uzeti njih 17. Ove 4 značajke koje smo izbacili su lokacija jer svaki unos ima istu lokaciju, tako da taj parametar ne bi previše utjecao na model, datum jer već imamo podatke o danu, mjesecu i godini kao zasebne značajke. Zadnja dva parametra su prosječna koncentracija breze i trave, koje smo samo isključili jer nam je ideja bila da istreniramo model samo na jednoj vrsti. Naš model i rezultati koje smo dobili temelje se samo na prosječnoj koncentraciji ambrozije, jer je ambrozija biljka koja izaziva najviše simptoma u Hrvatskoj [14].

Kako imamo veliki raspon vrijednosti između naših značajki, odlučili smo normalizirati podatke pomoću MinMaxScaler-a, čija funkcija izgleda ovako:

$$X_{scale} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Rezultat ove funkcije su podaci u rasponu od 0 do 1. Na slici 5.1 možemo vidjeti skalirane vrijednosti podataka kroz period od 22 godine. Dodatno, odlučili smo da ćemo podatke iz našeg skupa filtrirati. Na slici 5.1 vidimo da je velik dio podataka o prosječnoj koncentraciji ambrozije jednak nuli, što ima smisla jer ambrozija cvate tijekom ljeta, točnije u 8. i 9. mjesecu. Stoga smo odlučili iz svake godine gledati period od 1.6. do 30.11. Imamo dakle dva skupa podataka na kojima smo trenirali naš model: prvi skup obuhvaća period od 22 godina sa svim danima, dok je drugi skup podataka u rasponu od 22 godina, ali za pola manji jer iz svake godine uzimamo period od 6 mjeseci, što čini 4026 dana.



(a) (b)

Slika 5.1: Graf (a) prikazuje vrijednosti svih podataka kroz period od 22 godine. Graf (b) prikazuje vrijednsoti podataka kroz period od 4 mjeseci.

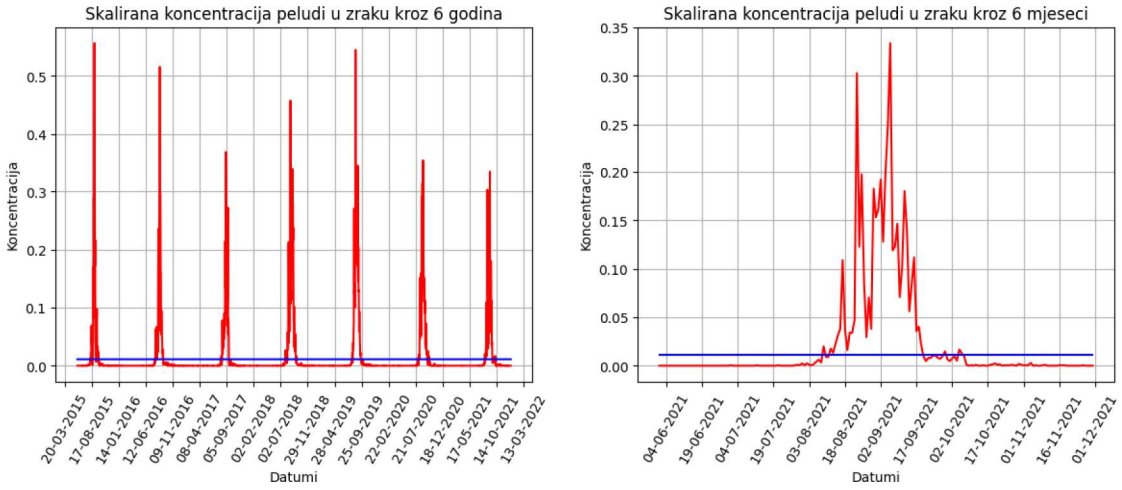
Priprema podataka još nije gotova jer ćemo naše podatke podijeliti na skup podataka za treniranje, koji će sadržavati 70% od ukupnog broja podataka, što iznosi 5625 dana ili 2818 dana za filtrirani skup podataka. Preostalih 30% podataka bit će rezervirano za testiranje modela, što čini 2408 dana za ne filtrirani skup podataka, a za filtrirani skup podataka to je 1205 dana. Kako radimo s vremenski ovisnim podacima, odlučili smo da će ulazni podaci sadržavati informacije za 3, 5 i 7 dana, odnosno $T \in \{3,5,7\}$. Za pojednostavljeni primjer, možemo uzeti da je $x = [1,2,3,4,5,6,7]$, pa će naš model dobiti na ulazu $[1,2,3]$ te će trebati predvidjeti 4, zatim će dobiti $[2,3,4]$ i kao rezultat će trebati predvidjeti 5, i tako dalje. Slično funkcionira za 5 i 7 dana.

Nakon što smo pripremili podatke, možemo opisati naš model [20]. Koristili smo PyTorch-ovu implementaciju LSTM-a, koja odgovara arhitekturi koju smo opisali u poglavlju 4. Parametri koji su nam bili potrebni za definiranje našeg modela uključuju veličinu ulaza, što za nas iznosi broj značajki (17), veličinu skrivenog sloja, odnosno broj skrivenih stanja koje smo postavili na 512. Također, definirali smo da će model koristiti vektor pristranosti. Treba još dodati da u našem modelu ne proslijeđujemo h^0 i c^0, pa su oni automatski postavljeni na nulu. Nakon LSTM-a, dodali smo još jedan Linearni sloj iz PyTorcha koji prima rezultat

od LSTM-a i izračunava ga pomoću sljedeće jednadžbe:

$$y = xA^T + b$$

i kao rezultat, želimo dobiti prosječnu koncentraciju peludi u zraku za sljedeći dan. Osim ovog pristupa, odlučili smo probati dodati još i dropout sloj, također iz PyTorch-a, između LSTM-a i Linearnog sloja kako bismo pokušali dobiti bolju predikciju. Za dropout smo uzeli vjerojatnosti od 25% i 50%, pri čemu je to vjerojatnost da će element biti jednak 0. Za funkciju gubitka odlučili smo koristiti PyTorch MSE funkciju, koju smo opisali u poglavlju o funkcijama gubitka. Kod optimizacije gradijenata, odlučili smo koristiti PyTorch-ev Adam (Adaptive Moment Estimation) optimizacijski algoritam. Neki od prednosti Adam algoritma su što može prilagoditi stopu učenja za svaki parametar tijekom učenja, brzo konvergira te uspješno normalizira gradijent. Što se tiče parametara, ostali su nam još stopa učenja, čije smo vrijednosti uzeli da budu 0.1, 0.01 i 0.001, te epohe. Za epohe smo odredili da će biti 300, 500, 700 i 1000.



(a) (b)

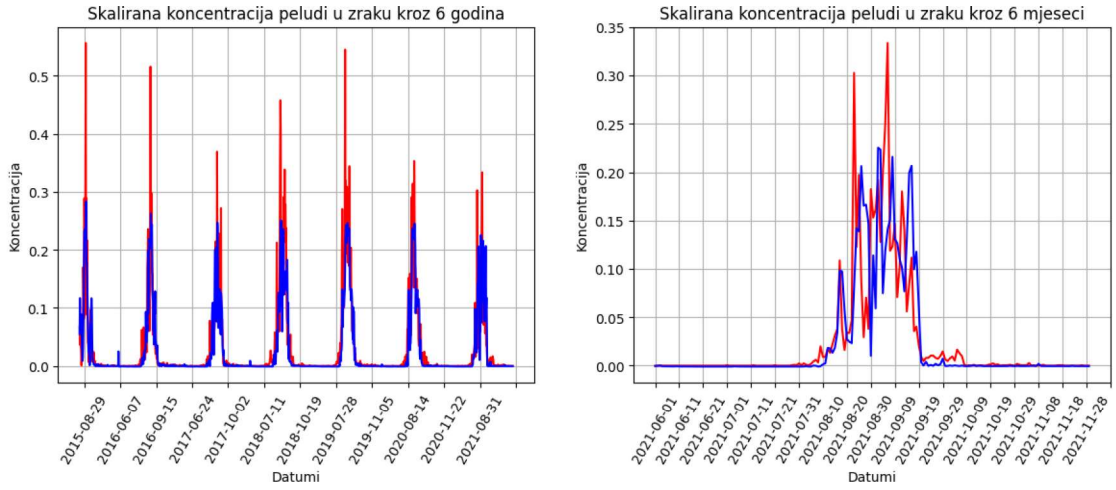
Slika 5.2: Graf pokazuje originalne vrijednosti (crvena boja) i predikciju (plava boja) kada je stopa učenja 0.1. Graf (a) prikazuje vrijednosti svih podataka kroz period od 7 godine dok graf (b) pokazuje vrijednsoti podataka kroz period od 6 mjeseci.

Rezultate koje smo dobili mogu se vidjeti u tablicama 5.1, 5.2 i 5.3. Svaka tablica predstavlja rezultate koji su se dobili za period od 3 dana, 5 dana ili 7 dana. Stupac 'Filtrirano' ima parametre 'Da' i 'Ne', pri čemu vrijednost 'Da' znači da smo uzeli samo 6 mjeseci iz svake godine, dok parametar 'Ne' znači da smo uzeli cijeli skup podataka. Obojeni dio tablice sadrži postotke koji predstavljaju preciznost modela, odnosno koliko naša predikcija u testnoj fazi odgovara stvarnim podatcima. Rezultate smo dobili primjenom Explained Variance Regression Score funkcije, koja je definirana na sljedeći način:

$$S(y_{org}, y_{pred}) = 1 - \frac{Var(y_{org} - y_{pred})}{Var(y_{org})}$$

pri čemu je *Var* varijanca. U tablicama se pod stupcem 'Stopa učenja' neće vidjeti podatci za stopu učenja od 0.1, jer rezultati nisu bili dobri, što možemo vidjeti na grafu na slici 5.2 pa ih nismo odlučili prikazati u tablicama.

Sada ćemo istaknuti najbolje rezultate. U dijelu gdje smo našem modelu proslijedili informacije od 3 prethodna dana kako bi nam vratio predikciju za četvrti dan, najveća preciznost bila je 69.91%. U tom pokušaju su podatci bili filtrirani, primijenili smo dropout od 50%, stopa učenja je bila 0.01, te smo rezultat dobili nakon 1000 epoha. Na slici 5.3 možemo vidjeti grafički kako se kreće naša predikcija.

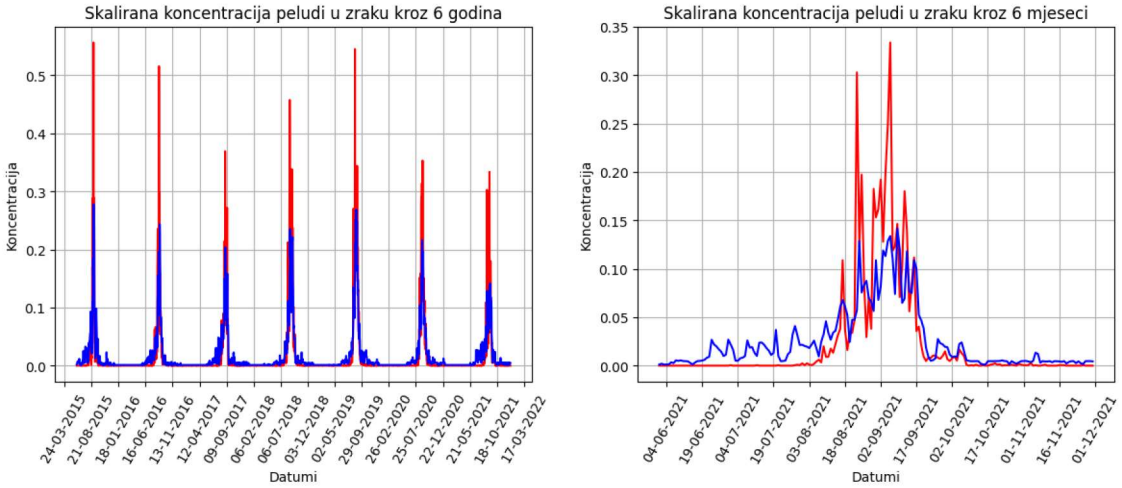


(a) (b)

Slika 5.3: Graf pokazuje originalne vrijednosti (crvena boja) i predikciju (plava boja) za period od 3 dana. Graf (a) prikazuje vrijednosti svih podataka kroz period od 6 godina dok graf (b) pokazuje vrijednsoti podataka kroz period od 6 mjeseci.

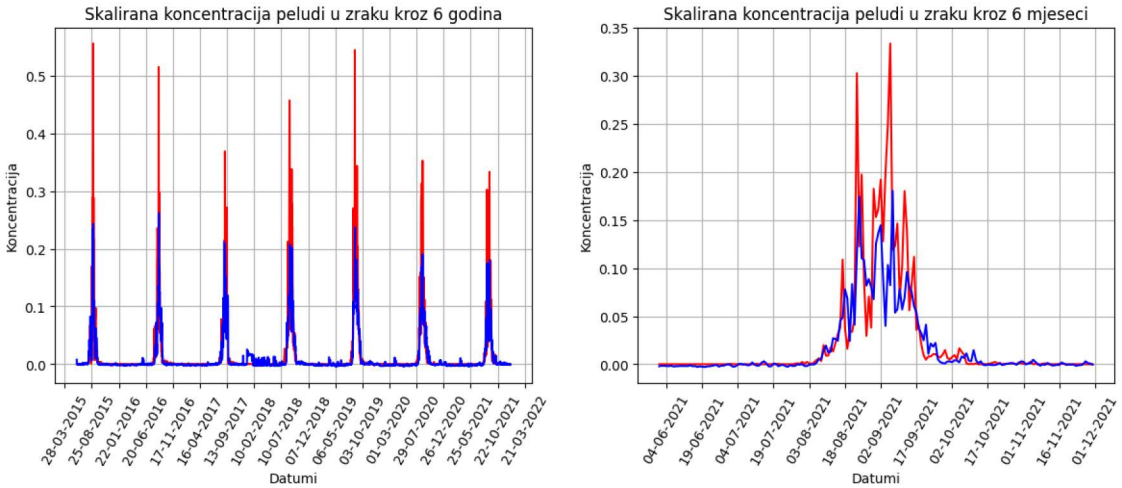
Sljedeći po redu je period od 5 dana gdje smo dobili predikciju za šesti dan. Ovdje nam je najveća preciznost bila 70.17%. U ovom slučaju podatci nisu bili filtrirani, primijenili smo dropout od 25%, stopa učenja je bila 0.01, te smo rezultat dobili nakon 1000 epoha. Na slici 5.4 možemo vidjeti grafički kako se kreće naša predikcija.

Zadnji nam je ostao period od 7 dana gdje smo dobili predikciju za osmi dan. Ovdje nam je najveća preciznost bila 68.51%. U ovom slučaju podatci nisu bili filtrirani, primijenili smo dropout od 50%, stopa učenja je bila 0.001, te smo rezultat dobili nakon 300 epoha. Na slici 5.5 možemo vidjeti grafički kako se kreće naša predikcija.



(a) (b)

Slika 5.4: Graf pokazuje originalne vrijednosti (crvena boja) i predikciju (plava boja) za period od 5 dana. Graf (a) prikazuje vrijednosti svih podataka kroz period od 6 godina dok graf (b) pokazuje vrijednsoti podataka kroz period od 6 mjeseci.



(a) (b)

Slika 5.5: Graf pokazuje originalne vrijednosti (crvena boja) i predikciju (plava boja) za period od 7 dana. Graf (a) prikazuje vrijednosti svih podataka kroz period od 6 godina dok graf (b) pokazuje vrijednsoti podataka kroz period od 6 mjeseci.

Filtrirano	Dropout	Stopa učenja	Epohe			
			300	500	700	1000
Da	0	0.01	54.79%	57.87%	55.95%	49.08%
		0.001	59.26%	53.41%	51.83%	52.04%
	0.25	0.01	55.89%	61.42%	53.60%	64.45%
		0.001	54.18%	56.94%	56.16%	61.03%
	0.5	0.01	54.07%	58.13%	57.41%	69.91%
		0.001	57.77%	60.56%	56.21%	56.72%
Ne	0	0.01	59.95%	60.37%	60.72%	57.76%
		0.001	56.24%	56.48%	51.03%	55.89%
	0.25	0.01	66.03%	53.03%	64.84%	62.55%
		0.001	57.21%	54.03%	55.69%	54.54%
	0.5	0.01	60.65%	56.74%	65.50%	63.85%
		0.001	51.80%	46.22%	50.38%	58.61%

Tablica 5.1: Tablica parametara modela i postotka točnosti modela (obojeni dio) za period od 3 dana

Filtrirano	Dropout	Stopa učenja	Epohe			
			300	500	700	1000
Da	0	0.01	45.86%	56.61%	56.40%	59.23%
		0.001	62.02%	56.46%	46.16%	56.68%
	0.25	0.01	64.82%	69.06%	63.37%	62.08%
		0.001	56.29%	58.01%	55.98%	48.20%
	0.5	0.01	63.95%	67.90%	62.03%	67.22%
		0.001	60.41%	54.19%	55.33%	56.87%
Ne	0	0.01	49.18%	59.63%	55.45%	56.66%
		0.001	57.21%	61.99%	61.73%	56.10%
	0.25	0.01	43.16%	53.46%	62.39%	70.17%
		0.001	54.50%	63.74%	63.56%	63.88%
	0.5	0.01	64.53%	62.93%	67.56%	41.81%
		0.001	57.69%	52.14%	45.90%	56.68%

Tablica 5.2: Tablica parametara modela i postotka točnosti modela (obojeni dio) za period od 5 dana

Filtrirano	Dropout	Stopa učenja	Epohe			
			300	500	700	1000
Da	0	0.01	56.61%	51.83%	51.03%	47.83%
		0.001	61.83%	61.74%	62.09%	60.50%
	0.25	0.01	51.35%	53.79%	68.00%	62.17%
		0.001	56.39%	61.46%	62.18%	61.43%
	0.5	0.01	28.25%	43.37%	43.05%	41.41%
		0.001	61.23%	59.36%	59.95%	61.41%
Ne	0	0.01	60.09%	53.92%	52.39%	63.21%
		0.001	62.08%	62.83%	53.37%	58.44%
	0.25	0.01	41.33%	64.63%	62.27%	67.86%
		0.001	61.40%	66.47%	58.76%	63.10%
	0.5	0.01	62.92%	44.00%	38.71%	33.98%
		0.001	68.51%	57.17%	63.26%	51.01%

Tablica 5.3: Tablica parametara modela i postotka točnosti modela (obojeni dio) za period od 7 dana

6 | Zaključak

U ovome radu pokušali smo napraviti LSTM model s kojim bismo mogli imati preciznu predikciju koncentracije peludi u zraku za kratke periode od 3, 5 i 7 dana. Kada usporedimo rezultate koje smo dobili, možemo ocijeniti naš model kao dobar, iako smo primijetili da ima mjesta za napredak.. Budući da smo dobili dva najbolja rješenja od ukupno tri, iz skupa podataka u kojem podaci nisu bili filtrirani, ne možemo sa sigurnošću reći da je filtriranje podataka manje uspješno. Također, trebamo uzeti u obzir da je proces treniranja bio brži kada smo koristili filtrirane podatke. Isto tako, to vrijedi i za stope učenja i broj epoha. Međutim, možemo primijetiti da je dodavanje dropout sloja poboljšalo naš model, jer su sva tri najbolja rezultata došla iz modela u kojima je primijenjen dropout.

Ako želimo dalje poboljšati rezultate, možemo isprobati različite stope učenja između 0.001 i 0.01. Također, možemo eksperimentirati s drugim funkcijama gubitka, povećati ili smanjiti veličinu skrivenog sloja, ili čak promijeniti linearni sloj. Dodatno vrijedi pokušati dodati početne težine, stanje ćelije i skriveno stanje, jer ako s dropout-om dobijemo bolje rezultate, to znači da su neki podaci manje utjecajni u odnosu na druge.

Iako ne možemo donijeti konkretan zaključak o tome što je najviše utjecalo na konačne rezultate, to je vjerojatno zato što su naši podaci ponavljaju oblik sličnih intervala na kratkom periodu. Najvjerojatnije bi nam pomoglo kada bismo imali više podataka, ali iz različitih izvora.

Literatura

[1] Charu C. Aggarwal, *Neural Networks and Deep Learning*, Springer International Publishing AG, part of Springer Nature 2018

[2] Filippo Maria Bianchi, Enrico Maiorino, Michael C. Kampffmeyer, Antonello Rizzi, Robert Jenssen, *Recurrent Neural Networks for Short-Term Load Forecasting*, SpringerBriefs in Computer Science, 2017

[3] Filippo Maria Bianchi, Enrico Maiorino, Michael C. Kampffmeyer, Antonello Rizzi, Robert Jenssen, *An overview and comparative analysis of Recurrent Neural Networks for Short Term Load Forecasting*, SpringerBriefs in Computer Science, 2017

[4] Nithin Buduma, Nikhil Buduma, Joe Papa, *Fundamentals of Deep Learning*, O'Reilly Media, Inc., 2017

[5] Hardik Goel, Igor Melnyk, Arindam Banerjee, *R2N2: Residual Recurrent Neural Networks for Multivariate Time Series Forecasting*, September 12, 2017

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, The MIT Press; Illustrated edition (November 18, 2016)

[7] Klaus Greff, Rupesh K. Srivastava, Jan Koutn´ik, Bas R. Steunebrink, Jurgen Schmidhuber, *LSTM: A Search Space Odyssey*, IEEE Transactions on Neural Networks and Learning Systems 28, 2016

[8] Naylani Halpern-Wight, Maria Konstantinou, Alexandros G. Charalambides, Angèle Reinders, *Training and Testing of a Single-Layer LSTM Network for Near-Future Solar Forecasting*, Applied Sciences, 2020

[9] Sepp Hochreiter, Jürgen Schmidhuber, *Long Short-Term Memory*, Massachusetts Institute of Technology, 1997

[10] Harrison Kinsley, Daniel Kukiela, *Introducing Neural Networks*, Harrison Kinsley, 2020

[11] Ben Kröse, Patrick van der Smagt, *An introduction to Neural Networks*, The University of Amsterdam, 1996

[12] Zachary C. Lipton, John Berkowitz, Charles Elkan, *A Critical Review of Recurrent Neural Networks for Sequence Learning*, 2015

[13] Web izvor dostupan na
https://www.realforall.com/language/en/welcome/

[14] Web izvor dostupan na https://stampar.hr/hr/peludna-prognoza

[15] Web izvor dostupan na https://pytorch.org/docs/stable/index.html

[16] Web izvor dostupan na https://pandas.pydata.org/docs/

[17] Web izvor dostupan na https://numpy.org/doc/stable/

[18] Web izvor dostupan na https://matplotlib.org/stable/index.html

[19] Web izvor dostupan na https://scikit-learn.org/stable/index.html

[20] Web izvor dostupan na https://github.com/DinoSarlija/LSTM_pellon

Sažetak

U ovome radu upoznajemo neuronske mreže osmišljene za strojno učenje, obrađujemo funkcije i koncepte koji zajedno oblikuju neuronsku mrežu. Želimo napraviti model neuronske mreže s kojim bi riješili problem preciznog predviđanja budućih događaja na temelju povijesnih podataka. Za problem predikcije koncentracije peludi u zraku koristimo LSTM model te uspoređujemo dobivene rezultate ovisno o zadanim parametrima.

Ključne riječi

Neuronske mreže, rekurentne neuronske mreže, strojno učenje, LSTM, pelud.

LSTM model for predicting pollen concentration in the air

Summary

In this paper, we introduce neural networks designed for machine learning, discuss the functions and concepts that collectively shape a neural network. We aim to create a neural network model that can solve the problem of accurately predicting future events based on historical data. For the pollen concentration prediction problem, we employ an LSTM model and compare the obtained results based on the specified parameters.

Keywords

Neural networks, recurrent neural networks, machine learning, LSTM, pollen.

Životopis

Rođen sam 8.9.1997. godine u Osijeku. Pohađao sam Osnovnu školu "Retfala" u Osijeku te sam nakon završetka upisao I. gimnaziju Osijek. Nakon što sam završio I. gimnaziju upisujem sveučilišni preddiplomski studij Matematike i računarstva na Odjelu za matematiku Sveučilišta Josipa Jurja Strossmayera u Osijeku 2017. godine. Na preddiplomskom studiju u 2020. godini sudjelujem u projektu "Hrvatska pamet Hrvatskoj" te osvajam prvo mjesto. Preddiplomski studij završavam 2020. godine sa završnim radom na temu "Usporedba metoda za kalibraciju kamera" te iste godine upisujem diplomski studij Matematike, smjer Matematika i računarstvo na Odjelu za matematiku u Osijeku.