Rekurzivne neuronske mreže

Vučković, Lea

Sveučilište u Rijeci - Fakultet za matematiku
Diplomski studij Diskretna matematika i primjene

Lea Vučković

Rekurzivne neuronske mreže

DIPLOMSKI RAD

Rijeka, svibanj 2024.

Sveučilište u Rijeci - Fakultet za matematiku
Diplomski studij Diskretna matematika i primjene

Lea Vučković

Rekurzivne neuronske mreže

DIPLOMSKI RAD

Mentor: doc. dr. sc. Sanda Bujačić Babić
Kolegij: Strojno učenje

Rijeka, svibanj 2024.

Sadržaj

1 Uvod **2**

2 O biološkim i umjetnim neuronima **3**

3 Neuronske mreže **5**
3.1 Perceptron 5
3.2 Arhitektura neuronske mreže 9
3.3 Načini učenja neuronske mreže 12
3.3.1 Nadzirano učenje neuronske mreže 12
3.3.2 Nenadzirano učenje neuronske mreže 13

4 Vrste neuronskih mreža **14**
4.1 Unaprijedna neuronska mreža (Feed Forward neural network) . 14
4.2 Višeslojni perceptron (eng. Multilayer perceptron) 16
4.3 Konvolucijska neuronska mreža (eng. Convutional neural network) 17
4.4 Radijalne neuronske mreže (eng. Radial-basis function networks) 18
4.5 Modularne neuronske mreže (eng. Modular neural network) . 19

5 Rekurzivne neuronske mreže (eng. Recursive neural network) **19**
5.1 Rekurentne neuronske mreže (eng. Recurrent neural network) 22
5.2 Mreže dugog kratkoročnog pamćenja (eng. Long short-term memory) 28

6 Primjer programske realizacije rekurentne neuronske mreže **31**
6.1 Definicija dionica 32
6.2 Baza podataka 32
6.3 Analiza podataka 34
6.4 Programski kôd 48

7 Zaključak **51**

Ključne riječi

Biološki neuron, umjetni neuron, perceptron, neuronska mreža, učenje neuronske mreže, rekurzivne neuronske mreže, rekurentne neuronske mreže.

Sažetak

U diplomskom radu uvode se osnovni matematički koncepti potrebni za razumijevanje i primjenu neuronskih mreža.
Detaljno se opisuje način rada neuronske mreže te proces učenja iste na konkretnim podacima. Neuronske mreže se danas izrazito često primjenjuju u različitim djelatnostima te su se iz tog razloga osmislile različite arhitekture neuronskih mreža, s ciljem njihove optimalne prilagodbe specifičnostima konkretnih problema. U radu su predstavljeni najčešći tipovi neuronskih mreža te je poseban naglasak stavljen na rekurzivne i rekurentne neuronske mreže. Njihov princip rada je prikazan pomoću primjera te je za kraj detaljno opisana programska realizacija rekurentne neuronske mreže u programu Python.

1 Uvod

Umjetna inteligencija i strojno učenje predstavljaju revolucionarne inovacije u području informacijske tehnologije koje su uvelike promjenile način na koji primjenjujemo "računalnu inteligenciju". Takvi sustavi su osmišljeni i realizirani tako da naizgled imaju sposobnost učenja i donošenja odluka bez eksplicitnog programiranja te imaju široku primjenu u raznim područjima industrije, zdravstva, znanosti, itd. U ovom radu ćemo se upoznati s osnovnim pojmovima umjetne inteligencije, a posebice strojnog učenja te ćemo uvesti pojmove kao što su simboličko razmišljanje, modeli i vrste strojnog učenja, neuronske mreže i tipovi neuronskih mreža, itd.
Inspiracija iz prirode je temeljna ideja mnogih projekata koji se kasnije, svojim razrađivanjem, postupno udaljavaju od svoje prvobitne ideje i poprima oblik konačnog rezultata. Priroda nam je, vrlo često, glavni uzor tako da se često oslanjamo na intrigantne i kvalitetne prirodne sustave čije karakteristike želimo implementirati u okviru našeg djelokruga. Jedan takav nepresušan uzor i inspiracija je ljudski mozak koji je i danas, nakon brojnih revolucionarnih otkrića, većim djelom ostao nepoznanica ali je zato istovremeno bio velika inspiracija brojnim izumima, među kojima je i model neuronske mreže. Neuronska mreža je jedan od modela strojnog učenja koji ima ključnu ulogu u mnogim aplikacijama kao što su prepoznavanje slika, obrada prirodnog jezika, prevođenje jezika na neki drugi izabrani jezik, itd. Osnovna građevna jedinica neuronske mreže je umjetni neuron koji je nastao po uzoru na biološki neuron biološke neuronske mreže.
Neuronske meže su se počele razvijati još 1940-ih godina. Dva znanstvenika Warren McCullough i Walter Pitts[1] su predstavili prvi formalni model neurona, nakon čega je Frank Rosenblatt[2] razvio perceptron, jednostavan model koji ima jedan sloj neurona i može rješavati probleme binarne klasifikacije. Nakon 20 godina, 1970-ih, interes za neuronsku mrežu je pao zbog velikih ograničenja sposobnosti perceptrona. Posljedica pada interesa bila je objava knjige "Perceptrons" [3] koju su objavili Marvin Minsky i Seymour Papert u kojoj su detaljnije objasnili pojam perceptrona, njegove sposobnosti i ograničenja. Između 1980. i 1990. godine pojavljuju se novi modeli poput višeslojnih perceptrona, čime su se prijašnja ograničenja jednosloj-

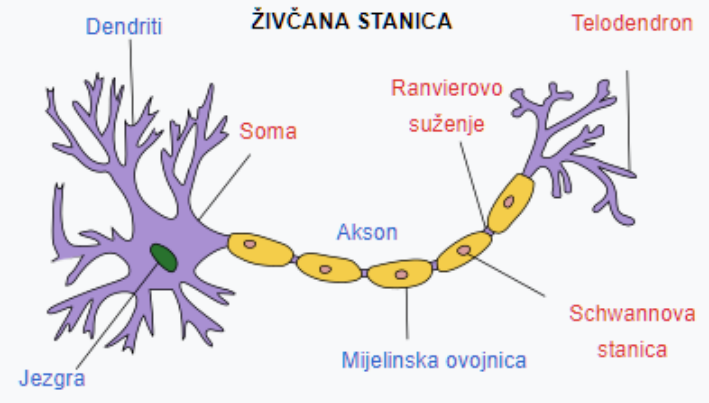
[1]Godine 1943. američki neurolog i kibernetičar Warren McCulloch sa Sveučilišta Illinois u Chicagu te logičar i kognitivni psiholog Walter Pitts objavili su "Logički račun ideja neizbježnih u živčanoj aktivnosti", opisujući "McCulloch - Pitts neuron", odnosno prvi matematički model neuronske mreže. [1]

[2]Frank Rosenblatt (11. srpnja 1928. - 11. srpnja 1971.) bio je američki psiholog zapažen na području umjetne inteligencije. Ponekad ga se naziva ocem dubokog učenja zbog njegovog pionirskog rada na umjetnim neuronskim mrežama. [2]

nog perceptrona reducirala te su takvi složeniji modeli postali učinkovitiji te prilagodljiviji na složenije zadatke. Nedugo nakon novih otkrića naišlo se na nove probleme poput problema nestajućeg gradijenta 2000-ih godina. Problem nestajućeg gradijenta je fenomen koji se javlja tijekom treniranja dubokih neuronskih mreža, gdje gradijenti koji se koriste za ažuriranje parametara neuronske mreže postaju izuzetno mali ili "nestaju". Time ne postoji značajni napredak u poboljšanju modela neuronske mreže prilikom njezinog treniranja. [4] Taj problem se pojavio prilikom treniranja neuronske mreže. Također su se razvile i tehnike reduciranja tog problema koji se i danas može pojaviti prilikom treniranja raznih modela te je potrebno obratiti pažnju na isti. Prekretnicu razvoja neuronskih mreža predstavlja uspjeh na natjecanju ImageNet gdje su Geoffrey Hinton i njegovi suradnici predstavili konvolucijsku neuronsku mrežu kao vrstu neuronske mreže. [5] Nakon toga, neuronske mreže postaju sve popularnije i budi se sve veći interes za njihov razvoj i impementaciju. Napredak i razvoj neuronskih mreža aktualan je i danas. Za specifične probleme se razvijaju odgovarajuće vrste neuronskih mreža stoga je nastala i rekurzivna neuronska mreža te rekurentna neuronska mreža, posebne vrste neuronskih mreža koje uključuju vremensku komponentu. Motivacija za nastanak takvih neuronskih mreža je riješavanje problema koji u svojim okvirima uključuju i vremensku komponentu, odnosno tijek vremena.

2 O biološkim i umjetnim neuronima

Inspiracija za umjetne neurone, osnovne sastavne jedinice svake neuronske mreže, potekla je iz prirode, preciznije iz biološkog neurona. Za razumijevanje rada neuronske mreže, nužno je razumjeti kako radi biološki neuron. Biološki neuron je sporiji od umjetnog neurona, ali brojnost neurona u mozgu može donekle nadoknaditi brzinu.
Biološki neuoron u mozgu je posebna i ujedno najsloženija vrsta stanice u organizmu. Živčani sustav sastoji se od preko 86 milijardi međusobno povezanih neurona. Biološki neuron građen je od dendrita, tijela stanice i aksona. Dendriti su kratki produžeci koji s osjetilnih organa šalju podražaj u tijelo stanice. Tijelo stanice sastoji se od jezgre, kromosoma i jezgrice. Akson je duži produžetak neurona. On može biti po nekoliko puta duži od tijela stanice. Biološki neuron prima kratke elektroničke podražaje i pretvara ih u signale. Kada neuron primi određeni broj signala od drugih neurona, proizvede vlastiti signal. Svaki neuron je sastavni dio neuronske mreže koja sadrži milijarde neurona. [6]



Slika 1: Građa biološkog neurona [6]

Umjetni neuron je nastao po uzoru na biološki neuron. Njega nazivamo model umjetnog neurona (*eng. Threshold Logic Unit* (TLU)). Ulazne podatke, kojih ima p, označimo s $x_1, x_2, ..., x_p$. Vezama između neurona pridružene su težine (eng. weight) koje označavamo redom s $w_1, w_2, ..., w_p$. Težine su realni brojevi kojima možemo direktno utjecati na izlazne vrijednosti, odnosno rezultate koje daje neuronska mreža. Težinu možemo shvatiti kao jačinu veze, odnosno utjecaj određene veze u neuronskoj mreži. Veća vrijednost težine će značajnije doprinositi vrijednosti izlaznog podatka. TLU računa sumu umnožaka odgovarajućih težina i ulaznih podataka

$$u = w_1x_1 + ... + w_px_p$$

te dobivena suma postaje argument *aktivacijske funkcije* $\varphi : \mathbb{R} \to \mathbb{R}$ čija je vrijednost finalni rezultat neuronske mreže. Vrlo često u izračunu sudjeluje i prag ili pristranost (eng. bias) kojeg uvodimo kao ulazni podatak vrijednosti -1 i težine Θ. Izlazni podatak možemo zapisati i u vektorskom obliku:

$$y = \varphi(u - \theta).$$

Umjetna neuronska mreža je masivni paralelni procesor koji služi za učenje. Sličnost s biološkom mrežom je ta što ima sposobnost učenja preko veza između neurona.
Neuronske mreže su vrlo često višeslojne te se vrijednosti aktivacijskih funkcija prenose na sljedeći neuron, sve do posljednjeg sloja. Aktivacijska funkcija određuje koji će se neuroni aktivirati u sljedećem sloju, a koji neće. Postoji mnogo vrsta aktivacijskih funkcija i načešće ih dijelimo u tri vrste; step funkcije, linearne funkcije i nelinearne funkcije. Najpoznatija step funkcija

je funkcija $f : \mathbb{R} \to \{0, 1\}$ definirana kao

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

Primjer nelinearne aktivacijske funkcije je sigmoidna funkcija $\sigma : \mathbb{R} \to \langle 0, 1 \rangle$ definirana kao:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}.$$

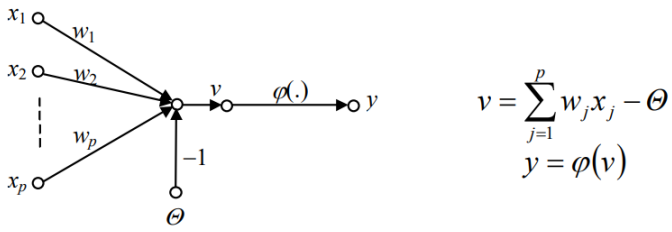
Linearna aktivacijska funkcija ne doprinosi rezultatu, odnosno vrijednost aktivacijske funkcije proporcionalan je iznosu sume. Primjer linearne aktivacijske funkcije je identična funkcija $g : \mathbb{R} \to \mathbb{R}$,

$$g(x) = x.$$

3 Neuronske mreže

3.1 Perceptron

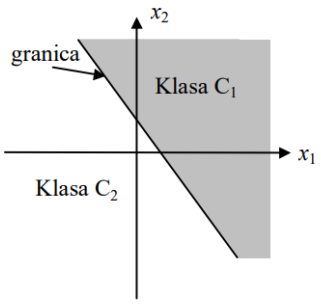
Perceptron je jedna od najjednostavnijih struktura neuronskih mreža za klasifikaciju uzoraka [7]. Osmislio ju je Frank Rosenblatt 1957. godine. Perceptron se sastoji od jednog neurona. Promotrimo jednoslojni perceptron prikazan na slici 2.



Slika 2: Građa jednoslojnog perceptrona [7]

Perceptron služi kako bi klasificirao vektor $\mathbf{x} = [x_1, ..., x_p]^T$ u jednu od dvije klase koje možemo označiti s C_1 i C_2. Za ulazni vektor perceptron daje izlaznu informaciju koja služi kao klasifikator za dani vektor. U ovom slučaju, područja su odvojena hiperravninom koja je definirana izrazom $\sum_{j=1}^{p} w_j x_j - \Theta = 0$, gdje su $w_1, ..., w_p, \Theta$ težine, a $x_1, ..., x_p$ ulazni podaci. Promotrimo primjer klasifikacije u dvije dimenzije. Na slici 3 je prikazano

područje klasifikacije u dvije dimenzije, gdje granicu između dvije klase predstavlja pravac $w_1x_1 + w_2x_2 - \Theta = 0$. Ako se točka nalazi u sivom području, ona priprada klasi C_1. U suprotnom, pripada klasi C_2.



Slika 3: Područje klasifikacije u dvije dimenzije [7]

Primjer 3.1 *Želimo odlučiti hoćemo li ići na koncert. Postoji nekoliko kriterija koji mogu utjecati na našu odluku. Odlučili smo uzeti u obzir nama važne kriterije: je li izvođač dobar, je li vrijeme povoljno, hoće li prijatelj ići, hoće li se posluživati hrana i hoće li se posluživati piće. Svaki od tih kriterija možemo označiti redom s x_1, x_2, x_3, x_4, x_5 i mogu imati vrijednost 0 ili 1 što odgovara poruci je li kriterij zadovoljen ili nije, redom. Važnost pojedinog kriterija izražavamo u težinama. Onaj kriterij koji nam je važniji imati će veću težinu. Prikažimo ove podatke u tablici:*

Kriterij	Ulazni podatak	Težina
Izvođač je dobar	$x_1 = 0$ ili $x_1 = 1$	$w_1 = 0.7$
Vrijeme je povoljno	$x_2 = 0$ ili $x_2 = 1$	$w_2 = 0.6$
Prijetelj će ići	$x_3 = 0$ ili $x_3 = 1$	$w_3 = 0.5$
Posluživat će se hrana	$x_4 = 0$ ili $x_4 = 1$	$w_4 = 0.3$
Posluživat će se piće	$x_5 = 0$ ili $x_5 = 1$	$w_5 = 0.4$

Tablica 1: Tablica podataka

Pomoću težina vidimo da je za odluku najbitniji izvođač, a najmanje je bitno hoće li se posluživati hrana. Zanima nas kako ćemo doći do odgovora. Perceptron zahtijeva da unaprijed odredimo prag. U ovom primjeru uzmimo da je to 1.5. Perceptron vraća logičku vrijednost True ako je suma veća od zadanog praga koji iznosi 1.5. Najprije se izračuna suma umnožaka ulaznih

vrijednosti i odgovarajućih težina. Uzmimo da je izvođač dobar $(x_1 = 1)$*, vrijeme nije povoljno* $(x_2 = 0)$*, prijatelj ide na koncert* $(x_3 = 1)$*, hrana se neće posluživati* $(x_4 = 0)$ *i piće će se posluživati* $(x_5 = 1)$*. Dakle, dobivamo:* $x_1 \cdot w_1 = 1 \cdot 0.7 = 0.7, \quad x_2 \cdot w_2 = 0 \cdot 0.6 = 0, \quad x_3 \cdot w_3 = 1 \cdot 0.5 = 0.5, \quad x_4 \cdot w_4 = 0 \cdot 0.3 = 0, \quad x_5 \cdot w_5 = 1 \cdot 0.4 = 0.4.$
Sada sumiramo dobiveno:

$$0.7 + 0 + 0.5 + 0 + 0.4 = 1.6.$$

Budući je suma jednaka 1.6*, dobiveni zbroj je veći od postavljenog praga, izlazni rezultat je True i upućuje na to da ćemo ići na koncert. Zadavanje praga je relativno, što bi značilo da, što je manji prag, imamo veću želju za odlaskom na koncert.*

Perceptron je najjednostavnija vrsta neuronske mreže koja se sastoji od samo 1 neurona. Prednost perceptrona je njegova jednostavnost. Perceptronom se mogu modelirati jednostavne logičke funkcije kao što su "AND" i "OR".

Primjer 3.2 *Logička funkcija XOR, odnosno, isključivo ili, ima dva ulaza i jedan izlaz. Izlaz je jedinica ako su ulazi različiti, a nula ako su ulazi jednaki. Ulazne i izlazne vrijednosti logičke funkcije XOR nalaze se u sljedećoj tablici:*

x	y	XOR(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

Tablica 2: XOR funkcija

Cilj je konstruirati neuronsku mrežu koja će računati XOR funkciju. U ovom primjeru ćemo navesti dva rješenja, jedno dano neuronskom mrežom s jednim skrivenim slojem, a drugo korištenjem jednog neurona odnosno perceptronom.

Odredimo najprije neuronsku mrežu s jednim skrivenim slojem. Neka je aktivacijska funkcija zadana kao:

$$\phi(z) = \begin{cases} 0, & z \leq 0, \\ 1, & z > 0. \end{cases}$$

Nadalje, neka su zadane matrice težina W_1 *i* W_2 *te vektori* b_1 *i* b_2*:*

$$W_1 = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 1.5 \\ -0.5 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad b_2 = \begin{bmatrix} -1.5 \end{bmatrix}.$$

Ulaz predstavlja vektor $x_1 = \begin{bmatrix} x & y \end{bmatrix}$ *te vrijedi*

$$x_2 = \phi(W_1 x_1 + b_1),$$
$$x_3 = \phi(W_2 x_2 + b_2),$$

gdje je x_3 *izlaz.*
Označimo neuronsku mrežu s $f : \mathbb{R}^2 \to \mathbb{R}$. *Naš je zadatak odrediti funkciju* f *tako da ona ima jednake vrijednosti kao i logička funkcija XOR.*
Dakle, za ulaz

$$x_1 = \begin{bmatrix} x \\ y \end{bmatrix}$$

je

$$x_2 = \begin{bmatrix} \phi(-x - y + 1.5) \\ \phi(x + y - 0.5) \end{bmatrix}$$

te daljnjim uvrštavanjem x_2 *dobivamo:*

$$f(x, y) = \phi(\phi(-x - y + 1.5) + \phi(x + y - 0.5) - 1.5).$$

Ako usporedimo izlaze dobivene funkcije f *i funkcije XOR, uočavamo da su oni jednaki.*

x	y	$XOR(x, y)$	$f(x, y)$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	1	1

Tablica 3: Usporedba XOR funkcije i funkcije f

Promjenimo li matricu težina W_1 *i stavimo da je*

$$W_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

tada neuronska mreža više neće davati iste vrijednosti kao i funkcija XOR. *Dakle, s obzirom na novu matricu* W_1, *nova neuronska mreža dana je funkcijom*

$$g(x, y) = \phi(\phi(x + y + 1.5) + \phi(x + y - 0.5) - 1.5).$$

Usporedimo u tablici vrijednosti funkcije XOR *i funkcije* g.

x	y	$XOR(x,y)$	$g(x,y)$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	1

Tablica 4: Usporedba XOR funkcije i funkcije g

Na osnovu ovog primjera možemo zaključiti da relativno mala izmjena težina mogu rezultirati značajnim razlikama u izlaznim vrijednostima neuronske mreže.

Promotrimo sada rješenje ovog problema koje dobivamo pomoću perceptrona. Aktivacijska funkcija će ostati ista, međutim dimenzija ulaza više neće biti 2, nego 3. Stoga, ulazni vektor sada ima sljedeći oblik:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix}.$$

Nadalje, sada je

$$w = \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix}, \quad b = \begin{bmatrix} -0.5 \end{bmatrix}.$$

Tražena transformacija je oblika:

$$\phi(w^T x + b).$$

Funkcija f dana je s

$$f(x_1, x_2) = \phi(x_1 + x_2 - 2x_1 x_2 - 0.5).$$

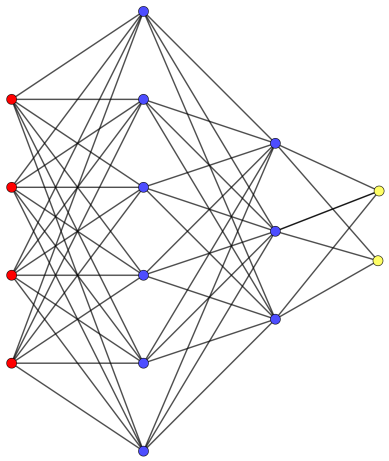
Lako se provjeri da funkcija f i logička funkcija XOR imaju jednake vrijednosti.

U ovom primjeru smo pokazali da neuronska mreža ne mora bit jedinstvena.

3.2 Arhitektura neuronske mreže

Neuronska mreža se s matematičkog gledišta može promatrati i kao funkcija jedne ili više varijabli koja se sastoji od niza stanja. U neuronskoj mreži glavnu ulogu imaju težine koje se mijenjaju, odnosno prilagođavaju s ciljem

što uspješnijeg rješavanja zadanog zadatka. Ovaj postupak se naziva *učenje neuronske mreže.*



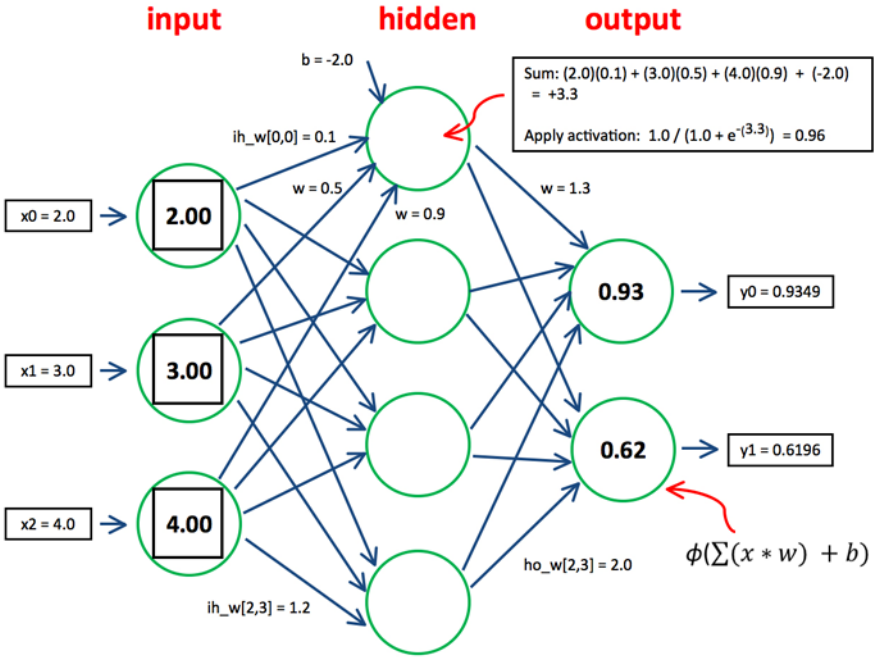
Slika 4: Neuronska mreža

Na slici 4 se nalazi grafički prikaz jedne neuronske mreže. Ulazni sloj se sastoji od neurona crvene boje, skriveni slojevi su neuroni plave boje te je izlazni sloj žute boje. Općenito, kada u neuronskoj mreži imamo nekoliko slojeva i/ili u jednom sloju nekoliko neurona, težine između neurona zapisujemo u matrice. Označimo s x_k k-ti sloj u neuronskoj mreži. Dakle, u neuronskoj mreži koja ima n slojeva, težine između dva sloja neurona prikazujemo matricom W_k gdje $k = 1, ..., n-1$. U matrici $W_k = \left[w_{i,j}\right]$, element na poziciji (i, j) je broj $w_{i,j}$ koji predstavlja težinu između i-tog i j-tog neurona, a koji je između k-tog i $k+1$-og sloja. Nadalje, kao i kod perceptrona, dodaje se dodatni ulaz u svakom sloju, kojeg nazivamo prag ili pristranost, pa vektor takvih dodatnih ulaza možemo označiti sa b_k, gdje $k = 1, ..., n-1$. Vektor b_k će imati onoliko komponenata koliko je neurona u odgovarajućem sloju. Primjerice, komponenta $b_{k,1}$ je dodatan ulaz za prvi neuron u k-tom sloju. Broj neurona iz sloja u sloj se može mijenjati.
Aktivacijske funkcije definiramo kao funkcije $\phi : \mathbb{R}^n \to \mathbb{R}^m$ takve da njihova vrijednost

$$\phi(W_k x_k + b_k)$$

postaje ulazni podatak za sljedeći sloj, tj. imamo da je $x_{k+1} = \phi(W_k x_k + b_k)$, $k = 1, ..., n-1$.

Primjer 3.3 *Promatramo jednostavan primjer neuronske mreže koja ima 3 sloja.*



Slika 5: Neuronska mreža [8]

Ulazni podaci su $x_1 = \begin{bmatrix} 2 & 3 & 4 \end{bmatrix}$*, odnosno* x_1 *je ulazni sloj. Radi jednostavnosti, pratit ćemo račun jednog neurona u skrivenom sloju.*
Svaki neuron u ulaznom, odnosno input sloju povezan je s tim neuronom kojeg pratimo u primjeru i težine veza su redom $w_1 = 0.1, w_2 = 0.5, w_3 = 0.9$*, što bi značilo da imamo matricu* W_1 *dimeznije* 3×4*. U toj matrici definiran je samo prvi stupac jer imamo zadane težine koje povezuju sve neurone ulaznog sloja i jedan neuron u skrivenom sloju. Stoga, matrica* W_1 *izgleda ovako:*

$$W_1 = \begin{bmatrix} 0.1 & w_{1,2} & w_{1,3} & w_{1,4} \\ 0.5 & w_{2,2} & w_{2,3} & w_{2,4} \\ 0.9 & w_{3,2} & w_{3,3} & w_{3,4} \end{bmatrix}$$

Dodatni vektor u svakom sloju predstavlja dodatne ulaze za odgovarajući neuron. U ovom primjeru, za promatrani neuron, dodatni ulaz iznosi -2*. Dodatni vektor* b_1 *ima četiri komponente jer se u prvom sloju nalaze 4 neurona. Također zbog jednostavnosti, u ovom primjeru ima poznatu samo prvu komponentu koja se odnosi na prvi neuron u tom sloju. Dakle*

$$b_1 = \begin{bmatrix} -2 & b_{1,2} & b_{1,3} & b_{1,4} \end{bmatrix}.$$

Sada, neuronska mreža preuzima te podatke, množi s odgovarajućim težinama i šalje taj rezultat sljedećem neuronu. Uzimajući u obzir da računamo samo jednu komponentu tog izraza imamo:

$$x_1 \cdot w_{1,1} + x_2 \cdot w_{1,2} + x_3 \cdot w_{1,3} + (-2) = 2 \cdot 0.1 + 3 \cdot 0.5 + 4 \cdot 0.9 + (-2) = 3.3.$$

Dobiveni rezultat se šalje, kao argument, aktivacijskoj funkciji koja računa nove vrijednosti u svakom neuronu. Na primjer, neka se u ovom primjeru koristi sigmoida [3]*:*

$$\phi(x) = \frac{1}{1 + \exp^{-x}}.$$

Uvrštavanjem argumenta 3.3 u aktivacijsku funkciju, dobivamo rezultat $\phi(3.3) = 0.96$*. Dakle, sljedeći sloj je oblika*

$$x_2 = \begin{bmatrix} 0.96 & x_2 & x_3, & x_4 \end{bmatrix}.$$

Analogni postupak se provedi za preostale neurone u tom sloju. Dobiveni rezultati u svakom neuronu sada predstavljaju ulazne podatke za sljedeći sloj neurona. Postupak se nastavlja sve dok ne dođemo do izlaznog sloja.

3.3 Načini učenja neuronske mreže

3.3.1 Nadzirano učenje neuronske mreže

Proces učenja neuronske mreže podrazumijeva određivanje optimalnih težina tako da izlazni podatak neuronske mreže bude to sličniji očekivanom rješenju. Kao i u većini dobro poznatih modela strojnog učenja, tako se i kod neuronskih mreža, najčešće skup podataka kojeg koristimo organizira u skup podataka za trening i skup podataka za testiranje. U trening skupu se nalaze podaci oblika (x, y), gdje je x nezavisna varijabla/input, a y je zavisna varijabla/output. Neuronska mreža prima input x i računa izlaznu vrijednost $f_w(x)$[4]. Cilj je postići $f_w(x) \approx y$, odnosno da neuronska mreža izračuna izlazni podatak što sličniji točnom podatku koji se nalazi unutar trening skupa podataka. Kako bi mogli utvrditi koliko je točna svaka izlazna vrijednost neuronske mreže u usporedbi s odgovarajućim vrijednostima iz

[3] Funkcija koju nazivamo logistička funkcija, također je poznata kao sigmoidna funkcija ili sigmoida

[4] Neuronsku mrežu matematički možemo interpretirati kao funkciju, pa je označavamo sa f_w

trening skupa podataka, računamo sumu kvadrata greške koju označavamo s $E(w)$ i definiramo:

$$E(w) = \sum_{(x,y)\in X} (f_w(x) - y)^2,$$

gdje je X trening skup podataka. Nastojeći da ta suma bude manja, što znači da je neuronska mreža u generalnom smislu izračunala izlazne vrijednosti tako da su one relativno bliske vrijednostima iz trening skupa, dolazimo do optimizacijskog problema, odnosno nastojanja da se odredi minimum sume kvadrata greške:

$$\min_w E(w) = \sum_{(x,y)\in X} (f_w(x) - y)^2.$$

Ovaj optimizacijski problem najčešće nije lako riješiti zbog velikog broja parametara, a problem dodatno kompliciraju parametri iz skrivenih slojeva koji imaju složen utjecaj na izlaznu vrijednost. Postupak učenja neuronske mreže u okviru kojeg se uvodi trening skup u odnosu na kojeg neuronska mreža adaptira težine te putem njih predviđa nepoznate vrijednosti, zove se nadzirano učenje.

3.3.2 Nenadzirano učenje neuronske mreže

Neuronske mreže mogu se koristiti i za generiranje novih podataka. U tom slučaju, prolazak kroz neuronsku mrežu, koju možemo označiti sa g_w, odvijao bi se unatrag. Za neku slučajnu vrijednost izlaznog podatka y generira se ulazna vrijednost x. Na primjer, za neku unesenu sliku, neuronska mreža generira novu. Za takve neuronske mreže učenje je nenadzirano. Nenadzirano učenje se može usporediti s načinom kako djeca uče, promatrajući okolinu i oponašajući pojave u njoj. Kod nenadziranog učenja baza podataka koja se koristi za treniranje algoritma, sadrži podatke koji nemaju oznaku. Oznaka se može shvatiti kao stvarna, odnosno točna ili referentna vrijednost podatka. Cilj nenadziranog učenja nije predvidjeti točnu vrijednost, već naučiti uzorak u podacima, međusobni odnos između podataka te strukturu podataka. Iz tog razloga je teško odrediti koliko algoritam dobro radi jer nema točnih rezultata za usporedbu kao što je to postojalo kod nadziranog učenja.

Pojasnimo detaljnije kako funkcionira nenadzirano učenje na neuronskoj mreži kojoj je cilj nacrtati novu sliku temeljeći je na trening skupu koji se sastoji od mnogobrojnih slika. Najprije je potrebno odrediti trening skup koji sadrži veliki broj slika. Cilj je odrediti težine tako da slučajne sličice koje generira neuronska mreža g_w budu što sličnije trening skupu. Također, potrebno je

generirati skup tkz. lažnih slika. Cilj je da te lažne slike budu što sličnije onima iz trening skupa. Označimo sa $\{z_1, ..., z_n\}$ trening skup. Dana je neuronska mreža g_w parametrizirana s težinama w te označimo s $\{x_1, ..., x_n\}$ skup lažnih slika koje je neuronska mreža generirala. Da bi se lažna slika x_1 generirala, potreban je slučajni iznos y_1 koji se daje kao ulaz neuronskoj mreži. Na taj način dobivamo da je $x_1 = g_w(y_1)$. Analogno se dobivaju preostale lažne slike. Cilj nenadziranog učenja je odrediti težine tako da skup lažnih slika bude što sličniji trening skupu. I u ovom slučaju, kao i kod nadziranog učenja, susrećemo se s optimizacijskim problemom kojeg možemo zapisati kao

$$\min_w d(\{x_1, ..., x_n\}, \{z_1, ..., z_n\}).$$

S obzirom da znamo da lažne slike ovise o neuronskoj mreži, odnosno težinama unutar mreže, problem možemo zapisati u sljedećem obliku

$$\min_w d(\{g_w(y_1), ..., g_w(y_n)\}, \{z_1, ..., z_n\}).[9]$$

Funkcija $d(\{x_1, ..., x_n\}, \{z_1, ..., z_n\})$ predstavlja udaljenost između skupova $\{x_1, ..., x_n\}$ i $\{z_1, ..., z_n\}$. Za određivanje udaljenosti možemo koristiti različite funkcije, ovisno o pristupu, no u ovom radu to nećemo detaljnije razmatrati. Tip učenja neuronske mreže najviše ovisi o vrsti zadatka za koji će se određena neuronska mreža koristiti. Ako je cilj neuronske mreže predviđati vrijednosti, učenje neuronske mreže bit će nadzirano. Međutim, ako je cilj neuronske mreže generirati nove vrijednosti, neovisno o ulaznim podacima, učenje takve neuronske mreže bit će nenadzirano.

4 Vrste neuronskih mreža

U ovom poglavlju predstavit ćemo nekoliko vrsta neuronskih mreža te objasniti razlike među njima. Svaka vrsta neuronske mreže se može koristiti za određene probleme. Međutim, može se dogoditi da isti problem može riješiti više različitih neuronskih mreža.

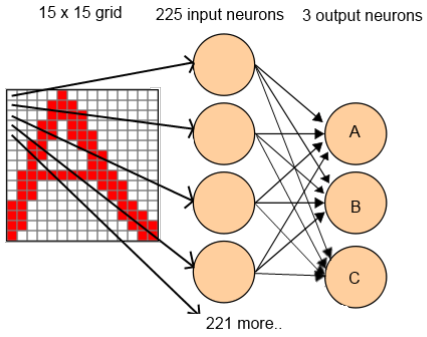
4.1 Unaprijedna neuronska mreža (Feed Forward neural network)

Unaprijedna neuronska mreža (eng. Feed Forward Neural Network) jedan je od jednostavnijih primjera neuronske mreže jer podaci prolaze samo u jednom smjeru kroz mrežu. Grafički prikaz primjera unaprijedne neuronske mreže možemo vidjeti na slici 4. Skriveni slojevi mogu, ali i ne moraju postojati. Ovisno o broju skrivenih slojeva, neuronske mreže se klasificiraju u

jednoslojne unaprijedne ili višeslojne unaprijedne neuronske mreže. Kompleksnost neuronske mreže ovisi o broju skrivenih slojeva. Budući ne postoji mogućnost prolaska podataka unazad (od zadnjeg prema prvom sloju neuronske mreže), jednom izrabrane težine ostaju fiksirane u ovom slučaju. Ulazne podatke množimo s težinama i taj rezultat predstavlja argument aktivacijske funkcije koja daje izlaznu vrijednost. Aktivacijska funkcija, ovisno o zadanom pragu (najčešće 0), daje vrijednost 1 ako je zadovoljen prag (vrijednost argumenta aktivacijske funkcije je veća od 0) i time je aktiviran neuron koji se nalazi u sljedećem sloju neuronske mreže. Primjerice, na slici 5 je grafički prikazana neuronska mreža koja ima ulazni sloj, jedan skriveni sloj i izlazni sloj. Rezultat aktivacijske funkcije u prvom neuronu skrivenog sloja je 0.96 što je veće od 0 stoga će taj neuron biti aktiviran te će biti uključen u daljnji izračun. U suprotnom, taj neuron će biti zanemaren i u tom slučaju će rezultat biti -1. [10]

Jedna od velikih prednosti unaprijedne neuronske mreže je njezina jednostavnost prilikom dizajniranja i implementacije te širok spektar primjene ovakvog modela strojnog učenja. Njezin jednosmjerni prolazak podataka kroz mrežu omogućava veću brzinu pri računanju. Također, ovakav tip neuronske mreže nije osjetljiv na šumove u podacima jer ih lako detektira te zanemaruje, dajući precizne rezultate. Jednostavnost neuronske mreže, koliko je prednost, toliko je i nedostatak. No, za složenije probleme koji zahtijevaju složeniji model u okviru kojeg se omogućava dvosmjernost protoka podataka potrebno je tražiti druga kompleksnija rješenja.

Unaprijedne neuronske mreže mogu se koristiti za probleme jednostavnijih klasifikacija podataka. Na primjer, mogu se koristiti za prepoznavanje rukom pisanih znamenki, lica i govora.



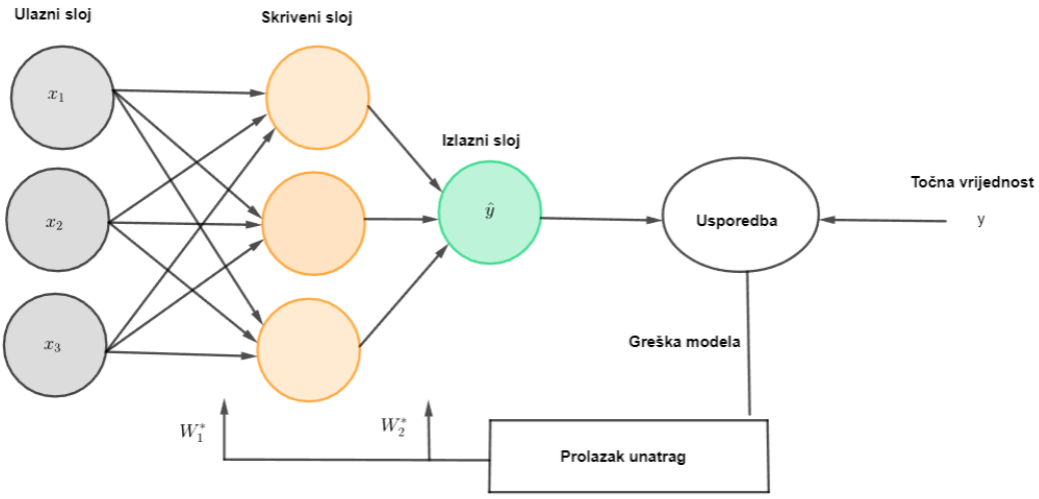
Slika 6: Unaprijedna neuronska mreža [11]

Primjerice, na slici 6 je prikazana unaprijedna neuronska mreža koja prepoznaje slova na slici. Svaka slika se sastoji od piksela i svaki piksel je jedna ulazna vrijednost unaprijedne neuronske mreže. U ovom slučaju, slika ima 225 piksela, stoga se ulazni sloj sastoji od 225 neurona. Izlazni sloj se sastoji od 3 neurona koji, kao rezultat, daje slovo prepoznato na slici, a u ovom primjeru je moguće detektirati jedno od tri slova, A, B, C.

4.2 Višeslojni perceptron (eng. Multilayer perceptron)

Višeslojni perceptron (eng. Multilayer Perceptron) je neuronska mreža u kojoj su najčešće, ali ne nužno uvijek, svi neuroni u sloju povezani sa svim neuronima sljedećeg sloja. U tom slučaju je zovemo potpuno povezana neuronska mreža. Višeslojni preceptron ne mora uvijek biti potpuno povezan, ali je poželjno imati što više povezanih neurona radi preciznijih rezultata. Za razliku od unaprijedne neuronske mreže, osim prolaska unaprijed radi računanja rezultata modela, postoji i prolazak unatrag. U okviru neuronskih mreža koje su konstruirane tako da se protok podataka može realizirati unatrag (od izlaznog prema početnom sloju), taj je smjer ključan budući omogućuje da se kritički promatraju izlazni podaci s ciljem smanjenja greške modela i to ukoliko se utvrdi da se izlazne podatke treba poboljšati uspoređujući ih s trening podacima nakon unaprijednog prolaska kroz neuronsku mrežu. Greška modela se računa tako da se uspoređuju izlazni podaci mreže i točni podaci. [10]
Na slici 7 je prikazan višeslojni perceptron koji ima jedan skriveni sloj. Ulazni i skriveni slojevi imaju po tri neurona, a izlazni sloj jedan neuron. U matricama W_1 i W_2 pohranjene su odgovarajuće težine veza između neurona u različitim slojevima. Prolaskom unaprijed, izračunava se rezultat $\hat{y}$ koji se uspoređuje sa točnom vrijednosti y. Izračunava se greška modela koja se želi minimizirati. Polazak unatrag omogućuje promjenu vrijednosti težina odnosno dobivaju se nove matrice težina W_1^* i W_2^* koje doprinose boljem izlaznom rezultatu.

Prednost višeslojnog perceptrona je ta da se njime mogu modelirati kompleksniji problemi, ali time je i kreiranje takve mreže kompleksnije. Višeslojni perceptron može biti spor prilikom računanja, ukoliko ima veliki broj skrivenih slojeva. Višeslojni perceptron se može koristiti prilikom prepoznavanja govora i prevođenja. Ukoliko imamo problem kompleksnije klasifikacije, višeslojni perceptron može biti dobar izbor za rješavanje tog problema.



Slika 7: Višeslojni perceptron

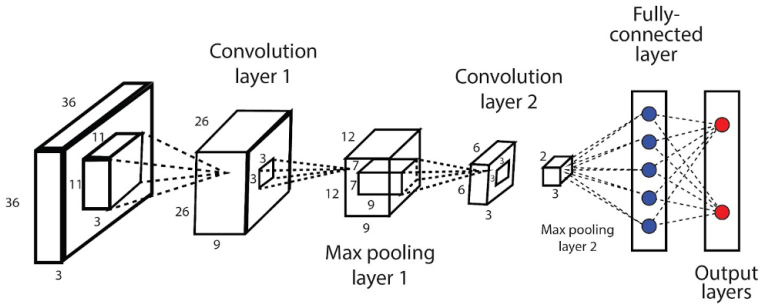
4.3 Konvolucijska neuronska mreža (eng. Convutional neural network)

Konvolucijska neuronska mreža (eng. Convutional neural network) sastoji se od nekoliko vrsta slojeva neurona. Prvi sloj, nakon ulaznog sloja, zove se konvolucijski sloj. U konvolucijskom sloju se zadržavaju važne značajke slike (npr. rubovi) konvolucijskom operacijom[5]. Konvolucijskih slojeva može biti i više, ovisno o složenosti mreže. Cilj je da sliku iz RGB skale pretvorimo u sliku sive skale. Nakon konvolucijskog sloja se nalazi sloj sažimanja (eng. pooling layer). U tom sloju također postoje operacije koje su zaslužne za reduciranje svojstava. Sloj sažimanja obrađuje podatke koji su proizašli iz konvolucijskog sloja uzimajući u obzir dominantne značajke. Posljednji sloj neuronske mreže ovog tipa je klasifikacijski sloj. Dominantne značajke iz sloja sažimanja postaju ulazni podaci višeslojnog perceptrona. Postoji nekoliko vrsta konvolucijskih neuronskih mreža: LeNet, AlexNet, VGGNet, GoogLeNet, ResNet i ZFNet. [12]

Prednost ovog tipa neuronske mreže je njezina primjenjivost na aktualne i raznolike probleme klasifikacije. Nedostatak je njezina složenost prilikom kreiranja i korištenja. Konvolucijske neuronske mreže se primjenjuju za prepoznavanje i klasifikaciju slika i videa. Također se koriste za prepoznavanje

[5]Konvolucijska operacija je složeniji izračun koji koristi Hamardov produkt. Za više informacija može se pročitati članak [12].

govora te prevođenja.

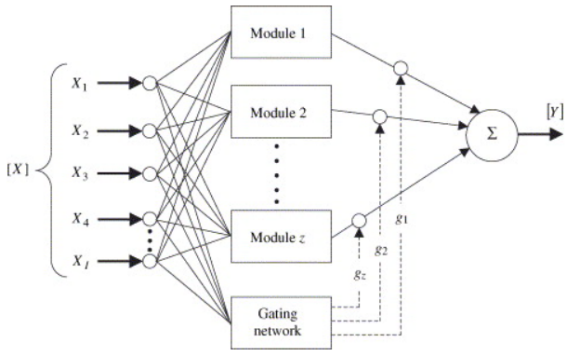


Slika 8: Konvolucijska neuronska mreža [13]

4.4 Radijalne neuronske mreže (eng. Radial-basis function networks)

Radijalne neuronske mreže (eng. Radial-basis function networks, RBF) sadrže ulazni sloj, sloj RBF neurona te izlazni sloj. Arhitektura RBF neuronske mreže nešto je drukčija od većine standardnih neuronskih mreža. Broj neurona u skrivenom sloju mnogo je veći od broja neurona u ulaznom sloju, ali ne može biti veći od broja podataka u trening skupu. Cilj RBF neuronske mreže je usporediti ulazne podatke s podacima iz trening skupa. Svaki neuron u skrivenom sloju sadrži prototip. Ulazni podatak se u skrivenom sloju uspoređuje s prototipom svakog neurona i traži se neuron kojemu je on najsličniji. Taj prototip najčešće predstavlja neki podatak iz trening skupa. Sličnost između podatka i prototipa računa se euklidskom udaljenošću. Izlazni sloj predstavlja standardni klasifikacijski sloj u kojemu se koristi aktivacijska funkcija kako bi se klasificirao ulazni podatak. [10]
Velika prednost ove vrste neuronskih mreža je što mogu modelirati kompleksne i nelinearne veze između podataka. Primjerice, vrlo su primjenjive u aplikacijama za prepoznavanje govora i slika gdje mogu prepoznati kompleksni uzorak, poput slike ili glasovnog zapisa, koji predstavlja ulaznu vrijednost. Također, mogu biti korisne prilikom aproksimacija nelinearnih funkcija koje se koriste u raznim domenama poput fizike, inžinjestva i slično. Vrlo je korisna u detektiranju anomalija u uzorku podataka. Takva detekcija može biti korisna u financijskom sektoru za detektiranje prevara prilikom transakcija. [14] Nedostatak radijalnih neuronskih mreža je velika osjetljivost na dimenziju podataka. [15]

4.5 Modularne neuronske mreže (eng. Modular neural network)



Slika 9: Modularna neuronska mreža [10]

Modularne neuronske mreže (eng. Modular neural network) su neuronske mreže koje se sastoje od nekoliko (vrsta) neuronskih mreža. Složeniji problemi se raščlanjuju na podprobleme i nezavisno rješavaju pri čemu ti procesi ne utječu jedni na druge. Posljedično, pojedinačni izračuni su brži i efikasniji. Shemu modularne neuronske mreže možemo vidjeti na slici 9. [10] Prednost ovih neuronskih mreža je smanjenje složenosti modela. Kombinacijom nekoliko modela može se provesti kompleksni zadatak uz manje utrošenog vremena i veće efikasnosti. [16] Modularne neuronske mreže su vrlo primjenjive u robotskim autonomnim sustavima jer mogu kontrolirati različite funkcije i dijelove robota, od senzora, preko analize do djelovanja robota. Ovaj tip neuronske mreže vrlo je primjenjiva u obradi prirodnog jezika, gdje se istovremeno može rukovati sa sintaksom, semantikom i kontekstom [17].

5 Rekurzivne neuronske mreže (eng. Recursive neural network)

Rekurzivne neuronske mreže (eng. Recursive neural network) su posebna vrsta neuronskih mreža koje imaju veliki potencijal za analizu strukturalnih podataka, primjerice HTML web sjedišta, DNA molekule, analiza fotografija, itd. Rekurzivne neuronske mreže su prikladne za regresijske i klasifikacijske probleme. Glavna prednost ovih mreža je njihova mogućnost rada s podacima koji imaju različite veličine i topologije u usporedbi s klasičnim neuronskim mrežama koje mogu analizirati podatke fiksiranih veličina. Može se reći da su grafovi fleksibilniji u smislu strukture podataka, nego vektori. U usporedbi

s vektorima, koji unaprijed imaju definiranu dužinu koju određuju uzorci u trening setu, grafovi su promjenjljive strukture, upravo zbog njihovog proizvoljnog oblika, broja vrhova i broja bridova samim time se onda i koriste u rekurzivnim neuronskim mrežama, te kao takvi mogu reprezentirati podatke promjenjive strukture i dimenzije.

Definicija 5.1 *Graf G je uređena trojka $G = (V, E, I)$, gdje je V skup vrhova, a E skup bridova, a $I \subseteq V \times E$ relacija incidencije koja svakom elementu skupa E pridružuje dva elementa iz skupa V.*

Definicija 5.2 *Usmjereni graf G je uređena trojka $G = (V, E, I)$, gdje je V skup vrhova, E skup usmjerenih bridova (lukova), a $I \subseteq V \times E$ relacija incidencije koja svakom usmjerenom bridu pridružuje uređeni par (ne nužno različitih) vrhova u G.*

Definicija 5.3 *Neka je dan usmjereni graf $G = (V, E, I)$ i neka je v proizvoljan vrh skupa V. Skup roditelja vrha v je $pa[v] = \{u \in V | (u, v) \in E\}$, a skup djece vrha v je $ch[v] = \{u \in V | (v, u) \in E\}$.*

Ulazni stupanj vrha v je broj elemenata skupa $pa[v]$, odnosno $|pa[v]|$, a izlazni stupanj vrha v je broj elemenata skupa $ch[v]$, odnosno $|ch[v]|$.
U rekurzivnoj neuronskoj mreži, trening skup se sastoji od označenih grafova. Vrhovi grafa sadrže skup svih varijabli koje su karakterizirane vektorom. Te varijable mogu biti numeričke i/ili kategorijalne. Svaki vrh sadrži dio informacije važne u rješavanju zadatka. Ako postoji brid (v, w) između vrhova v i w, tada taj brid modelira logičku vezu između dijelova informacije koje predstavljaju ti vrhovi.
Rekurzivna neuronska mreža karakterizira *prijelazne funkcije* f i *izlazne funkcije* g koje su implementirane pomoću višeslojnog perceptrona. Rekurzivne neuronske mreže su prikladne za analizu usmjerenih acikličkih grafova koji imaju korijen grafa. Dakle, rekurzivne neuronske mreže slijede sljedeću rekurziju:

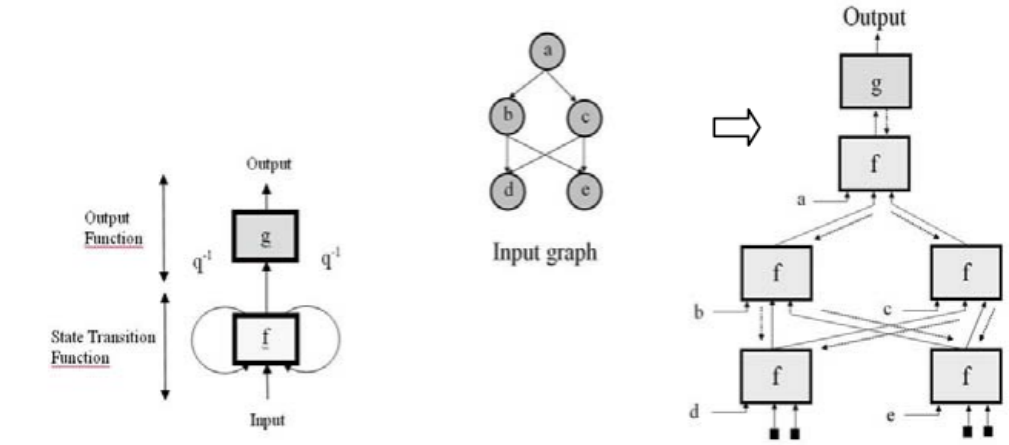
$$a(v) = f(a(ch[v]), I(v), v, W_f)$$
$$y(v) = g(a(v), v, W_g),$$

gdje W_f i W_g predstavljaju težine mreža f i g, a $I(v)$ predstavlja relaciju incidencije vrha v. Prije svega, potreban je ulazni graf pomoću kojega se "razmota" mreža stanja prijelaznih funkcija, što bi značilo da se kreira kopija ulaznog grafa. U ovoj fazi, kao rezultat dobiva se prijelazna neuronska mreža koja ima strukturu ulaznog grafa. Ovaj proces razmotavanja odvija

se u fazi učenja i fazi predviđanja. Vrhovi ulaznog grafa, u prijelaznoj mreži, su zamjenjeni kopijama stanja prijelazne funkcije, a korijen grafa zamjenjen je izlaznom funkcijom. Potom se može provesti prolazak unaprijed kroz dobivenu prijelaznu mrežu. Preciznije, u svakom vrhu v grafa U, stanje $a(v)$ dobiveno je pomoću prijelazne funkcije f čiji su argumenti stanja djece $ch[v]$ i relacija incidencije $I(v)$. Ako vrh v ima o djece, tada imamo da je

$$a(ch[v]) = (a(ch_1[v]), ..., a(ch_o[v])).$$

U tom slučaju, o je ujedno i najveći izlazni stupanj vrha v. Rekurzija započinje sa $a(null) = a_0$ što korespondira informaciji da vrh nema djecu. U korijenu grafa (označimo ga sa s), izlazni podatak dobiven je pomoću izlazne funkcije $y = g(a(s))$.



Slika 10: Proces razmotavanja [18]

Na lijevom dijelu slike 10 predstavljen je dijagram unaprijednog prolaska kroz prijelaznu mrežu, gdje se najprije uzastopno računaju stanja prijelazne funkcije f te se na kraju se izračuna konačan rezultat pomoću izlazne funkcije g. Također se može vidjeti postupak "razmotavanja". Ulazni graf ima 5 vrhova te ima određenu relaciju incidencije. Na temelju ulaznog grafa, kreira se novi graf koji imaju jednaki broj vrhova i podudarnu relaciju incidencije. Svaki vrh kreiranog grafa sadrži rezultate prijelaznih funkcija (stanja prijelazne funkcije) te dobiveni rezultat šalje svom roditelju, ovisno o relaciji incidencije. Tada vrh uzima u obzir relaciju incidencije i rezultate dobivene od strane vrhova djece kako bi se izračunalo novo stanje za taj vrh. Ovaj proces se ponavlja sve do korijena grafa, kada se prijelazna funkcija zamjenjuje

izlaznom funkcijom i dobiva se konačni rezultat. Dakle, podaci u trening skupu ne sadrže samo informacije (koje mogu biti uspoređivane i/ili mjerljive), nego i veze između tih informacija. Priroda takvih veza određena je u kontekstu u kojemu se primjenjuje rekurzivna neuronska mreža te se eksplicitno određuju veze između dijelova informacija. Na ovaj način pruža se veće znanje o podacima nego kad bi ih se promatralo bez integriranih veza. [18].
Rekurzivne neuronske mreže se najčešće koriste u analizi prirodnog jezika, prepoznavanju govora ili analizi DNA molekula. Mogu se koristiti za bilo koji problem koji uključuje strukturalne podatke, u smislu da osim informacija koje podatak nosi, važnost imaju i veze između informacija u podatku.

5.1 Rekurentne neuronske mreže (eng. Recurrent neural network)

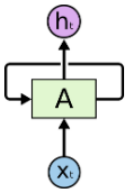
Uvedimo najprije nekoliko definicija iz teorije grafova.

Definicija 5.4 *Neka je G graf. Šetnja u G je konačan niz bridova u kojem su svaka dva uzastopna brida ili susjedni ili jednaki.*

Definicija 5.5 *Staza je šetnja u kojem su svi bridovi različiti. Put je staza u kojoj su svi vrhovi različiti osim eventualno početnog i krajnjeg vrha. Za stazu ili put kažemo da su zatvoreni ako je početni vrh jednak krajnjem.*

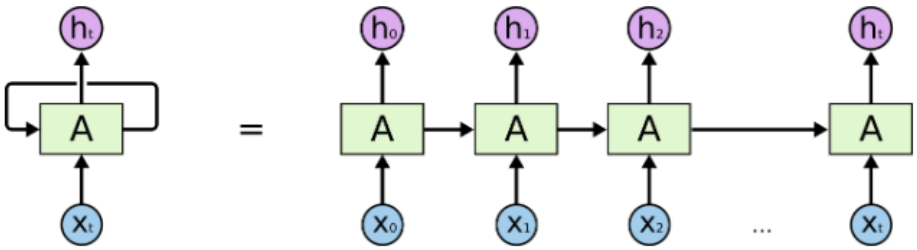
Definicija 5.6 *Ciklus je zatvorena staza.*

Rekurentne neuronske mreže (eng. Recurrent neural network) su rekurzivne neuronske mreže koje u svojoj strukturi imaju petlje, što bi značilo da graf po kojem se rekurentna neuronska mreža "razmotala" ima cikluse.
Motivacija za nastanak takvih neuronskih mreža je potreba za pamćenjem prethodnih informacija. Promotrimo dijagram rekurentne neuronske mreže.



Slika 11: Dijagram rekurentne neuronske mreže [19]

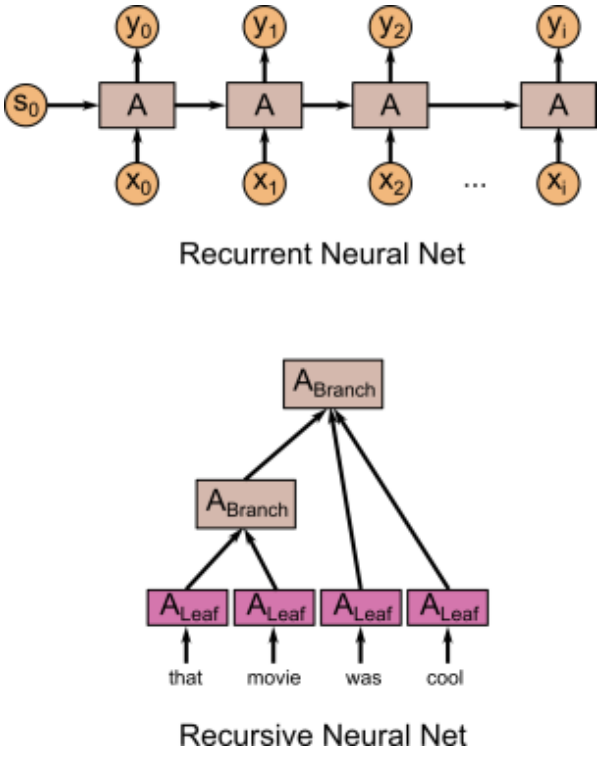
U gornjem dijagramu, dio neuronske mreže A prima ulazne podatak x_t i ispisuje izlazni podatak h_t. Petlja omogućuje da se izlazna vrijednost dijela A ponovno vraća u dio A kao ulazna vrijednost. Rekurentna neuronska mreža se može shvatiti kao više kopija jedne neuronske mreže i time se izlazni podatak jedne prenosi na sljedeću kopiju, sve do posljednje. Prikaz takve usporedbe se može vidjeti na sljedećoj slici:



Slika 12: Razmotavanje rekurentne neuronske mreže [19]

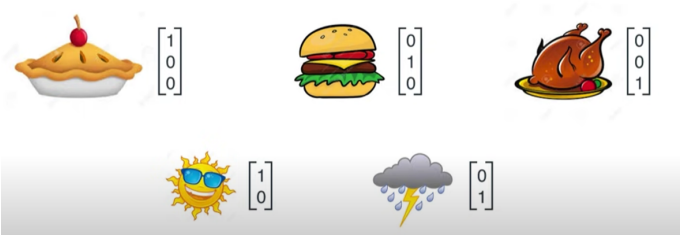
Jedna od prednosti rekurentnih neuronskih mreža je ideja da se mogu povezati prethodne informacije sa sadašnjim zadatkom. Primjerice, želimo li predvidjeti posljednju riječ u rečenici "Oblaci su na ...", velika je vjerojatnost da će posljednja riječ biti "nebo". U takvim slučajevima, gdje je jaz između relevantnih informacija i mjesta gdje su one potrebne mali, RNN mogu naučiti koristiti informacije iz prošlosti i predvidjeti buduću riječ. Međutim, postoje slučajevi gdje je potreban veći kontekst. Na primjer, želimo predvidjeti posljednju riječ u rečenici "Odrastao sam u Njemačkoj i tečno govorim ...". Nedavne informacije iz rečenice sugeriraju da riječ koja nedostaje je vjerojatno naziv jezika. Ako želimo dobro pogoditi o kojem se jeziku radi, potreban nam je kontekst iz prošlosti. Sasvim je moguće da jaz između relevantne informacije i mjesta gdje je ona potrebna postane vrlo velik. Nažalost, kako taj jaz raste, rekurentne neuronske mreže postaju nesposobnije naučiti povezivati nove informacije. Ovaj problem se zove problem dugoročne ovisnosti (eng. Long-Term Dependencies) [19], a rješavaju ga Mreže dugog kratkoročnog pamćenja (eng. Long short-term memory, LSTM). LSTM je posebna vrsta rekurentnih neuronskih mreža koja je u mogućnosti učiti dugoročne ovisnosti. LSTM su dizajnirane isključivo kako bi izbjegli problem dugotrajne ovisnosti. Promotrimo sada primjer u kojem ćemo koristiti rekurentne neuronske mreže. Rekurzivne neuronske mreže su generalizacija rekurentnih neuronskih mreža. Rekurentna neuronska mreža se "razmotava" kroz vrijeme. Rekurentne neuronske mreže se koriste u slučaju sekvencijalnih ulaznih podataka gdje ključnu ulogu ima vrijeme. Ulazni podatak rekurentne neuronske mreže ovisi o prethodno dobivenom izlaznom podatku. Primjerice,

rekurentne neuronske mreže su iznimno primjenjive u predviđanju sljedeće riječi u rečenici. Kako bi uspješno predvidjela sljedeću riječ, rekurentna neuronska mreža mora uzeti u obzir prethodnu riječ (u nekim slučajevima i nekoliko prethodnih riječi). Rekurentne neuronske mreže "posjeduju memoriju" koja ima ključnu ulogu prilikom predviđanja i koja se puni svaki put s novim izačunom.
S druge strane, rekurzivna neuronska mreža se ne mora razmotavati kroz vrijeme. Takve su mreže iznimno uspješne prilikom obrade prirodnog jezika koristeći stabla koja reprezentiraju stukruru podatka, ali ne i hijerarhiju podatka u usporedbi sa rekurentnim neuronskim mrežama te dubokim neuronskim mrežama. Rekurzivne neuronske mreže se najčešće bave obradom prirodnog jezika u slučaju klasifikacije semantičkog značenja rečenice gdje vrijeme nije ključan faktor. [20]



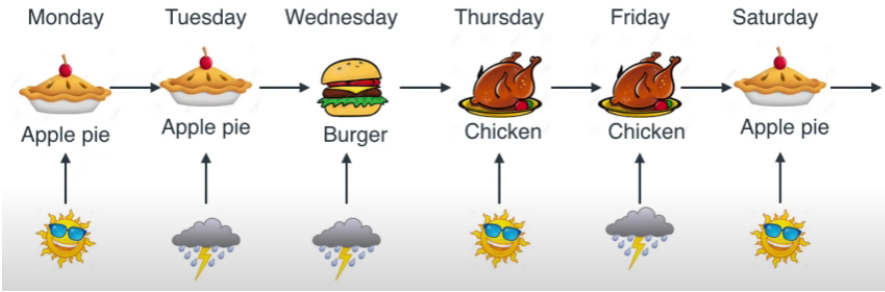
Slika 13: Usporedba arhitekture rekurzivnih i rekurentnih neuronskih mreža [20]

Primjer 5.1 *Kuhar svaki dan kuha ručak ovisno o vremenu (sunčano ili kišno) i ovisno o tome što je kuhao prethodni dan. On zna pripremati pitu, hamburger i piletinu. Prikazat ćemo te varijable pomoću vektora.*



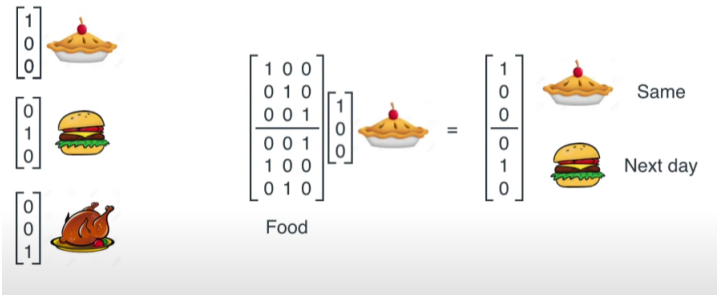
Slika 14: Varijable prikazane pomoću vektora [21]

Kuhar priprema ručak u sljedećem nizu: pita, hamburger, piletina. Ako je danas pripremao pitu, sutra će pripremati hamburger, a preksutra piletinu, itd. Osim što kuhar priprema ručak po pravilu pita-hamburger-piletina, on će odlučiti što će kuhati ovisno i o vremenu. Točnije, ako je vrijeme sljedeći dan sunčano (saznaje iz prognoze), kuhar će kuhati isto jelo koje je prethodno napravio, a ako je sljedeći dan kišno vrijeme, kuhar će kuhati sljedeće jelo u nizu. Primjer rasporeda pripremanja jela je prikazan na sljedećoj slici.



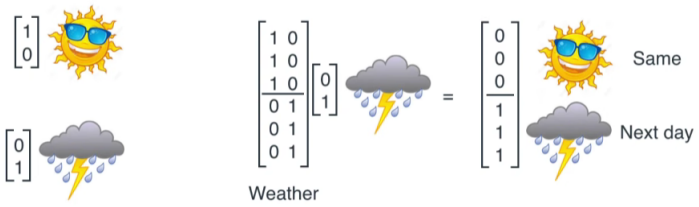
Slika 15: Raspored kuhanja [21]

Dakle, ako je kuhar u ponedjeljak pripremao pitu, a u utorak će prema prognozi biti sunčani dan, ponovit će to jelo i u utorak. Ako je za srijedu prognozirano kišno vrijeme, kuhar će u srijedu kuhati sljedeće jelo u nizu, a to je hamburger. Promotrimo sada matricu koja određuje jelo koje će kuhar sljedeći dan pripremati. Pomnožimo li tu matricu sa vektorom odgovarajućeg jela, dobivamo novi vektor gdje prve tri komponentne govore koja hrana se jela danas, a druge tri komponentne govore koja hrana će se jesti sutra. Dakle, danas smo jeli pitu, a sljedeće u nizu je hamburger.



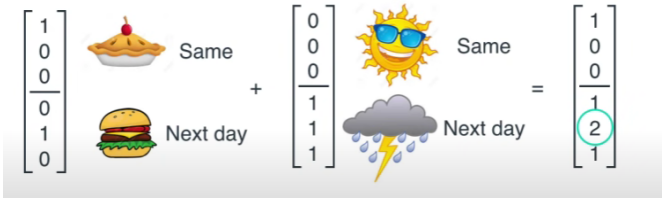
Slika 16: Predviđanje sutrašnjeg jela [21]

Analogno, matrica koja nam daje informaciju o vremenu govori kakvo je vrijeme danas i kakvo je vrijeme sutra i time kuhar odlučuje što će pripremati za ručak sljedeći dan.



Slika 17: Vremenska prognoza [21]

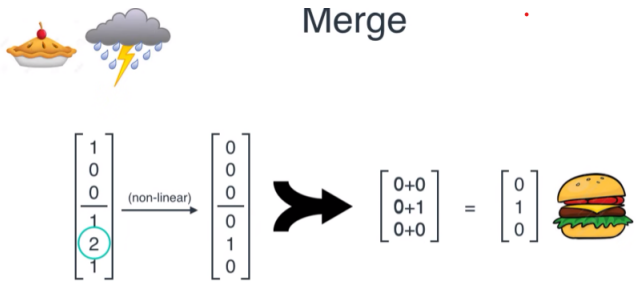
Kako je potrebno donijeti odluku na temelju dvije varijable, vremenu i jelu koje se pripremalo, potrebno je ove dvije matrice povezati, odnosno uzeti vektor koji nam daje informaciju o sljedećem jelu u nizu i vektor koji nam daje informaciju o vremenskoj prognozi i zbrojit ih.



Slika 18: Zbrajanje vektora hrane i vektora vremena [21]

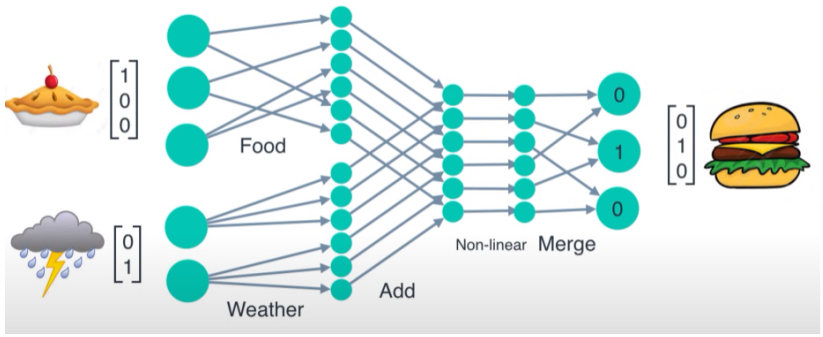
Uzmimo da je kuhar danas pripremao pitu te da je sutra kišni dan. Prvi vektor nam govori da je kuhar pripremao pitu i sljedeće u nizu je hamburger, a drugi vektor da je danas sunčano vrijeme, ali da je sljedeći dan kišno

vrijeme. Nakon što smo dobili novi vektor, na njega primjenimo nelinearnu funkciju i dobivamo novi vektor koji nam daje odgovor da će sljedeći dan kuhar pripremati hamburger.



Slika 19: Izlazni podatak [21]

Rekurentnu neuronsku mrežu može se prikazati i kao graf.



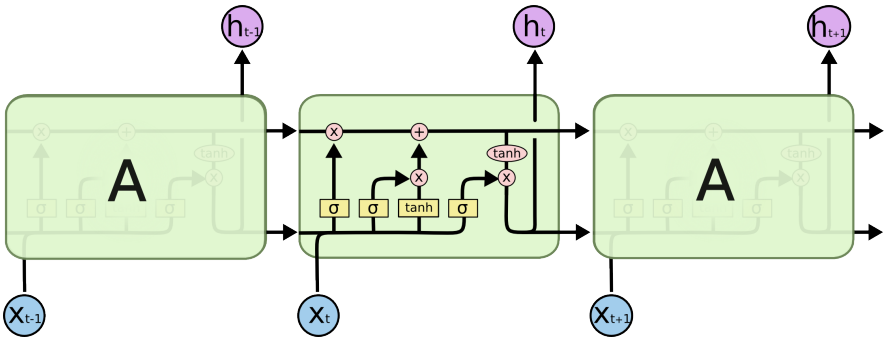
Slika 20: Rekurentna neuronska mreža [21]

Važno je napomenuti da izlazni podatak o jelu je zapravo ponovno ulazni podatak te iste rekurentne neuronske mreže i time ta mreža pamti jelo koje je prethodilo te na temelju toga određuje koje jelo je sljedeće po rasporedu uz nove ulazne podatke o vremenu.

Ovaj primjer opisuje problem kratkoročnog pamćenja. Međutim, ukoliko se radi o problemu dugoročnog pamćenja, tada je potrebno konstruirati puno složenije rekurentne neuronske mreže.

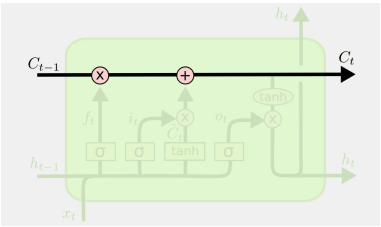
5.2 Mreže dugog kratkoročnog pamćenja (eng. Long short-term memory)

Mreže dugog kratkoročnog pamćenja (eng. Long short-term memory, LSTM) rješavaju problem dugog kratkoročnog pamćenja. Problem se može opisati kao problem povezivanja prijašnjih informacija s trenutnom. Taj problem se često javlja prilikom obrade prirodnog jezika gdje je cilj predvidjeti sljedeću riječ u rečenici ili prilikom analize videozapisa gdje je cilj predvidjeti sljedeći kadar. Dok je za mnoge neuronske mreže taj problem nesavladiv, mreže dugog kratkoročnog pamćenja su dizajnirane kako bi ga rješile. [19] Mreže dugog kratkoročnog pamćenja uvode Sepp Hochreiter i Jurgen Schmidhuber 1997.godine. [19] Navedene mreže mogu vrlo jednostavno rješavati probleme dugoročnog pamćenja. LSTM je posebna vrsta rekurentnih neuronskih mreža s dodatnim svojstvima. Njezina struktura je složenija od strukture standardnih rekurentnih neuronskih mreža. U svakom (ponavljajućem) sloju postoje četiri dodatna sloja koja na određeni način utječu na izlazne podatke.



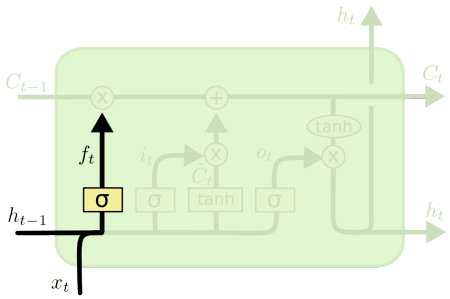
Slika 21: Struktura LSTM-a [19]

Stanje stanice (eng. cell state) je horizontalna linija na vrhu dijagrama. Stanje stanice se može shvatiti kao pokretna traka koja se kreće kroz sloj mreže. Najčešće, informacija u stanju stanice ostaje nepromijenjena.



Slika 22: Stanje stanice [19]

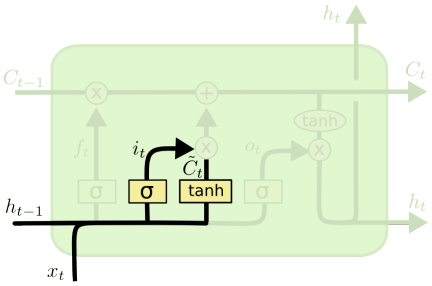
Međutim, LSTM imaju mogućnost utjecaja na promjenu informacije u stanju stanice pomoću prolaza (eng. gate). To su dijelovi sloja mreže koji mogu promjeniti ulaznu informaciju ili na temelju ulazne informacije donositi zaključke i kreirati izlaznu informaciju. Za početak, potrebno je odlučiti koliki dio ulazne informacije želimo zadržati, a koliki odbaciti. Takvu odluku donosi "forget gate layer" realiziran putem sigmoidne funkcije. Vrijednost sigmoidne funkcije upućuje koliki dio informacije će se zadržati. Vrijedi da 0 rezultira potpunim odbacivanjem informacije, a 1 rezultira potpunim zadržavanjem informacije. Vrijednosti h_{t-1} i x_t uključene su u izračun argumenta sigmoidne funkcije zajedno s odgovarajućom matricom težina W_f i dodatnim vektorom b_f. Vrijednost sigmoidne funkcije nalazi se unutar intervala $\langle 0, 1\rangle$ i utječe na informaciju koja se prosljeđuje u stanju stanice C_{t-1}, a odnosi se na to koliki će se dio informacije zadržati u daljnjem procesu, a $t = 1, ..., m$, gdje je m broj prijašnjih podataka koji se uzimaju u obzir za jedno predviđanje.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Slika 23: Forget gate layer [19]

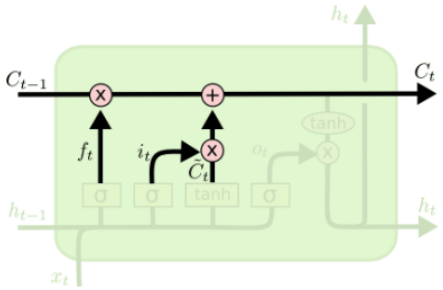
Nadalje, potrebno je odrediti što će se pohraniti kao nova informacija u stanje stanice. Taj postupak se sastoji od dva dijela. Prvi dio je sigmoidni sloj ulaznih vrata (eng. input gate layer) i odlučuje koje vrijednosti ćemo pohraniti. Na sličan način, kao i sloj vrata zaboravljanja (eng. forget gate layer), koristi sigmoidnu funkciju kojoj je argument $W_i[h_{t-1}, x_t] + b_i$, gdje je W_i odgovarajuća matrica težina i b_i odgovarajući dodatni vektor u trenutnom sloju mreže. Drugi dio je *tanh* sloj koji kreira vektor potencijalnih vrijednosti $\tilde{C}_t$ koje bi se mogle pohraniti u stanje.
Funkcija $tanh : \mathbb{R} \to \langle -1, 1\rangle$ definirana je kao $tanh(x) = \frac{sinh(x)}{cosh(x)}$.
Argument *tanh* funkcije je $W_C[h_{t-1}, x_t] + b_C$, gdje je W_C odgovarajuća matrica težina i b_C odgovarajući dodatni vektor u trenutnom sloju mreže. U sljedećem koraku, kombinacijom ovih dviju dobivenih vrijednosti, i_t i $\tilde{C}_t$, dobiva se nova informacija za pohranu.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Slika 24: Kreiranje nove informacije [19]

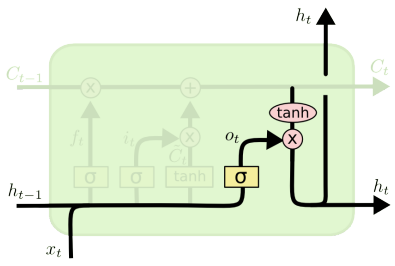
Sada je potrebno ažurirati staro stanje stanice C_{t-1} u novo stanje stanice C_t. Pomnožimo prijašnje stanje stanice s f_t i time zadržavamo ono što je potrebno. Potom, taj rezultat zbrojimo sa $i_t * \tilde{C}_t$, gdje je $*$ oznaka za množenje. Ta vrijednost je kandidat za novu vrijednost.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Slika 25: Kreiranje kandidata za novu vrijednost [19]

Konačni izlazni podatak je baziran na dosadašnjem stanju stanice. Najprije je potreban ponovni sigmoidni sloj σ_t koji određuje koji dio stanja stanice će utjecati na izlazni podatak. Nadalje, ažurirano stanje stanice C_t postaje argument funkcije *tanh* kako bi dobili vrijednosti između -1 i 1. Dobivene vrijednosti pomnožimo i dobivamo izlaznu vrijednost.



$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Slika 26: Kreiranje izlaznog podatka [19]

Velika prednost LSTM neuronskih mreža je njihova sposobnost pamćenja, odnosno zaboravljanja informacija u nizu podataka. Vrlo lako modeliraju probleme koji uključuju sekvencijalne nizove podataka te utjecaj prethodnih podataka na sljedeće podatke u nizu. Zbog toga su vrlo primjenjive u problemima prepoznavanja govora, prevođenja teksta i generiranje teksta. Otporne su u slučajevima kada nedostaju podaci ili postoje anomalije u podacima. Taj problem rješavaju u dijelu koji se zove sloj vrata zaboravljanja. Vrlo su prilagodljive te se mogu koristiti u regresijskim problemima i klasifikaciji. Međutim, usprkos mogućnosti modeliranja kompleksnih problema, LSTM neuronske mreže su računski skupe za treniranje i evaluiranje, pogotovo za velike baze podataka i sklone su pretjeranim prilagođavanjem bazi podataka. Zato uvijek zahtjevaju veću bazu podataka kako bi mogle naučiti uzorak u podacima i izbjeći pretjerano prilagođavanje manjim bazama podataka.

6 Primjer programske realizacije rekurentne neuronske mreže

U nastavku je naveden i detaljno objašnjen primjer programske realizacije rekurentne neuronske mreže s ciljem predviđanja završne cijene dionice tvrtke Apple s obzirom na nekoliko prethodnih dana. Uzet će se obzir završne cijene dionica u nekoliko proteklih dana te na temelju tih informacija predvidjet završna cijena sljedećeg dana. Programska realizacija provest će se u Python programu. Python je računalni programski jezik koji služi za kreiranje programa i aplikacija. Nasljednik je programskog jezika ABC. Najčešće se koristi u kreiranju web-stranica, aplikacija, software-a itd. Zbog svoje jednostavnosti i široke primjene, postao je među najkorištenijim programskim jezicima.
Python je osmislio Guido van Rossum 1980-ih godina u Centrum Wiskunde & Informatica (CWI) u Nizozemskoj. Njegova implementacija je započela 1989.godine. Python 2.0 objavljen je 16. listopada 2000. godine, a Python 3.0 3. prosinca 2008. godine kao unaprijeđenje prijašnjih verzija. Usavršavanja programa nastavljaju se danas. Od listopada 2023. godine Python 3.12 je stabilno izdanje koje se najčešće koristi. Unutar Pythona postoji veliki broj biblioteka koje sadrže brojne alate prilagođene raznim zadacima. Upravo je to svojstvo jedna od najvećih prednosti Python programa [22].

6.1 Definicija dionica

Dionica je vrijednosni papir koji omogućuje njenom vlasniku (dioničaru) pravo na dio dobiti koju odgovarajuća tvrtka može i želi distribuirati. Dobit dionice se dioničaru isplaćuje kroz dividende, odnosno društvene raspodjele tekućih ili akumuliranih prihoda dioničarima, zasnovane na broju i vrsti dionica koje su u vlasništvu dioničara [23]. Dodatna prednost posjedovanja dionica je i mogućnost zarade prilikom njihove prodaje po cijeni većoj od one po kojoj su kupljene. Cijena dionice varira s obzirom na nekoliko čimbenika gdje su neki od važnijih ponuda i potražnja dobra na tržištu, gospodarstvo, rat, industrija, politika i okoliš. Za dioničara je važno pratiti sve te čimbenike te biti u tijeku događanja koje se tiču njegove djelatnosti. Kako su izložene stalnim promjenama, ulaganje u dionice ne garantira pozitivni povrat uloga ni pozitivnu isplatu dividendi.

6.2 Baza podataka

Baza podataka na kojoj će se graditi model sadrži cijene dionica u periodu od 03. siječnja 2011. godine do 13. kolovoza 2017. godine [24]. Baza podataka je preuzeta sa web-stranice Kaggle na kojoj se nalaze mnogobrojne baze podataka zajedno sa implementiranim modelima strojnog učenja za analizu podataka. Za učitavanje baze podataka koristimo biblioteku *pandas* [25]. To je brz i učinkovit alat za manipulaciju podacima spremljenih u tablice. Važno svojstvo navedene biblioteke je da su podaci koje obrađuje indeksirani, odnosno označeni odgovarajućim brojem. Na taj način omogućen je brzi i direktni pristup podacima unutar strukture i zbog toga je to jedan od najčešće korištenih alata za analizu podataka. Nakon učitavanja navedene biblioteke, učitavaju se i podaci koji su spremljeni u csv. datoteku. CSV (eng. Comma Separated Values) je vrlo često korišten format za tablične baze podataka. CSV format se poistovjećuje sa Excel programom jer se najčešće CSV format upravo generira iz tog programa.[26]

```
import pandas as pd
baza_podataka=pd.read_csv("/content/apple_share_price.csv")
```

Tablica se sastoji od 1664 redaka i 6 stupaca. Retci predstavljaju pojedinačni dan u periodu u kojem su se pratile cijene. Prvi stupac označava datum. Stupac "Open" predstavlja početnu cijenu dionice na dan kada ju je dioničar odlučio prodati. Stupci "High" i "Low" predstavljaju najveću i najmanju cijenu dionice u tom danu redom. Stupac "Close" predstavlja završnu cijenu dionice tog dana nakon koje se cijena nije promjenila. Stupac "Volume" označava broj dionica koje su prodane ili kupljene u odgovarajućem danu.

	Date	Open	High	Low	Close	Volume
0	11-Aug-17	156.60	158.57	156.07	157.48	26257096
1	10-Aug-17	159.90	160.00	154.63	155.32	40804273
2	9-Aug-17	159.26	161.27	159.11	161.06	26131530
3	8-Aug-17	158.60	161.83	158.27	160.08	36205896
4	7-Aug-17	157.06	158.92	156.67	158.81	21870321
...	...	...	...	...	...	...
1659	7-Jan-11	47.71	48.05	47.41	48.02	77982212
1660	6-Jan-11	47.82	47.89	47.56	47.68	75106626
1661	5-Jan-11	47.08	47.76	47.07	47.71	63879193
1662	4-Jan-11	47.49	47.50	46.88	47.33	77337001
1663	3-Jan-11	46.52	47.18	46.41	47.08	111280407

1664 rows × 6 columns

Slika 27: Baza podataka

Sada će se modificirati baza podataka tako da podaci budu kronološki poredani odnosno da se najstariji podatak nalazi u prvom retku tablice, a najnoviji podatak u posljednjem retku tablice. Taj postupak nije nužno potreban, ali može olakšati razumijevanje i računanje. Za taj postupak najprije je potrebno obrnuti poredak indeksa u tablici pomoću naredbe *reindex*. U novonastaloj tablici se svakom retku ponovno dodjeljuju indeksi pomoću naredbe *reset_ index*. Iz baze podataka izbacujemo stupac *index* koji ne pridonose korisnim informacijama pomoću naredbe *drop*.

```
baza_podataka=baza_podataka.reindex(index=baza_podataka.index
                                    [::-1])
baza_podataka=baza_podataka.reset_index()
baza_podataka=baza_podataka.drop(['index'], axis=1)
```

Sada baza podataka izgleda ovako:

	Date	Open	High	Low	Close	Volume
0	3-Jan-11	46.52	47.18	46.41	47.08	111280407
1	4-Jan-11	47.49	47.50	46.88	47.33	77337001
2	5-Jan-11	47.08	47.76	47.07	47.71	63879193
3	6-Jan-11	47.82	47.89	47.56	47.68	75106626
4	7-Jan-11	47.71	48.05	47.41	48.02	77982212
...	...	...	...	...	...	...
1659	7-Aug-17	157.06	158.92	156.67	158.81	21870321
1660	8-Aug-17	158.60	161.83	158.27	160.08	36205896
1661	9-Aug-17	159.26	161.27	159.11	161.06	26131530
1662	10-Aug-17	159.90	160.00	154.63	155.32	40804273
1663	11-Aug-17	156.60	158.57	156.07	157.48	26257096

1664 rows × 6 columns

Slika 28: Baza podataka

6.3 Analiza podataka

Promotrimo vrijednosti odgovarajućih značajki koje se pojavljuju u bazi podataka. Naredbom *dtypes* može se dobiti brzi uvid u tip podataka u svakom pojedinačnom stupcu.

```
baza_podataka.dtypes
```

```
Date       object
Open      float64
High      float64
Low       float64
Close     float64
Volume      int64
dtype: object
```

Slika 29: Analiza podataka

U ovom slučaju vidimo da se u stupcu "Date" nalaze objekti, odnosno datumi. U stupcima "Open","High","Low" i "Close" nalaze se cijene dionica koji su realni brojevi te posljednji stupac "Volume" sadrži prirodne brojeve. Nadalje, naredbom *info()* se može dobiti detaljniji uvid o podacima kao što je broj podataka u svakom stupcu, broj vrijednosti koje nedostaju po stup-

cima te tip podataka.

```
baza_podataka.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1664 entries, 0 to 1663
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    1664 non-null   object
 1   Open    1664 non-null   float64
 2   High    1664 non-null   float64
 3   Low     1664 non-null   float64
 4   Close   1664 non-null   float64
 5   Volume  1664 non-null   int64
dtypes: float64(4), int64(1), object(1)
memory usage: 78.1+ KB
```

Slika 30: Analiza podataka

U svakom stupcu baze nalazi se 1664 vrijednosti te nijedna ne nedostaje. Broj vrijednosti koje nedostaju po stupcima se može dobiti i pomoću naredbe *isnull().sum()* koja isključivo daje tu informaciju.

```
baza_podataka.isnull().sum()
```

```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

Slika 31: Analiza podataka

Ponekad se može dogoditi da u bazi podataka nedostaju neke vrijednosti te se na tim pozicijama nalazi riječ *NaN* pa je potrebno postići dogovor kako će se te vrijednosti tretirati. Može se dogovoriti da se samo uklone retci u kojima vrijednosti nedostaju, a ponekad se te pozicije popune dogovorenim

vrijednostima, primjerice brojem 0 ili srednjom vrijednosti značajke. U tim slučajevima će se prikazati broj vrijednosti koje nedostaju za odgovarajuću značajku.
Naredbom *describe()* provodi se statistička obrada podataka po stupcima.

```
baza_podataka.describe()
```

	Open	High	Low	Close	Volume
count	1664.000000	1664.00000	1664.000000	1664.000000	1.664000e+03
mean	91.604105	92.39149	90.760619	91.594525	7.958391e+07
std	28.258360	28.42185	28.094248	28.272351	5.446191e+07
min	45.240000	45.39000	44.360000	45.050000	1.147592e+07
25%	66.912500	67.59750	66.385000	66.835000	3.883495e+07
50%	92.150000	92.55500	90.950000	92.010000	6.576690e+07
75%	112.542500	113.75500	111.550000	112.650000	1.042734e+08
max	159.900000	161.83000	159.110000	161.060000	4.702467e+08

Slika 32: Statistička obrada podataka

Naredba *describe* uzima u obzir samo stupce čije su vrijednosti brojevi i u ovom slučaju daje statističku obradu svih stupaca osim stupca "Date". Primjerice, promotrimo stupac "Close". Prvi redak "count" označava broj vrijednosti koje se nalaze u tom stupcu. Redak "mean" označava aritmetičku sredinu stupca, redak "std" označava standardnu devijaciju stupca, retci "min" i "max" označavaju minimalnu i maksimalnu vrijednost stupca, respektivno te retci "25%", "50%" i "75%" označavaju donji kvartil, medijan i gornji kvartil, redom. Donji kvartil je broj koji označava da je 25% podataka manje ili jednako toj vrijednosti, medijan je broj koji označava da je 50% podataka manje ili jednako toj vrijednosti te gornji kvartil je broj koji označava da je 75% podataka manje ili jednako toj vrijednosti. Iz svega dobivenog, lako je uočiti da je npr. srednja vrijednost značajke "Open" 91.604105, dok je standardna devijacija te značajke 28.25836 uz raspon vrijednosti od 45.24 do 159.0.

Učitat ćemo novu biblioteku *numpy* koja služi za računanje s objektima koji predstavljaju višedimenzionalna polja. Osim toga, navedena biblioteka koristi se za statističke obrade podataka, linearnu algebru, slučajne simulacije

podataka, sortiranje, selektiranje, itd. *Numpy* biblioteka je najčešće korištena biblioteka kada je u pitanju matematičko računanje [27].
Uvest ćemo novu varijablu *close_price* koja predstavlja završne cijene dionica. Završne cijene dionica su konačne cijene po kojoj su te dionice prodane, odnosno kupljene. Da bi se ta varijabla mogla koristiti u daljnjem izračunu potrebno ju je preoblikovati tako da ima 1664 redaka i 1 stupac.

```
import numpy as np
close_price=np.array(baza_podataka["Close"]).reshape(-1,1)
```

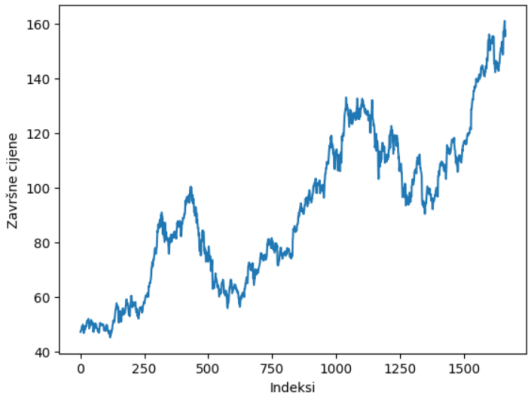
Varijabla ima sljedeću vrijednost:

```
array([[ 47.08],
       [ 47.33],
       [ 47.71],
       ...,
       [161.06],
       [155.32],
       [157.48]])
```

Slika 33: Završne cijene

Pomoću biblioteke *matplotlib.pyplot* podaci se mogu vizualno predočiti. Biblioteka *matplotlib.pyplot* [28] je skup funkcija koje se koriste za crtanje grafova i nalazi se unutar biblioteke *matplotlib*. Biblioteka *matplotlib* [29] koristi se za kreiranje statičkih i dinamičkih grafičkih vizualizacija u Python-u.

```
import matplotlib.pyplot as plt
plt.plot(close_price)
plt.xlabel("Indeksi")
plt.ylabel("Zavrsne cijene")
```



Slika 34: Završne cijene

Na slici 34 se nalazi grafički prikaz završnih cijena dionica s obzirom na vremenski tijek. Nezavisna varijabla su indeksi, koji predstavljaju vrijeme, a zavisna varijabla su završne cijene dionica. Može se primjetiti da cijene dionica imaju rast i pad kroz vrijeme te da se relacija između vremena i cijena ne može precizno odrediti, odnosno formalno zapisati. Može se zaključiti da generalno cijene rastu kroz vrijeme, ali ako se cijene promatraju na kraćem vremenskom periodu, njihove vrijednosti i padaju i rastu pa je zato najjednostavnije cijene prikazati linijskim grafom kako bi se rast i pad cijena mogao najbolje pratiti.
Jedan od standardnih postupaka koji se provodi u obradi podataka jest skaliranje podataka kako bi izračuni bili precizniji i pregledniji. Skaliranje podataka je postupak transformiranja vrijednosti značajki skupa podataka kako bi se nalazili unutar određenog raspona. [30] Cilj je osigurati da nijedna značajka ne bude dominantna u usporedbi s ostalima što doprinosi boljim radom algoritma. Skaliranje je vrlo važno u strojnom učenju jer većina algoritama koristi euklidsku udaljenost između dviju podatkovnih točaka u svojim izračunima. Cilj skaliranja je da se pojedinačni utjecaji specifičnih podataka umanjuju. Postoji nekoliko vrsta skaliranja, među kojima razlikujemo postupak naziva:

- standardizacija: prosjek svih vrijednosti značajki postaje 0, a standardna devijacija postaje 1,
- normalizacija: sve vrijednosti se postavljaju unutar intervala između 0 i 1,
- Min-Max skaliranje: najmanja vrijednost postaje 0, a najveća vrijednost postaje 1.

U ovom radu će se koristiti Min-Max skaliranje. Funkcija skaliranja definirana je kao:

$$X_{novo} = \frac{X_i - \min(S)}{\max(S) - \min(S)},$$

gdje je S skup svih podataka, X_i je vrijednost starog podatka, a X_{novo} vrijednost novog podatka za $i = 1, ..., 1664$.
Kako bi mogli skalirati podatke u Pythonu potrebno je uvesti iz biblioteke *sklearn.preprocessing* odgovarajući način skaliranja *MinMaxScaler* u kombinaciji sa naredbom *fit_transform*. Naredba uzima polje podataka, što su u ovom slučaju završne cijene dionica, te ih skalira navedenim postupkom. Rezultat te naredbe je ponovno polje završnih cijena, ali sa skaliranim vrijednostima.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
close_price=scaler.fit_transform(close_price)
```

Sada završne cijene imaju sljedeće vrijednosti:

```
array([[0.01749849],
       [0.01965348],
       [0.02292906],
       ...,
       [1.        ],
       [0.95052151],
       [0.96914059]])
```

Slika 35: Skalirane završne cijene

Dobili smo novo polje u kojemu se nalaze skalirane vrijednosti završnih cijena unutar intervala između 0 i 1. Na taj način su se umanjile velike udaljenosti između vrijednosti podataka, odnosno između završnih cijena dionica.

Prema osnovnim načelima strojnog učenja u kontekstu organizacije podataka s ciljem učenja potrebno je podijeliti podatke u dva dijela: podatke za treniranje modela i podatke za testiranje modela. Možemo odrediti da će 80% podataka služiti za treniranje modela, a preostalih 20% za testiranje.

```
train_size=int(len(close_price)*0.80)
test_size=len(close_price)-train_size
```

U ovom slučaju 1331 podatak služi za treniranje modela, a preostalih 333 služi za testiranje modela. U varijablu *train_data* spremamo prvih 1331 podataka za treniranje, a u varijablu *test_data* spremamo preostalih 333 podatka za testiranje modela.

```
train_data=close_price[0:train_size,:1]
test_data=close_price[train_size:len(close_price),:1]
```

Varijable *train_data* i *test_data* služit će u daljnjem kreiranju podataka za treniranje i podataka za testiranje. U ovom primjeru ćemo koristiti model rekurentne neuronske mreže, a kojoj je jedno od osnovnih svojstava mogućnost analiziranja i obrade vremenske dimenzije. S tim ciljem prilagođavamo oblik podataka. Potrebno je stvoriti niz podataka duljine 50, koji predstavljaju niz završnih cijena unutar 50 dana. Cilj modela je predvidjeti završnu cijenu za 51. dan s obzirom na prethodnih 50 dana.

Funkcija *create_dataset* kreira podatke koji će služiti za treniranje i testiranje modela. Argumenti funkcije su *dataset*, koji predstavlja podatke koji će

se slagati u nizove, i *time_step*, koji predstavlja duljinu budućeg niza. Za početak kreiraju se dvije prazne liste *dataX* i *dataY*. U listu *dataX* spremat će se nizovi duljine 50, a u listu *dataY* nizovi duljine 1. Upotrebom *for* petlje koristimo sve podatke koje imamo na raspolaganju iz uvedene baze podataka. Pomoćna varijabla a služi kako bi se spremio niz duljine 50. Za $i = 0$ u varijablu a sprema se prvih 50 završnih cijena te u pomoćnu varijablu b završna cijena za 51. dan. Kreirane liste, duljine 50 i duljine 1, spremaju se u početne liste *dataX* i *dataY*. U sljedećem koraku, za $i = 1$, u varijablu a spremaju se završne cijene od 2. dana do 51. dana, a u varijablu b završna cijena za 52. dan te bi se te liste ponovno ubacivale u liste *dataX* i *dataY*. Taj postupak se ponavlja sve dok se u liste ne bi spremilo posljednjih 50 završnih cijena i zadnja završna cijena.

```
def create_dataset(dataset, time_step):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]
        dataX.append(a)
        b=dataset[i + time_step, 0]
        dataY.append(b)
    return np.array(dataX), np.array(dataY)
```

Primjenimo sada tu funkciju na naše podatke za treniranje i podatke za testiranje za $time\_step = 50$.

```
time_step = 50
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)
```

Promotrimo sada kojeg su oblika dobiveni podaci.

```
print(X_train.shape),
print(y_train.shape),
print(X_test.shape),
print(y_test.shape)
```

```
(1280, 50)
(1280,)
(282, 50)
(282,)
```

Slika 36: Dimenzije dobivenih podataka

X_train se sastoji od 1280 nizova duljine 50, što znači da imamo 1280 nizova završnih cijena za 50 dana, a *y_train* sastoji se od 1280 vrijednosti koje

odgovaraju završnoj cijeni za 51. dan za svaki odgovarajući niz od 50 dana u *X_train*. Analogni zaključci vrijede za *X_test* i *y_test*.
Kreirani skupovi podataka *X_train* i *X_test* se moraju preoblikovati u odgovarajući predefinirani oblik za treniranje rekurentne neuronske mreže.

```
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],1)
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1],1)
print(X_train.shape)
print(X_test.shape)
```

```
(1280, 50, 1)
(282, 50, 1)
```

Slika 37: Nove dimenzije skupova podataka

Dobiveni skupovi podataka više nisu dvodimenzionalni, već trodimenzionalni. Primjerice, *X_train* ima oblik $(1280, 50, 1)$ što znači da ima 1280 nizova duljine 50 te svaki taj niz sadrži 50 nizova dimenzije 1. Analogno vrijedi za organizaciju podataka *X_test*. Na slici 38 prikazano je prvih 3 elementa niza *X_train* koji se sastoji od 1280 elemenata te sadrže 50 nizova duljine 1.

```
array([[[0.01749849],
        [0.01965348],
        [0.02292906],
        ...,
        [0.04508232],
        [0.04706491],
        [0.03706577]],

       [[0.01965348],
        [0.02292906],
        [0.02267046],
        ...,
        [0.04706491],
        [0.03706577],
        [0.01801569]],

       [[0.02292906],
        [0.02267046],
        [0.02560124],
        ...,
        [0.03706577],
        [0.01801569],
        [0.02379105]],

       ...,
```

Slika 38: Jedan element niza *X_train*

Jedan od načina kreiranja rekurentne neuronske mreže je putem biblioteke *Tensorflow*. *Tensorflow* je softverska biblioteka koja služi za kreiranje modela u strojnom učenju i umjetnoj inteligenciji. Navedena biblioteka može se koristiti za razne zadatke, ali se najčešće koristi za kreiranje neuronskih mreža. Primjenjiva je u velikom broju programskih jezika kao što su Python, JavaScript, Java i C++, a razvio ju je Google-ov tim za internu upotrebu [31]. Uz *TensorFlow* koristimo i *Keras*, biblioteku koja pruža Python sučelje za rješavanje problema strojnog učenja, a najčešće se odnosi na duboke neuronske mreže. Keras omogućuje rad neuronskih mreža, od obrade podataka, podešavanja parametara do implementacije modela [32].
Iz navedenih biblioteka koristimo klase za izgradnju neuronskih mreža. Klasa *Sequential* je dizajnirana za izgradnju neuronskih mreža na način da dodaje sloj po sloj u neuronsku mrežu. Na analogan način uvodimo klase i *Dense* i *LSTM*. Klasa *Dense* predstavlja potpuno povezani sloj neurona u neuronskoj mreži. Najčešće se koristi u završnoj fazi kreiranja neuronske mreže. Može se koristiti kao klasifikacijski ili regresijski sloj, ovisno o problemu [33]. Klasa *LSTM* predstavlja posebnu vrstu sloja u neuronskoj mreži. LSTM slojevi se razlikuju od Dense slojeva po tome što imaju dodatnu mogućnost pamćenja prethodnih informacija zato su vrlo primjenjivi u zadacima koji zahtjevaju vremensku dimenziju.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

U varijablu *model* spremamo model neuronske mreže. Taj model gradimo dodavanjem LSTM slojeva te običnog sloja. Tako kreirana neuronska mreža postaje rekurentna neuronska mreža. Argumenti u LSTM sloju su: *units*, *return_sequences* te *input_shape*. Argument *units* određuje broj neurona u sloju. Argument *return_sequences* ima vrijednost *True* ukoliko će se dodavati još slojeva nakon trenutnog. U suprotnom, ima vrijednost *False*. Argument *input_shape* precizira broj vremenskih koraka i broj značajki koje se uzimaju u obzir. U kôdu se u prvom LSTM sloju zadaje da je broj neurona u tom sloju 20 nakon kojega će se dodati još slojeva te se razmatra 50 prethodnih vrijednosti s jednom značajkom, odnosno razmatraju se završne cijene za prethodnih 50 dana. Analogno se definiraju preostali LSTM slojevi, no više nije potrebno unositi *input_shape*. Posljednji sloj definiran je kao *Dense* sloj sa jednim neuronom koji, kao svoj rezultat, daje jednu vrijednosti, odnosno predviđenu vrijednost završne cijene dionice za 51. dan.

Potrebno je kompajlirati neuronsku mrežu pomoću naredbe *compile* koja ima dva argumenta: *optimizer*, koji predstavlja optimizator neuronske mreže te *loss* koji predstavlja funkciju gubitka. Optimizator je algoritam koji

služi za podešavanje atributa neuronskih mreža poput težina i stope učenja. Ovdje se koristio *Adam* optimizator te za funkciju gubitka se koristila *mean_squared_error* s obzirom da je predviđena vrijednost varijable realan broj. Adam optimizator, skraćeno od "Adaptive Moment Estimation", je iterativni algoritam optimizacije koji se koristi za minimiziranje funkcije gubitka tijekom treniranja neuronskih mreža. Srednja kvadratna pogreška (Mean Squared Error, MSE) mjeri pogrešku u statističkim modelima. Procjenjuje prosječnu kvadratnu razliku između točnih i predviđenih vrijednosti. Kada model nema greške, MSE je jednak nuli. Kako se pogreška modela povećava, njegova vrijednost raste. Formula za računanje MSE-a je

$$MSE = \frac{\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2}{n},$$

gdje je y_i točna vrijednost, $\hat{y}_i$ predviđena vrijednost i n je broj podataka u trening setu. Naredbom *summary* dobivamo informacije o kreiranoj rekurentnoj neuronskoj mreži.

```
model=Sequential()
model.add(LSTM(20,return_sequences=True,input_shape=(50,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mse',optimizer='adam')
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 50, 20)            1760

 lstm_4 (LSTM)               (None, 50, 50)            14200

 lstm_5 (LSTM)               (None, 50)                20200

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 36211 (141.45 KB)
Trainable params: 36211 (141.45 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Slika 39: Rekurentna neuronska mreža

Dobivena rekurentna neuronska mreža ima tri LSTM sloja i jedan Dense sloj. U drugom stupcu dobivene tablice prikazuju se podaci o ulaznim i izlaznim dimenzijama pojedinačnih slojeva. Prvi sloj ima duljinu serije *None*, broj vremenskih koraka 50 (broj dana koji se uzima u obzir) i dimenziju izlaznog vektora 20 (broj neurona u tom sloju). Posljednji stupac prikazuje broj parametara koji se nalaze u odgovarajućem sloju. Naposljetku su prikazane informacije o ukupnom broju parametara unutar rekurentne neuronske mreže, odnosno koji od njih su prilagođeni, a koji ostali nepromjenjeni. U ovom slučaju su se svi parametri prilagodili.

Kreiranu rekurentnu neuronsku mrežu potrebno je trenirati na odgovarajućim podacima za što se koristi naredba *fit* koja ima nekoliko argumenata kao što su podaci za treniranje, broj epoha i duljina serije. Podaci za treniranje odnose se na prethodno kreirane skupove poadataka *X_train* i *y_train*. Broj epoha je broj ponavljanja treninga rekuretne neuronske mreže, a duljina serije se odnosi na količinu podataka koje će se uzeti u trening kroz svaku epohu.

```
model.fit(X_train, y_train, epochs = 40, batch_size = 30)
```

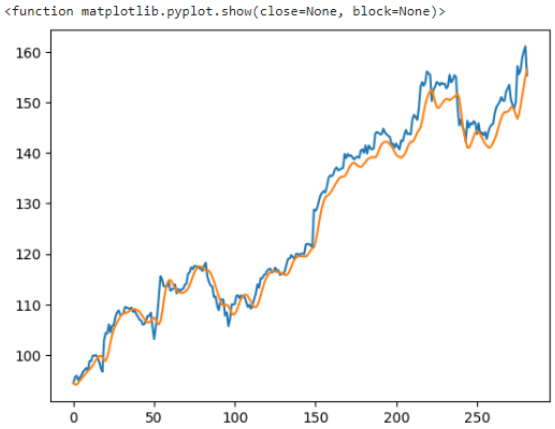
U ovom slučaju uzelo se da je broj epoha 40, a duljina serije 30, a te vrijednosti mogu podešavati u cilju poboljšanja algoritma.
U varijablu *y_pred* spremamo predviđanja koja je rekurentna neuronska mreža izračunala. U ovom koraku se koriste podaci za testiranje koji su spremljeni u *X_test*. Dobivene vrijednosti moraju se preoblikovati iz skaliranih u originalne podatke pomoću naredbe *scaler.inverse_transform*. Prije nego usporedimo predviđene podatke i točne podatke, potrebno je preoblikovati *y_test* te vrijednosti koje sadrži preoblikovati iz skaliranih u originalne.

```
y_pred=model.predict(X_test)
y_pred=scaler.inverse_transform(y_pred)
y_test=y_test.reshape(-1,1)
y_test=scaler.inverse_transform(y_test)
```

Grafički usporedimo dobivene rezultate: plava linija označava točne vrijednosti, a narančasta linija predviđene vrijednosti.

```
plt.plot(y_test)
plt.plot(y_pred)
```

U grafičkom prikazu se može vidjeti kako predviđanja nisu jednaka točnim vrijednostima, ali se može reći da model dobro prati stvarne vrijednosti.



Slika 40: Grafički prikaz točnih i predviđenih vrijednosti

Pozivanjem funkcije *mean_squared_error* iz biblioteke *sklearn.metrics* može se izračunati greška modela, odnosno korijen iz greške modela.

```
from sklearn.metrics import mean_squared_error
import math
print(math.sqrt(mean_squared_error(y_test,y_pred)))
```

Dobivena greška je 2.9254, pa se može zaključiti da model dobro predviđa završne cijene što se naslućivalo i iz grafa. Koristeći taj model, pokušat će se predvidjeti završna cijena dionice za datum 12.08.2017. koja ne postoji u bazi podataka, pa je potrebno uzeti u obzir prethodnih 50 dana.

```
close_price = pd.DataFrame(close_price)
```

	0
0	0.017498
1	0.019653
2	0.022929
3	0.022670
4	0.025601
...	...
1659	0.980605
1660	0.991552
1661	1.000000
1662	0.950522
1663	0.969141

1664 rows × 1 columns

Slika 41: Završne cijene prikazane u tablici

Uzima se u obzir posljednjih 50 završnih cijena koje spremamo u varijablu *x_ input* pomoću naredbe *tail(50)*. Potrebno je promjeniti oblik te varijable kako bi model mogao predvidjeti novu završnu cijenu.

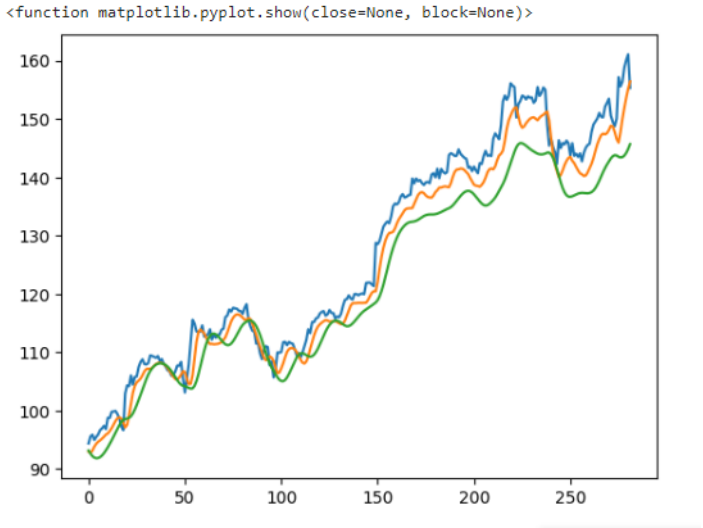
```
x_input=close_price.tail(50)
x_input_new=np.array(x_input).reshape(1,50,1)
```

Konačno, koristeći trenirani model na temelju posljednjih 50 dana, dobivamo završnu cijenu za 51.dan na datum 12.08.2017. Prije ispisa predviđene cijene potrebno je vrijednosti transformirati iz skalirane u originalnu.

```
new_price=model.predict(x_input_new)
new_price=scaler.inverse_transform(new_price)
```

Predviđanje je da će završna cijena za navedeni datum biti 154.22 dolara.

Kreiranoj rekurentnoj neuronskoj mreži je za treniranje algoritma bilo potrebno dvije minute, za što može zaključiti da je jako puno. Iz navedenih razloga modeli se treniraju na jakim računalima s brzim procesorima koji obrađuju jako velike baze podataka. Prilikom modeliranja realnih problema redovito je broj slojeva i broj neurona po sloju puno veći pa je onda i zahtjevnost algoritma puno veća i samim time bi vrijeme koje je potrebno za treniranje modela bilo puno duže.
U okviru ovog primjera kreirat ćemo još jednu rekurentnu neuronsku mrežu sa drukčijim parametrima te će se njihovi rezultati usporediti. Nova rekurentna neuronska mreža ima četiri LSTM sloja te u svakom sloju redom 10, 15, 10, 20 neurona. Posljednji Dense sloj ima jedan neuron kao i u prijašnjem slučaju zajedno s argumentima za *compile* funkciju. Za treniranje navedene rekurentne mreže broj epoha jednak je 20, a duljina serije 15, što je dvostruko manje nego što je bilo zadano za prethodnu rekurentnu neuronsku mrežu. Može se primjetiti da je također bilo potrebno dvije minute za treniranje algoritma, što navodi da bez obzira na promijenjene parametre, vrijeme potrebno za kreiranje modela će ostati približno jednako. Nakon što se izvršilo predviđanje podataka i računanje greške modela, greška modela je 6.49, što daje naslutiti da je greška ove rekurentne neuronske mreže veća od greške prethodnog modela koja je iznosila 2.954. Povećana greška može se uočiti i na grafičkom prikazu na slici 39, gdje zelena linija prikazuje predviđanja novog modela, narančasta linija prikazuje predviđanja prošlog modela te plava linija predstavlja točne vrijednosti. Za kraj se izračunala predviđena vrijednost završne cijene dionice na datum 12.08.2017. te iznosi 147.01 dolara. U usporedbi s prethodnim modelom, gdje je cijena bila 154.22 dolara, uočava se veća razlika i potencijalna greška modela.



Slika 42: Usporedba predviđenih vrijednosti dvije rekurentne neuronske mreže

Rekurentne neuronske mreže imaju jako važnu ulogu kada se u problem uvodi sekvencijalni niz podataka. Najčešći slučaj kada se koriste rekurentne neuronske mreže je kada problem ima vremensku dimenziju. Tada se podaci shvaćaju kao nizovi podataka u određenom vremenu. Rekurentne neuronske mreže imaju veliku prednost u tome što imaju mogućnost pamćenja prijašnjih podataka te su time primjenjive kada se u obzir uzimaju nizovi podataka. Drugi modeli strojnog učenja nemaju takvo svojstvo, odnosno nisu podobni za probleme koji uključuju vremenski tijek. Zato je u ovom primjeru, kada se prati kretanje završnih cijena kroz vrijeme, odnosno s obzirom na prethodnih 50 dana, rekurentna neuronska mreža bila pravi izbor. Drugi modeli su se također mogli koristiti, ali njihove greške bi bile mnogo veće te time ih nije potrebno uvoditi.

6.4 Programski kôd

```
import pandas as pd
baza_podataka=pd.read_csv("/content/apple_share_price.csv")
baza_podataka=baza_podataka.reindex(index=baza_podataka.index
                                    [::-1])
baza_podataka=baza_podataka.reset_index()
baza_podataka=baza_podataka.drop(['index'], axis=1)

baza_podataka.dtypes
baza_podataka.info()
baza_podataka.isnull().sum()
baza_podataka.describe()

import numpy as np
close_price=np.array(baza_podataka["Close"]).reshape(-1,1)

import matplotlib.pyplot as plt
plt.plot(close_price)
plt.xlabel("Indeksi")
plt.ylabel("Zavrsne cijene")

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
close_price=scaler.fit_transform(close_price)

train_size=int(len(close_price)*0.80)
test_size=len(close_price)-train_size
train_data=close_price[0:train_size,:1]
test_data=close_price[train_size:len(close_price),:1]

def create_dataset(dataset, time_step):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]
        dataX.append(a)
        b=dataset[i + time_step, 0]
        dataY.append(b)
    return np.array(dataX), np.array(dataY)


time_step = 50
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)
print(X_train.shape),
print(y_train.shape),
print(X_test.shape),
print(y_test.shape)
```

```
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],1)
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1],1)
print(X_train.shape)
print(X_test.shape)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM

model=Sequential()
model.add(LSTM(20,return_sequences=True,input_shape=(50,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mse',optimizer='adam')
model.summary()
model.fit(X_train, y_train, epochs = 40, batch_size = 30)

model2=Sequential()
model2.add(LSTM(10,return_sequences=True,input_shape=(50,1)))
model2.add(LSTM(15,return_sequences=True))
model2.add(LSTM(10,return_sequences=True))
model2.add(LSTM(20))
model2.add(Dense(1))
model2.compile(loss='mse',optimizer='adam')
model2.summary()
model2.fit(X_train,y_train, epochs=20, batch_size=15)

y_pred=model.predict(X_test)
y_pred2=model2.predict(X_test)
y_pred=scaler.inverse_transform(y_pred)
y_pred2=scaler.inverse_transform(y_pred2)
y_test=y_test.reshape(-1,1)
y_test=scaler.inverse_transform(y_test)


plt.plot(y_test)
plt.plot(y_pred)
plt.plot(y_pred2)

from sklearn.metrics import mean_squared_error
import math
print(math.sqrt(mean_squared_error(y_test,y_pred)))
print(math.sqrt(mean_squared_error(y_test,y_pred2)))

close_price = pd.DataFrame(close_price)
x_input=close_price.tail(50)
x_input_new=np.array(x_input).reshape(1,50,1)
new_price=model.predict(x_input_new)
```

```
new_price2=model2.predict(x_input_new)
new_price=scaler.inverse_transform(new_price)
new_price2=scaler.inverse_transform(new_price2)
```

7 Zaključak

U ovom diplomskom radu uvedene su definicije osnovnih pojmova i zakonitosti koje se odnose na neuronske mreže te je detaljno opisan proces učenja i implementacije rekurzivnih i posebno rekurentnih neuronskih mreža. Promatrane vrste neuronskih mreža služe kao moćan alat za analizu sekvencijalnih i strukturalnih podataka. U drugom dijelu rada provodi se programska realizacija rekurentne neuronske mreže na primjeru predviđanja završnih cijena dionica tvrtke Apple. Rekurzivne neuronske mreže vrlo su primjenjive u mnogim drugim aspektima poput obrade prirodnog jezika i slika. Zbog njihove mogućnosti modeliranja složenih problema, ključne su u područjima poput umjetne inteligencije i strojnog učenja. U posljednjih nekoliko desetljeća neuronske mreže doživjele su vrlo velik napredak te se sve više stručnjaka u području umjetne inteligencije usmjeravaju prema daljnjem razvoju istih. Unaprijeđenja se usmjeravaju na poboljšanje arhitekture, efikasnosti učenja zajedno sa novim tehnikama za interpretaciju i analizu rezultata dobivenih navedenim modelima. Razvoj i unaprijeđenje neuronskih mreža svih vrsta doprinjet će procvatu tehnologija u području umjetne inteligencije i strojnog učenja.

Popis slika

1 Građa biološkog neurona [6] 4
2 Građa jednoslojnog perceptrona [7] 5
3 Područje klasifikacije u dvije dimenzije [7] 6
4 Neuronska mreža 10
5 Neuronska mreža [8] 11
6 Unaprijedna neuronska mreža [11] 15
7 Višeslojni perceptron 17
8 Konvolucijska neuronska mreža [13] 18
9 Modularna neuronska mreža [10] 19
10 Proces razmotavanja [18] 21
11 Dijagram rekurentne neuronske mreže [19] 22
12 Razmotavanje rekurentne neuronske mreže [19] 23
13 Usporedba arhitekture rekurzivnih i rekurentnih neuronskih mreža [20] 24
14 Varijable prikazane pomoću vektora [21] 25
15 Raspored kuhanja [21] 25
16 Predviđanje sutrašnjeg jela [21] 26
17 Vremenska prognoza [21] 26
18 Zbrajanje vektora hrane i vektora vremena [21] 26
19 Izlazni podatak [21] 27
20 Rekurentna neuronska mreža [21] 27
21 Struktura LSTM-a [19] 28
22 Stanje stanice [19] 28
23 Forget gate layer [19] 29
24 Kreiranje nove informacije [19] 30
25 Kreiranje kandidata za novu vrijednost [19] 30
26 Kreiranje izlaznog podatka [19] 30
27 Baza podataka 33
28 Baza podataka 34
29 Analiza podataka 34
30 Analiza podataka 35
31 Analiza podataka 35
32 Statistička obrada podataka 36
33 Završne cijene 37
34 Završne cijene 37
35 Skalirane završne cijene 39
36 Dimenzije dobivenih podataka 40
37 Nove dimenzije skupova podataka 41
38 Jedan element niza X_train 41

39 Rekurentna neuronska mreža 43
40 Grafički prikaz točnih i predviđenih vrijednosti 45
41 Završne cijene prikazane u tablici 45
42 Usporedba predviđenih vrijednosti dvije rekurentne neuronske mreže 47

Literatura

[1] https://www.historyofinformation.com/detail.php?entryid=782 (pristup sadržaju stranice: 05.03.2023.)

[2] https://en.wikipedia.org/wiki/Frank_Rosenblatt (pristup sadržaju stranice: 05.03.2023.)

[3] https://www.amazon.com/Perceptrons-Introduction-Computational-Geometry-Expanded/dp/0262631113 (pristup sadržaju stranice: 05.03.2023.)

[4] https://www.engati.com/glossary/vanishing-gradient-problem (pristup sadržaju stranice: 20.03.2023.)

[5] https://svelezg.medium.com/krizhevsky-sutskever-hinton-and-imagenet-d1c7f2fba113 (pristup sadržaju stranice: 20.03.2023.)

[6] https://hr.wikipedia.org/wiki/Neuron (pristup sadržaju stranice: 26.03.2023.)

[7] https://www.fer.unizg.hr/_download/repository/04-Perceptron.pdf (pristup sadržaju stranice: 01.04.2023.)

[8] https://visualstudiomagazine.com/articles/2013/05/01/neural-network-feed-forward.aspx (pristup sadržaju stranice: 15.04.2023.)

[9] Peyré, Gabriel. Mathematics of Neural Networks (pristup sadržaju stranice: 15.04.2023.)

[10] https://www.mygreatlearning.com/blog/types-of-neural-networks/ (pristup sadržaju stranice: 25.04.2023.)

[11] https://robertbeisicht.wordpress.com/2014/07/04/feed-forward-neural-network-in-javascript/ (pristup sadržaju stranice: 24.04.2023.)

[12] https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/ (pristup sadržaju stranice: 24.04.2023.)

[13] https://www.analyticssteps.com/blogs/convolutional-neural-network-cnn-graphical-visualization-code-explanation (pristup sadržaju stranice: 24.04.2023.)

[14] https://medium.com/@evertongomede/radial-basis-function-networks-rbfn-a-powerful-tool-in-machine-learning-ef3b040c202e (pristup sadržaju stranice: 24.04.2023.)

[15] https://www.researchgate.net/figure/Advantages-and-disadvantages-of-the-AI-technique-based-classifier_tbl1_332627457 (pristup sadržaju stranice: 24.04.2023.)

[16] https://www.analyticssteps.com/blogs/basics-modular-neural-networks (pristup sadržaju stranice: 24.04.2023.)

[17] https://medium.com/the-modern-scientist/exploring-the-efficacy-and-applications-of-modular-neural-networks-in-modern-ai-3ea3d56950a9 (pristup sadržaju stranice: 24.04.2023.)

[18] https://arxiv.org/ftp/arxiv/papers/0911/0911.3298.pdf (pristup sadržaju stranice: 02.05.2023.)

[19] http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (pristup sadržaju stranice: 10.05.2023.)

[20] https://maryambafandkar.me/recursive-neural-network-vs-recurrent-neural-network/ (pristup sadržaju stranice: 10.05.2023.)

[21] https://www.youtube.com/watch?v=UNmqTiOnRfg (pristup sadržaju stranice: 19.05.2023.)

[22] https://en.wikipedia.org/wiki/Python_(programming_language) (pristup sadržaju stranice: 08.08.2023.)

[23] https://www.moj-bankar.hr/Kazalo/D/Dividende(pristup sadržaju stranice: 10.08.2023.)

[24] https://www.kaggle.com/code/rohiniselvaraj0107/stock-prediction-using-lstm-of-apple-inc/input (pristup sadržaju stranice: 01.09.2023.)

[25] https://pandas.pydata.org/docs/ (pristup sadržaju stranice: 02.09.2023)

[26] https://docs.python.org/3/library/csv.html (pristup sadržaju stranice: 02.09.2023)

[27] https://wiki.python.org/moin/NumPy (pristup sadržaju stranice: 15.09.2023.)

[28] https://matplotlib.org/stable/tutorials/pyplot.html (pristup sadržaju stranice: 01.10.2023.)

[29] https://matplotlib.org/ (pristup sadržaju stranice: 02.10.2023.)

[30] https://medium.com/codex/why-scaling-your-data-is-important-1aff95ca97a2 (pristup sadržaju stranice: 02.11.2023.)

[31] https://en.wikipedia.org/wiki/TensorFlow (pristup sadržaju stranice: 03.12.2023.)

[32] https://www.tensorflow.org/guide/keras (pristup sadržaju stranice: 03.12.2023.)

[33] https://aitechtrend.com/mastering-dense-layers-the-key-to-neural-network-success/ (pristup sadržaju stranice: 03.12.2023.)

[34] https://www.fer.unizg.hr/_download/repository/01-Uvod-1s.pdf (pristup sadržaju stranice: 13.04.2023.)

[35] https://www.w3schools.com/ai/ai_perceptrons.asp (pristup sadržaju stranice: 05.04.2023.)

[36] https://www.yourdatateacher.com/2021/05/10/how-many-neurons-for-a-neural-network/ (pristup sadržaju stranice: 23.04.2023.)

[37] https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python (pristup sadržaju stranice: 04.08.2023.)

[38] https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/ (pristup sadržaju stranice: 15.05.2023.)

[39] https://docs.python.org/3/library/math.html(pristup sadržaju stranice: 10.11.2023.)