Analiza problema triju tijela pomoću BiLSTM neuralne mreže

Sfarčić, Frane

Sveučilište u Splitu
Prirodoslovno-matematički fakultet

ANALIZA PROBLEMA TRIJU TIJELA POMOĆU BiLSTM NEURALNE MREŽE

Diplomski rad

Frane Sfarčić

Split, rujan 2024.

Temeljna dokumentacijska kartica

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za fiziku
Ruđera Boškovića 33, 21000 Split, Hrvatska

Diplomski rad

Analiza Problema triju tijela pomoću BiLSTM neuralne mreže

Frane Sfarčić

Sveučilišni diplomski studij Fizika, smjer Računarska Fizika

Sažetak:

Problem triju tijela u fizici prepoznat je kao složena dinamika između triju tijela s masom. Ta tri tijela međusobno djeluju gravitacijskom silom. Princip računanja njihovog gibanja uz poznate početne uvjete je poznat. Već mala promjena jednog parametra u početnim uvjetima može radikalno promijeniti dinamiku gibanja, čineći je gotovo kaotičnom. U ovom radu, prepoznajući mogućnosti neuralnih mreža, pokušava se pristupiti problemu s novim pogledom. Računalnim generiranjem početnih uvjeta numeričkim metodama prikupljeni su podaci o položajima triju tijela, kao i o njihovim brzinama te međusobnim udaljenostima. U kodu su implementirane sile. Metodom Runge-Kutta izračunati su vremenski koraci, a evolucije sustava su spremane u datoteke pogodne za Long Short-Term Memory (LSTM) rekurzivne neuralne mreže. Kroz različite evolucije sustava moglo se učiti o sustavu s ciljem stvaranja računalnog modela koji uključuje dinamiku problema triju tijela. Također se pokušalo predvidjeti sljedeći korak u evoluciji sustava.

Korišten je programski jezik Python i dovoljno snažno računalo kako bi se osiguralo pogodno okruženje za rad. Neuralne mreže zahtijevaju snažne grafičke kartice i mnogo kalibriranja kako bi se postiglo rješenje. Ovi rezultati su pokazali da su takve metode obećavajuće, no budući da se radi o složenom sustavu, potrebno je mnogo podataka kako bi mreža što bolje učila. S trenutnim mogućnostima postignuti su vrlo dobri rezultati. Za daljnji napredak mogu se koristiti još snažnija računala, više podataka te daljnje kalibriranje postavki.

Ključne riječi: Problem triju tijela, BiLSTM, Runge-Kutta, Numeričke metode, Neuralna mreža

Rad sadrži: 39 stranica, 22 slike, 3 tablice, 19 literaturnih navoda. Izvornik je na hrvatskom jeziku

Mentor: doc. dr. sc. Marko Kovač

Ocjenjivači: doc. dr. sc. Marko Kovač
izv. prof. dr. sc. Petar Stipanović
doc. dr. sc. Toni Šćulac

Rad prihvaćen: 27. rujna 2024.

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu.

Basic documentation card

University of Split
Faculty of Science
Department of Physics
Ruđera Boškovića 33, 21000 Split, Croatia

Master thesis

Analysis of the Three-Body Problem using a BiLSTM Neural Network

Frane Sfarčić

University graduate study of Physics specialization in Computational Physics

Abstract:

The three-body problem in physics is recognized as a complex dynamic interaction between three bodies with mass. These bodies interact with each other through gravitational forces. While the principle of calculating their motion given initial conditions is known, even a slight change in one parameter can alter the dynamics so radically that the system almost becomes chaotic. In this paper, recognizing the potential of neural networks, we attempt to approach the problem from a new perspective.

By generating initial conditions numerically through computational methods, data on the positions of the three bodies were collected, along with their velocities and mutual distances. The forces acting between them were also implemented in the code. Using the Runge-Kutta method, time steps were calculated, and the system's evolutions were saved in files suitable for Long Short-Term Memory (LSTM) recurrent neural networks. By analyzing various system evolutions, insights into the system and the laws governing its dynamics could be deduced. Predictions of the next step in the system's evolution were attempted.

The Python programming language and a sufficiently powerful computer were used to create a suitable working environment. Neural networks require strong graphics cards and extensive calibration to obtain accurate solutions. From these results, it was concluded that these methods are promising. However, given the complexity of the system, a significant amount of data is needed for the network to learn effectively. With the resources available, quite good results were achieved. For further advancements, even more powerful computers, more data, and further calibration of settings will be necessary.

Keywords: Three-body problem, BiLSTM, Runge-Kutta, Numerical methods, Neural network.

Thesis consists of: 39 pages, 22 figures, 3 tables, 19 references. Original language: Croatian

Supervisor: Assist. Prof. Dr. Marko Kovač

Reviewers: Assist. Prof. Dr. Marko Kovač
Assoc. Prof. Dr. Petar Stipanović
Assist. Prof. Dr. Toni Šćulac

Thesis accepted: September 27. 2024.

Thesis is deposited in the library of the Faculty of Science, University of Split.

Sadržaj

1 Uvod

Da bi se došlo do znanja o prirodi, potrebno je osjetiti okruženje i postavljati pitanja. Jednostavna pitanja nekada vode do jednostavnih odgovora, ali isto tako često mogu voditi do složenih odgovora ili nerješivih zavrzlama. Mnogi su se pitali kako dva tijela gravitacijski međudjeluju bez utjecaja dodatnih sila. Proučavali su i dolazili do spoznaja. Dodavanjem trećeg tijela međudjelovanje ostaje uvjetovano istim zakonima, ali gibanja postaju mnogo kompliciranija. Dodavanje dodatnih tijela ili razumijevanje gibanja realnog sustava, kao što je zrak, postaje gotovo nemoguće koristeći zakon gravitacije do kojeg je u 17. stoljeću došao Newton. Henri Poincaré je pokazao da nema analitičkog rješenja za taj problem [1]. Jaka osjetljivost na promjene početnih uvjeta otvorila je vrata teoriji kaosa. Mnogi matematičari i fizičari su proučavali taj problem, čime su produbili znanje, ali područje i dalje nema jednostavne odgovore.

Pokušavajući doći do znanja, uvijek koristimo trenutne mogućnosti tehnologije. Niels Bohr je ustanovio teoriju o građi atoma koristeći tada aktualnu kvantnu teoriju [2]. Skupina fizičara R. Brout, F. Englert, P. Higgs, G. S. Guralnik, C. R. Hagen i T. W. B. Kibble (1964.) predvidjela je česticu koju tadašnja tehnologija nije mogla eksperimentalno potvrditi [3]. Tek desetljećima kasnije, izgradnjom akceleratora u Ženevi, potvrđeno je postojanje Higgsovog bozona. Današnja napredna računala omogućuju brže i lakše proračune. Pojam umjetne inteligencije potječe od Alana Turinga, koji je zamislio 'univerzalni stroj' ili Turingov stroj [4]. Napravio je Turingov test, kojim se testira sposobnost stroja da iskaže inteligentno ponašanje koje se ne razlikuje od čovjekovog. Godine 1956. izraz 'umjetna inteligencija' osmislio je John McCarthy [5]. Kroz 1980-e uvedene su rekurzivne neuralne mreže [6], a 1997. godine izumljena je 'Long Short-Term Memory' mreža [7], značajna za ovaj rad.

U ovom radu opisuje se numeričke metode generiranja podataka, potrebnih za učenje neuralne mreže. Daje se kratko fizikalno objašnjenje problema triju tijela. Zatim se prolazi teorija BiLSTM NM (neuralna mreža). Objašnjavaju se računalne metode potrebne za izvođenje svih koraka i računalni alati. Objašnjava se cilj učenja same NM te se prikazuju rezultati i zaključci koji iz njih proizlaze.

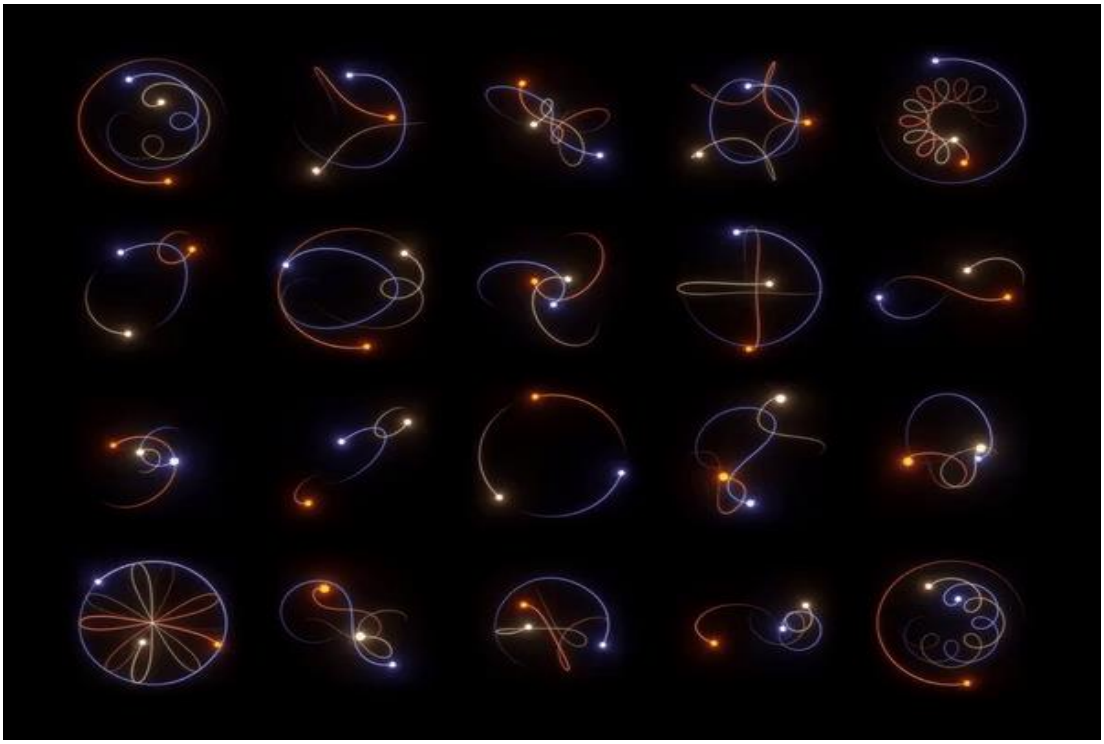
2 Problem triju tijela

Problem triju tijela je klasičan problem u mehanici koji predviđa gibanje tri tijela koja međusobno gravitacijski djeluju. Ovaj problem uključuje početne položaje i početne brzine triju tijela koja u našem slučaju imaju masu jednaku jedinici. Zatim se računaju njihova posljedična gibanja uzrokovana Newtonovim jednadžbama gibanja i Newtonovim zakonom univerzalne gravitacije [8].

$$\overrightarrow{a_1}(t) = Gm_2 \frac{\overrightarrow{r_2}(t) - \overrightarrow{r_1}(t)}{|\overrightarrow{r_2}(t) - \overrightarrow{r_1}(t)|^3} + Gm_3 \frac{\overrightarrow{r_3}(t) - \overrightarrow{r_1}(t)}{|\overrightarrow{r_3}(t) - \overrightarrow{r_1}(t)|^3} \tag{2.1}$$

$$\overrightarrow{a_2}(t) = Gm_1 \frac{\overrightarrow{r_1}(t) - \overrightarrow{r_2}(t)}{|\overrightarrow{r_1}(t) - \overrightarrow{r_2}(t)|^3} + Gm_3 \frac{\overrightarrow{r_3}(t) - \overrightarrow{r_2}(t)}{|\overrightarrow{r_3}(t) - \overrightarrow{r_2}(t)|^3} \tag{2.2}$$

$$\overrightarrow{a_3}(t) = Gm_1 \frac{\overrightarrow{r_1}(t) - \overrightarrow{r_3}(t)}{|\overrightarrow{r_1}(t) - \overrightarrow{r_3}(t)|^3} + Gm_2 \frac{\overrightarrow{r_2}(t) - \overrightarrow{r_3}(t)}{|\overrightarrow{r_2}(t) - \overrightarrow{r_3}(t)|^3} \tag{2.3}$$



Slika 1: *Primjeri periodičnih gibanja triju tijela (Slika preuzeta sa [9])*

U jednadžbama (2.1), (2.2) i (2.3) masa je jednaka jedan za sva tijela, a gravitacijska konstanta također jedan. Za simulacije ti parametri nisu bili važni; bitno je bilo računati njihovu interakciju opisanu formulama. Periodične putanje moguće je dobiti samo za posebne slučajeve

gdje su rješenja simetrična u prostoru i vremenu. Takva gibanja su prikazana na Slika 1, gdje je prikazano 20 periodičnih gibanja. Za ovakve orbite, mase trebaju biti iste te početni uvjeti ekstremno kalibrirani tj. precizno postavljeni položaji i brzine. Mala odstupanja od idealnih uvjeta mogu dovesti do kaotičnog ponašanja, pa početne vrijednosti moraju biti izuzetno točne. Čak i u simulacijama, numeričke greške se akumuliraju, stoga je kalibracija potrebna kako bi numeričke metode bile dovoljno precizne da održe točnost potrebnu za opažanje periodičnog gibanja. Sustav mora održavati simetriju (npr. jednake udaljenosti ili kutove između tijela) cijelo vrijeme. Sitne greške mogu narušiti simetriju i spriječiti periodično ponašanje. Ovakvi slučajevi u realnom sustavu su skoro nemogući. Ako su početni uvjeti poznati, putanja se može točno izračunati. Ako nisu periodična gibanja, nakon nekog vremena putanje se ne mogu precizno izračunati. Ne postoji generalno analitičko rješenje za bilo kakvu konfiguraciju početnih uvjeta. Matematičar Karl Fritiof Sundman je 1912. godine dokazao da postoji analitičko rješenje u obliku Puiseuxovih nizova, osobito nizova na potenciju $t^{1/3}$. Ovaj niz konvergira za sve realne brojeve *t*. U praksi, niz konvergira toliko sporo da bi za astronomska promatranja bila potrebna barem $10^{8000000}$ članova da bi se postigla smislena preciznost, što ga čini iznimno nepraktičnim [10].

2.1 Runge-Kutta metoda

Danas uz pomoć računala možemo znatno olakšati izračune putanja. No, kako računalo ima svoja ograničenja, ne mogu se koristiti infinitezimalno mali brojevi za diferencijalne jednadžbe. Zbog toga će se raditi aproksimacije koje će biti dovoljno dobre za vremenski korak koji je suvisao.

Runge-Kutta metode su obitelj iterativnih tehnika koje se koriste za rješavanje običnih diferencijalnih jednadžbi. Runge-Kutta metoda četvrtog reda (RK4) jedna je od najšire korištenih zbog svoje ravnoteže između točnosti i računalne učinkovitosti [11]. Ako je početni oblik dan kao:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0 \tag{2.4}$$

Cilj je pronaći vrijednost varijable y u nekom kasnijem vremenu $t = t_0 + h$, gdje je h veličina koraka. RK4 metoda pruža aproksimaciju za $y(t_0 + h)$ koristeći sljedeće jednadžbe:

$$k_1 = hf(t_n, y_n) \tag{2.5}$$

$$k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \tag{2.6}$$

$$k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \tag{2.7}$$

$$k_4 = hf(t_n + h, y_n + k_3) \tag{2.8}$$

Onda ažurirajući vrijednost y :

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.9}$$

Na kraju se pomiče i vremenski korak: $t_{n+1} = t_n + h$.

2.1.1 Runge-Kutta-Fehlberg

U ovome radu, koristila se Fehlbergovom metoda koja uključuje 6 međukoraka koji su dani u jednadžbama ispod:

$$k_1 = h \cdot f(t_n, y_n) \tag{2.10}$$

$$k_2 = h \cdot f\left(t_n + \frac{1}{4}h, y_n + \frac{1}{4}k_1\right) \tag{2.11}$$

$$k_3 = h \cdot f\left(t_n + \frac{3}{8}h, y_n + \frac{3}{32}k_1 + \frac{9}{32}k_2\right) \tag{2.12}$$

$$k_4 = h \cdot f\left(t_n + \frac{12}{13}h, y_n + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right) \tag{2.13}$$

$$k_5 = h \cdot f\left(t_n + h, y_n + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right) \tag{2.14}$$

$$k_6 = h \cdot f\left(t_n + \frac{1}{2}h, y_n - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right) \tag{2.15}$$

Ovako izgleda formula za sljedeći korak, služeći se sa gore navedenim međukoracima:

$$y_{n+1} = y_n + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6 \tag{2.16}$$

2.1.2 Pogreška

Za procjenu pogreške napravljen je jedan puni korak za $y(t + h)$. Zatim je izračunat y nakon pola koraka $h/2$. Služeći se ovim međukorakom, izračunat je y nakon još jednog polukoraka $h/2$. To predstavlja aproksimaciju za rješenje $(t + h)$ uzimajući polukorake y^{half}.

Razlika između dva rezultata $y^{\text{full}} - y^{\text{half}}$ pokazuje koliko se račun mijenja korištenjem manje ili veće veličine koraka. Izračunava se apsolutna i relativna pogreška:

$$\text{apsolutna\_pogreška} = |y^{\text{half}} - y^{\text{full}}| \tag{2.17}$$

$$\text{relativna\_pogreška} = \frac{\text{apsolutna\_pogreška}}{|y^{\text{half}}| + \epsilon} \tag{2.18}$$

Gdje je ϵ neki maleni broj koji osigurava da u razlomku ne dijelimo s 0. Onda se računa pogreška uzimajući najveću pogrešku između relativne i apsolutne.

$$\text{error} = max(\text{apsolutna\_pogreška}, \text{relativna\_pogreška}) \tag{2.19}$$

Ovo osigurava da procjena pogreške uzima u obzir i apsolutnu i relativnu skalu pogreške, čineći je robusnom za različite y. Adaptivno podešavanje koraka koristi procijenjenu pogrešku kako bi se prilagodila veličina koraka h za kontrolu točnosti rješenja. Ako je pogreška unutar prihvatljive granice (target_error), korak se prihvaća. Ako pogreška premašuje dopuštenu granicu, veličina koraka se smanjuje, a korak se ponovno izračunava.

Nova veličina koraka h_{new} računa se na temelju trenutne veličine koraka h, ciljane pogreške (target_error) i procijenjene pogreške:

$$h_{\text{new}} = h \times \left(\frac{\text{error}}{\text{target\_err}}\right)^{0.25} \times 0.9 \tag{2.20}$$

Eksponent 0.25 odgovara činjenici da je metoda četvrtog reda (pogreška je proporcionalna s h_4). Faktor 0.9 je sigurnosni faktor koji osigurava da je nova veličina koraka dovoljno konzervativna da vjerojatno zadovolji toleranciju pogreške u sljedećem koraku. Nadalje, postoje još dodatni koraci opreza:

$$h_{\text{new}} = max\left(min(h_{\text{new}}, h \times \text{s\_factor}), \frac{h}{\text{s\_factor}}\right) \tag{2.21}$$

$$h_{\text{new}} = max(min(h_{\text{new}}, h_{\text{max}}), h_{\text{min}}) \tag{2.22}$$

Ovo znači da se h_{new} ne može previše razlikovati od trenutne veličine koraka h za više od faktora s_factor. Faktor s_factor kontrolira koliko agresivno se veličina koraka prilagođava na temelju procjene pogreške. Veća vrijednost s_factor omogućuje agresivnije promjene veličine koraka, što potencijalno može rezultirati bržim konvergencijama, ali istovremeno može učiniti metodu manje stabilnom. Suprotno tome, manja vrijednost s_factor rezultirala bi konzervativnijim promjenama veličine koraka, osiguravajući stabilnost, ali možda na račun sporije konvergencije.

3 Neuronske Mreže

Neuronske mreže [12] su vrsta umjetne inteligencije koja imitira način na koji ljudski mozak obrađuje informacije. Ova tehnologija koristi strukturu inspiriranu biološkim neuronskim mrežama, gdje su osnovne jedinice "neuroni" povezane u složene mreže.

Neuronske mreže se sastoje od više slojeva (eng. *Layers*): ulazni sloj, skriveni slojevi i izlazni sloj. Svaki sloj se sastoji od velikog broja čvorova ili "neurona" koji su povezani s čvorovima u sljedećem sloju, ilustrirano na Slika 2. Kada se ulazni podaci unesu u mrežu, oni prolaze kroz ove slojeve. Dok prolaze, svaki neuron obrađuje podatke pomoću određene matematičke funkcije.

Učenje u neuronskim mrežama odvija se kroz proces zvan "treniranje". Tijekom treniranja, mreža koristi skup podataka za prilagodbu težina veza između neurona kako bi poboljšala točnost svojih predikcija ili klasifikacija. Najčešći algoritam koji se koristi za treniranje je unazadna propagacija (eng. *Backpropagation*). Ona pomaže mreži da minimizira razliku između predviđenih i stvarnih rezultata.

Njihova sposobnost da uče i prilagođavaju se iz podataka čini ih izuzetno moćnim alatima za rješavanje složenih problema.

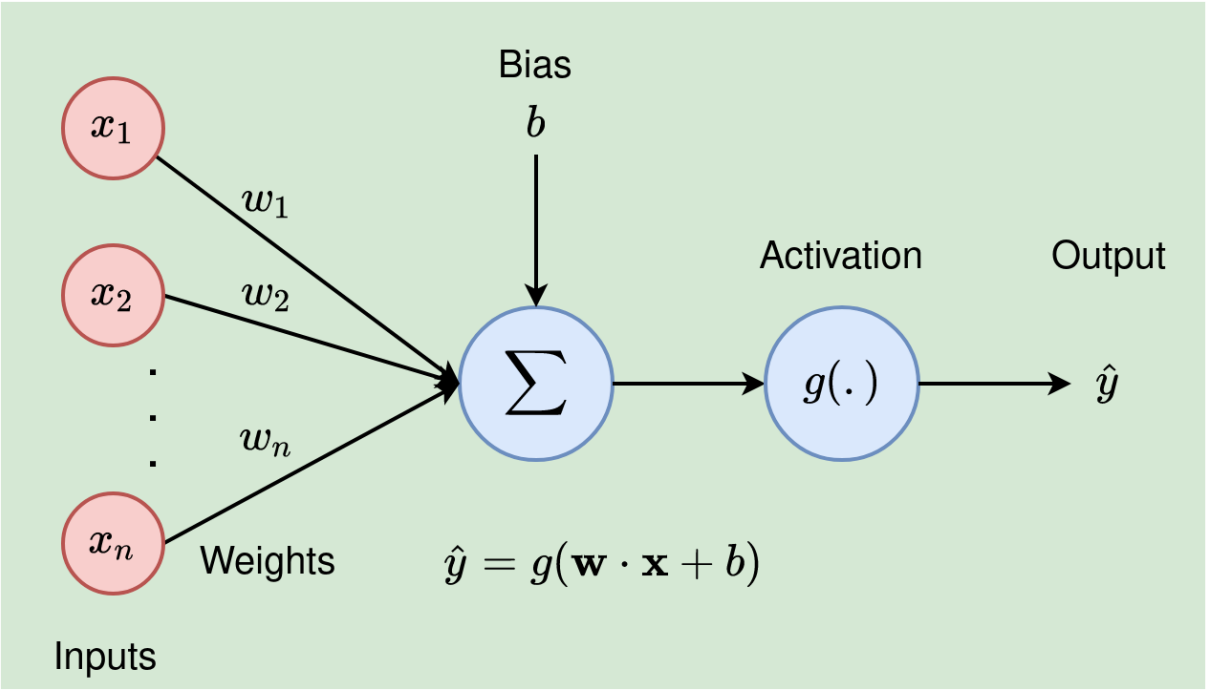
U posljednjih nekoliko godina, napredak u hardverskim tehnologijama kao što su grafički procesori (eng. *GPU*) i specijalizirani akceleratori za neuronske mreže, kao i dostupnost velikih količina podataka, značajno su ubrzali razvoj i primjenu neuronskih mreža. Zbog toga su neuronske mreže postale temelj mnogih modernih tehnoloških rješenja i inovacija.

3.1 Propagacija unaprijed

Propagacija unaprijed (eng. *Forward pass*) je proces izračunavanja izlaza (eng. *Output*) neuronske mreže za zadani ulaz (eng. *Input*) prolaskom kroz sve slojeve mreže. To uključuje:

- **Ulazni podaci**: Uvođenje ulaznih podataka u mrežu.
- **Izračuni u slojevima**: Svaki sloj obavlja svoje izračune (npr. množenje matrica, aktivacije) kako bi transformirao ulazne podatke.
- **Izračun izlaza**: Proizvodnja konačnog izlaza mreže.

Tijekom prolaza naprijed, predikcije modela se temelje na trenutnom stanju njegovih parametara težina (eng. *Weight*) i pristranosti (eng. *Bias*).



Slika 2: *Shematski prikaz neurona u neuralnoj mreži.* $x_1 \ldots x_n$ *su ulazni podaci,* $\hat{y}$ *su izlazi. (Slika preuzeta sa [13]).*

Formula koja prikazuje kako neuron izračunava ispis je dana sa ovim izrazom.

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3.1}$$

Gdje je ϕ aktivacijska funkcija koja uzima linearnu kombinaciju linearnih funkcija zasebnih ulaza x. Svrha ϕ je da uvede nelinearnost u inače linearnu funkciju. Nekoliko funkcija je odabrano kao najpogodnijih:

- $Sigmoid$: $\phi(z) = \frac{1}{1+e^{-z}}$ (3.2)
- $HyperbolicTangent$: $\phi(z) = \frac{e^{2z}-1}{e^{2z}+1}$ (3.3)
- $ReLU$: $\phi(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$ (3.4)
- $Leaky\ ReLU$: $f(z) = \begin{cases} z, & z \geq 0 \\ \alpha z, & z < 0 \end{cases}$ (3.5)
- $Softmax$: $\phi(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$ (3.6)

Gdje je z u jednadžbama od (3.2) do (3.6), $z = \sum_{i=1}^{n} w_i x_i + b$.

Sigmoid funkcija ima glatki gradijent i rezultate koji su u rasponu (0,1), što je dobro za probabilističke interpretacije. Mana joj je što može imati problem s nestajanjem gradijenta za velike pozitivne ili negativne ulaze, što može usporiti ili čak zaustaviti trening. Kada se koristi metoda propagacija unatrag za ažuriranje težina u neuronskoj mreži, ključni korak je računanje gradijenata (derivacija) funkcije gubitka u odnosu na težine svakog sloja. Ti gradijenti se zatim koriste za prilagodbu težina kako bi se smanjila ukupna pogreška.

Međutim, kod dubokih mreža gradijenti često postaju vrlo mali (blizu nuli) dok prolaze kroz mnoge slojeve unatrag, što dovodi do toga da se težine u početnim slojevima gotovo uopće ne ažuriraju. Ovaj efekt nazivamo "nestajući gradijent".

Za hyperbolic tangent, raspon je (-1,1) i vrijednosti su simetrične oko nule, što može olakšati optimizacije. Mana je slična sigmoidu, nestajajući gradijent.

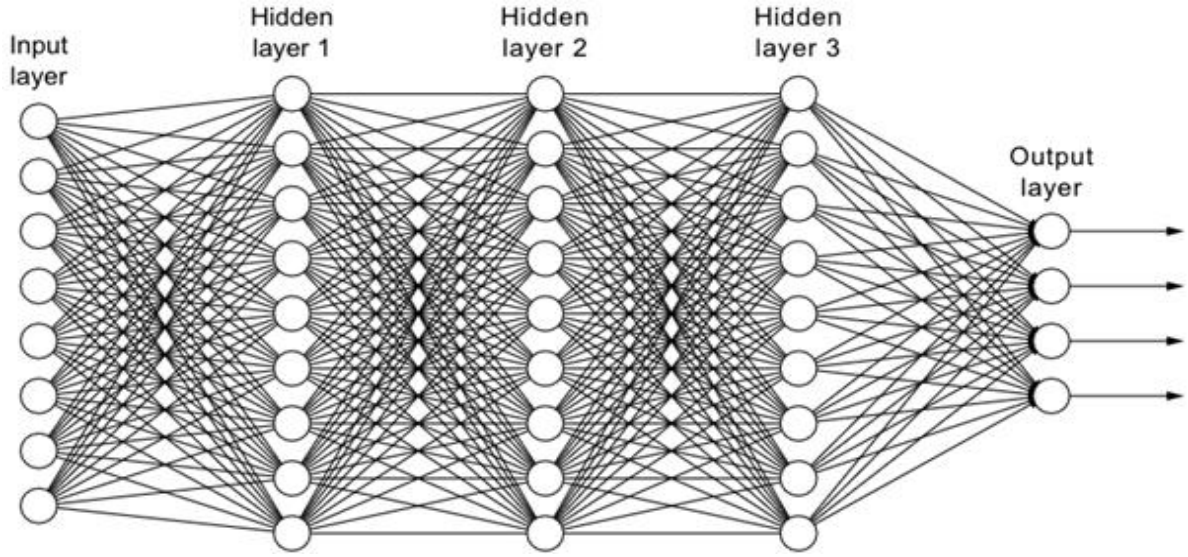
ReLU ima raspon $[0,\infty)$. Prednost je što je računalno efikasan, pomaže u ublažavanju problema nestajanja gradijenta te promovira rasprostiranje informacija unutar mreže. Mana je mogući problem umirućeg ReLU-a, gdje neuroni mogu postati neaktivni i prestati učiti. Umirući ReLU se događa kada neuron neprekidno prima negativne ulazne vrijednosti tijekom treninga. Ako je ulaz u ReLU funkciju često negativan, tada će izlaz ReLU-a stalno biti 0. Ako se to dogodi tijekom treniranja mreže, derivacija (gradijent) će također biti 0, što znači da taj neuron neće više učiti.

Leaky ReLU ima raspon $(-\infty,\infty)$ i ublažava problem umirućeg ReLU-a dopuštajući malen gradijent koji nije nula kada je ulaz negativan. Zadržava mnoge prednosti ReLU-a kao što su računalna efikasnost i rasprostranjenost informacija unutar mreže. Mana je osjetljivost performansi na izbor parametra α, te i dalje postoji rizik od neaktivnih neurona, iako je znatno smanjen u usporedbi s običnim ReLU-om.

Softmax se često koristi u izlaznom sloju neuronskih mreža za klasifikaciju jer pretvara ulaze u vektor vjerojatnosti. Svaka vjerojatnost za svaku klasu ima raspon od 0 do 1, pri čemu se sve vjerojatnosti zbrajaju do 1 za sve klase. Softmax pruža jasno probabilističko tumačenje izlaza, što je korisno za donošenje odluka i razumijevanje povjerenja modela. Računalno je zahtjevan zbog eksponencijalnih i normalizacijskih koraka, što je posebno istaknuto kod velikog broja klasa. Softmax može biti osjetljiv na velike ulazne vrijednosti. To može uzrokovati numeričku nestabilnost. Ovo se često može ublažiti oduzimanjem maksimalne ulazne vrijednosti prije primjene funkcije.

Parametri w i b na Slika 2 nazivaju se težinama i pristranostima. Ove varijable su "naučene" jer su početno postavljene nasumično. Međutim, ako imamo neke podatke s poznatim ulazima i poznatim izlazima, možemo prilagoditi model prema željenoj konvergenciji. Time definiramo težine i pristranosti. Radimo sumaciju po parametrima i onda puštamo kroz aktivacijsku funkciju $g(*)$. Kombinacija neurona definira mrežu. Ovisno koliko kompliciranu mrežu želimo implementirati, dodajemo slojeve i tako dobivamo skrivene slojeve. Na Slika 3, se može vidjeti generalni prikaz slojeva. Mreže se dijele na slojeve:

- Prvi sloj su ulazi, njih ne ubrajamo kao sloj u brojanju slojeva.
- Izlaz iz prvog sloja koristi se kao ulaz u sljedeći sloj proizvoljne veličine, a taj se proces ponavlja kako bi se stvorila dublja mreža.
- Posljednji sloj je izlaz mreže, koji može biti broj, vektor, tenzor, itd. Ponovno, kao i sa prvim slojem, često se ne uključuje u brojanje.



Slika 3: *Shematski prikaz slojeva neuralne mreže (Slika preuzeta sa [13]).*

3.2 Propagacija unatrag

Propagacija unatrag (eng. *backpropagation*) je algoritam koji se koristi za treniranje neuronskih mreža. Omogućuje izračun gradijenata funkcije gubitka (eng. *Loss function*) u odnosu na parametre mreže, što je ključno za ažuriranje tih parametara tijekom treniranja. Nakon što je napravljena propagacija unaprijed i dobivena predviđanja mreže, koristi se

funkcija gubitka za usporedbu predikcija s stvarnim vrijednostima. Funkcija gubitka mjeri koliko su predviđanja mreže udaljene od stvarnih vrijednosti.

"Učenje" se provodi sljedećim koracima: Neka je y' vektor predviđenih vrijednosti, a y vektor stvarnih opažanja. Funkcija $L(y, y')$ nazvana funkcija gubitka, definirana je na način da minimizacija ove funkcije daje pravilo ažuriranja težina w i pristranosti b sve dok se ne dostigne globalni minimum te funkcije. Neke od korištenih funkcija gubitka su:

Cross entropy: $L(y, y') = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot log(y_i')$ (3.7)

Mean square: $L(y, y') = -\frac{1}{m}\sum_{i=1}^{m}(y_i - y_i')^2$ (3.8)

Mean absolute value: $L(y, y') = -\frac{1}{m}\sum_{i=1}^{m}|y_i - y_i'|$ (3.9)

Root mean squared error: $L(y, y') = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2}$ (3.10)

3.2.1 Propagacija unaprijed

Opisati ćemo korake kod propagacije unatrag.

Predaktivacija skrivenog sloja: $z_1 = w_1 x + b_1$

Aktivacija skrivenog sloja: $a_1 = \mathrm{ReLU}(z_1) = \max(0, z_1)$

Predaktivacija izlaznog sloja: $z_2 = w_2 a_1 + b_2$

Izlaz: $\hat{y} = z_2$.

Uzimamo funkciju gubitka izraženu u jednadžbi (*3.10*) jer je ona korištena u radu.

Za jedan uzorak (gdje je $N = 1$): $L = \sqrt{(\hat{y} - y)^2} = |\hat{y} - y|$

3.2.2 Gradijenti izlaznog sloja

Derivacija gubitka u odnosu na predviđeni izlaz $\hat{y}$: $\frac{\partial L}{\partial \hat{y}} = \frac{\hat{y} - y}{|\hat{y} - y|} = \mathrm{sign}(\hat{y} - y)$

Derivacija u odnosu na predaktivaciju izlaznog sloja z_2: $\frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial \hat{y}}$

Gradijenti u odnosu na w_2 i b_2: $\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_2} \cdot a_1$ $\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_2}$

3.2.3 Gradijenti skrivenog sloja.

Derivacija u odnosu na aktivaciju skrivenog sloja a_1: $\frac{\partial L}{\partial a_1} = \frac{\partial L}{\partial z_2} \cdot w_2$

Derivacija u odnosu na predaktivaciju skrivenog sloja z_1: $\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1}$

Gdje: $\frac{\partial a_1}{\partial z_1} = \begin{cases} 1 \ ako \ z_1 > 0 \\ 0 \ ako \ z_1 < 0 \end{cases}$

Gradijenti u odnosu na w_1 i b_1: $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_1} \cdot x \quad \frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1}$

3.2.4 Ažuriranje težina i pomaka.

- $w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$
- $b_1 \leftarrow b_1 - \eta \frac{\partial L}{\partial b_1}$
- $w_2 \leftarrow w_2 - \eta \frac{\partial L}{\partial w_2}$
- $b_2 \leftarrow b_2 - \eta \frac{\partial L}{\partial b_2}$

Ovdje jednostavno koristimo prethodne korake uz parametra η što je stopa učenja (eng. *Learning rate*), da bismo ažurirali težine i pristranost za sljedeću propagaciju unaprijed.

3.3 Optimizacijski algoritmi

U ovom dijelu opisujemo važnu komponentu strojnog učenja, a to su algoritmi optimizacije. Diskretna priroda računanja predstavlja mnoge poteškoće u provedbi matematičkih operacija poput integracije i derivacije. Algoritmi optimizacije predstavljaju skup algoritama koji se bave takvim matematičkim operacijama za različite funkcije. Postoji veliki broj različitih algoritama optimizacije, ali u ovom dijelu predstavljamo samo one koji su važni za ovaj rad.

Funkcija gubitka navedena u prethodnom odjeljku koristi se kako bi se opisala mjera izlaza neuronske mreže kada se samo jedan uzorak značajki koristi kao ulaz. Korištenje više od jednog uzorka značajki kao ulaznih podataka istovremeno naziva se grupiranjem (eng. *batching*). Ovisno o mogućnostima računala, ima različit utjecaj na brzinu konvergencije prema rješenju. Funkcija formirana korištenjem grupiranja naziva se funkcijom troška (eng. *cost function*) i najčešće se označava slovom J, pri čemu se pojam funkcije gubitka rezervira samo za izlaz pojedinačnih uzoraka značajki. Odnosno funkcija troška za grupu veličine n je:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L_i(\theta) \tag{3.11}$$

Algoritmi optimizacije pružaju nam pravila ažuriranja parametara mreže. Broj tih parametara izuzetno je velik. Stoga dijelimo algoritme optimizacije na dva tipa. Algoritmi temeljeni na gradijentu koriste se kada je poznata derivacija funkcije:

- Gradient descent
- Stochastic gradient descent
- Adaptive momentum optimizer
- Nesterov momentum

Algoritmi koji nisu temeljeni na gradijentu koriste se kada derivacija nije poznata ili je previše kompleksna za izračun:

- Genetic algorithms
- Bayes optimiziation
- Random search
- Simulated annealing

Promjena u parametrima u Gradient descent i Stochastic gradient descent razlikuje se u načinu na koji se uzorci značajki daju kao ulaz jedan po jedan. Gradient descent uzima skup uzoraka, dok Stochastic gradient descent uzima samo jedan uzorak. Promjena u parametrima određena je:

$$\theta_{new_i} = \theta_{current_i} - \alpha \frac{\delta J(\theta_1, \dots, \theta_n)}{\delta \theta_i}, \quad i = 1,2, \dots, n \tag{3.12}$$

Gdje je α (stopa učenja) koja predstavlja veličinu koraka određujući koliko mijenjamo trenutnu težinu. Podešavanje ovog parametra je od najveće važnosti. Stopa učenja α definira veličinu koraka koja se koristi u algoritmima temeljenim na gradijentima. Ta veličina koraka određuje veličinu promjene težine u zadanom smjeru. Stopa učenja se postavlja na α = 0,001 pri korištenju gradijentnog spuštanja. Međutim, konvergencija prema rješenju je izuzetno spora. Za složene modele konvergencija može biti eksponencijalno spora. Zbog toga postoje brojni načini za implementaciju tzv. opadanja stope učenja. Kod opadanja stope učenja variramo veličinu α. Obično je smanjujemo od početno postavljene vrijednosti nakon određenog broja simulacijskih koraka n.

3.3.1 ADAM

ADAM (eng. *Adaptive Moment Estimation*) je optimizacijski algoritam koji se koristi u primjenama strojnog učenja i dubokog učenja. To je kombinacija dviju metodologija gradijentnog spuštanja: RMSProp (eng. *Root Mean Square Propagation*) i Momentum.

Kao i RMSProp, ADAM koristi kvadrat gradijenta za skaliranje stope učenja (pristup nazvan adaptivne stope učenja), a kao i Momentum, ADAM prati pomični prosjek gradijenta (pristup nazvan momentum). To čini ADAM algoritmom koji je adaptivan u odnosu na momente.

ADAM proširuje grupno gradijentno spuštanje i ima za cilj poboljšati brzinu konvergencije prema rješenju. Također može riješiti probleme koji proizlaze iz lokalnih minimuma funkcija i funkcijskih platoa.

Inicijalizacija kreće sa postavljanjem hiperparametara: α (stopa učenja), β_1 (parametar za prvi moment), β_2 (parametar za drugi moment), i ε (mala vrijednost za numeričku stabilnost). Postavljanje trenutnih vrijednosti prvog momenta $m_0 = 0$ i drugog momenta $v_0 = 0$.

Za svaki korak iteracije t, računaju se gradijenti g_t za trenutne parametre modela.

Ažuriraju se prvi i drugi momenti:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{3.13}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{3.14}$$

Ispravljanje pristranosti (eng. *Bias Correction*).

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{3.15}$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{3.16}$$

Zbog adaptivnih stopa učenja, ADAM često konvergira brže od drugih algoritama. Dodavanje ε u nazivnik osigurava stabilnost u računanju, čak i kada su gradijenti mali. Dobro funkcionira na velikim i bučnim podacima, kao i u problemima s velikim brojem parametara. Bučni podaci su podaci koji uključuju neprecizne, nepotpune, ili nasumične informacije koje mogu ometati proces učenja modela. U radu je korišten ADAM algoritam.

4 LSTM Neuralna Mreža

Dugoročna kratkoročna memorija (LSTM) je vrsta arhitekture ponavljajućih neuronskih mreža (eng. *Recurrent neural network*) skraćeno RNN. Dizajnirana kako bi prevladala problem nestajućeg gradijenta u tradicionalnim RNN-ovima. Predstavili su je Hochreiter i Schmidhuber 1997. godine. LSTM mreže se široko koriste u različitim zadacima učenja sekvenci. Koriste se kod obrade prirodnog jezika, prepoznavanja govora i predikcije vremenskih serija zbog svoje sposobnosti prepoznavanja obrazaca i dugoročnih trendova u seriji. Omogućuju bolja predviđanja budućih vrijednosti.

U ovom radu je dobro poslužila jer treba predvidjeti sljedeću sekvencu u kretanju triju tijela.

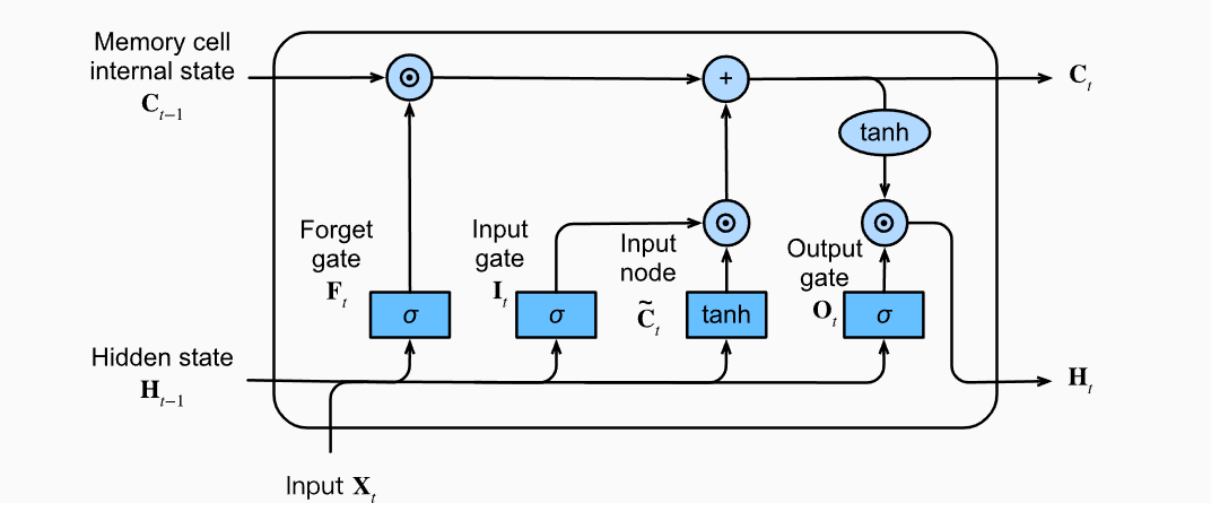
Evo pregleda kako LSTM funkcionira:

- Memorijske ćelije: Temeljna ideja LSTM-a je prisutnost memorijskih ćelija. One su odgovorne za pohranu i pristup informacijama tijekom dugih vremenskih razdoblja. Memorijske ćelije održavaju konstantnu vrijednost tijekom vremena. Mogu naučiti, pamtiti ili zaboraviti informacije na temelju ulaznih podataka.
- Vrata: LSTM mreže sadrže tri vrste vrata koja reguliraju protok informacija: ulazna vrata, vrata za zaboravljanje i izlazna vrata.
- Ulazna vrata: Ova vrata kontroliraju protok novih informacija u memorijsku ćeliju. Odlučuju koje vrijednosti iz ulaza treba pohraniti u ćeliju.
- Vrata za zaboravljanje: Vrata za zaboravljanje određuju koje informacije u memorijskoj ćeliji treba odbaciti ili zaboraviti. Selektivno brišu nevažne informacije iz ćelije.
- Izlazna vrata: Izlazna vrata kontroliraju protok informacija iz memorijske ćelije prema izlazu LSTM jedinice. Odlučuju koje dijelove stanja ćelije treba izvesti pri svakom vremenskom koraku.

Matematičke operacije: U svakom vremenskom koraku, LSTM jedinica izvodi nekoliko matematičkih operacija kako bi ažurirala svoje unutarnje stanje:

- Operacija vrata za zaboravljanje: Određuje koje informacije iz prethodnog stanja ćelije treba odbaciti.

$$f_t = \sigma\big(W_f \cdot [h_{t-1}, x_t] + b_f\big) \tag{4.1}$$



Slika 4: *Arhitektura LSTM Neuralne mreže (Slika preuzeta sa [14]).*

Operacija ulaznih vrata: Izračunava nove informacije koje treba dodati stanju ćelije na temelju trenutnog ulaza i prethodnog stanja ćelije. $\widetilde{C_t}$ je novi kandidat za zamijeniti memorijsku ćeliju C_t.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4.2}$$

$$\widetilde{C_t} = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{4.3}$$

Ažuriranje stanja ćelije: Ažurira stanje ćelije kombiniranjem informacija iz vrata za zaboravljanje i ulaznih vrata.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{4.4}$$

Operacija izlaznih vrata: Izračunava izlaz LSTM jedinice na temelju ažuriranog stanja ćelije i određuje koje dijelove stanja ćelije treba izvesti.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{4.5}$$

$$h_t = o_t * tanh(C_t) \tag{4.6}$$

Sve operacije od (4.1) do (4.6) se mogu vidjeti na Slika 4**.** Simboli su označeni s istim slovima.

Trening: Tijekom treninga, parametri LSTM mreže, uključujući težine i pristranosti vrata, optimiziraju se koristeći algoritme poput gradijentskog spuštanja te propagacije unatrag kroz vrijeme (eng. *Backpropagation through time*) skraćeno BPTT. Mreža uči prilagoditi ove parametre kako bi minimizirala pogrešku između predviđenog izlaza i stvarne vrijednosti.

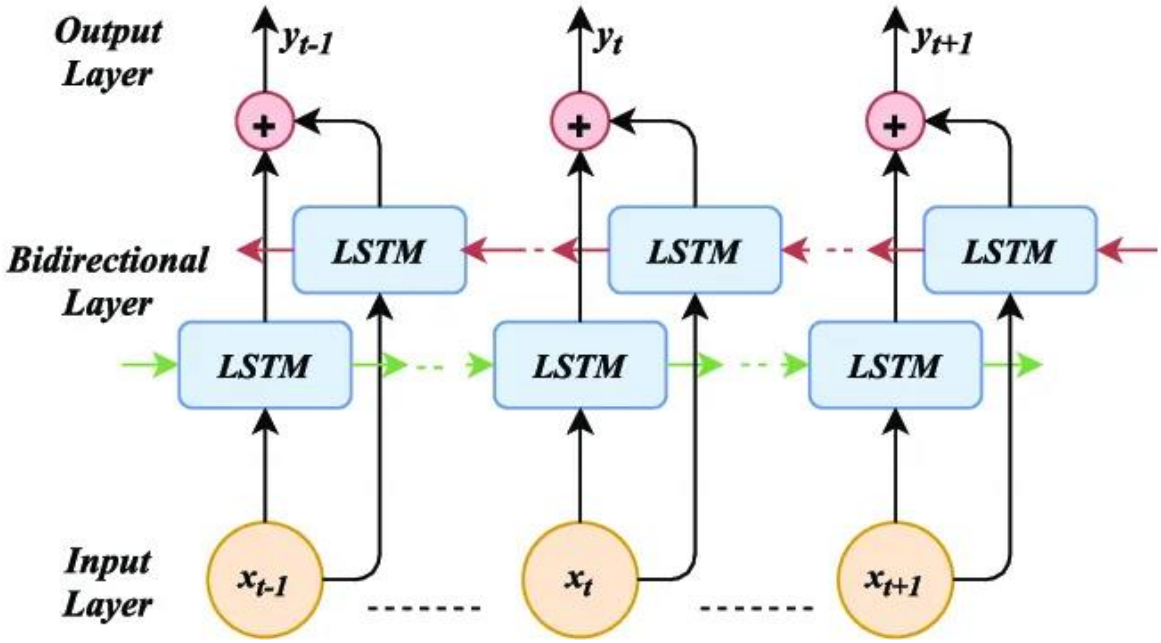
4.1 Bidirekcionalni Long Short-Term Memory (BiLSTM)

(BiLSTM) je vrsta ponavljajuće neuronske mreže (RNN) koja se koristi za obradu sekvencijalnih podataka [15]. Ključna karakteristika BiLSTM mreža je njihova sposobnost da uče iz konteksta u oba smjera – naprijed i unatrag – što omogućava bolju obradu informacija iz cijele sekvence. Možemo vidjeti prikaz na Slika 5 gdje imamo izlazni sloj (eng. *Output Layer*), bidirekcionalni sloj (eng. *Bidirectional Layer*) i ulazni sloj (eng. *Input Layer*).

Bidirekcionalna LSTM mreža sastoji se od dva LSTM sloja:

- **Forward LSTM:** Obrada sekvence od početka do kraja.
- **Backward LSTM:** Obrada sekvence od kraja prema početku.

Ova dva sloja neovisno obrađuju sekvencijalne podatke u oba smjera. Na taj način, BiLSTM može koristiti informacije iz cijele sekvence, što je posebno korisno kada je kontekst važan za predikciju.



Slika 5: *Shema bidirekcionalne lstm mreže (Slika preuzeta sa[16]).*

Unos podataka: Ulazni podaci (sekvenca) se šalju u mrežu.

Kombinacija izlaza: Izlazi iz oba LSTM sloja se kombiniraju. Ovo se može učiniti jednostavnim spajanjem (eng. *concatenation*) ili prosjekom (eng. *averaging*) izlaza, ovisno o implementaciji.

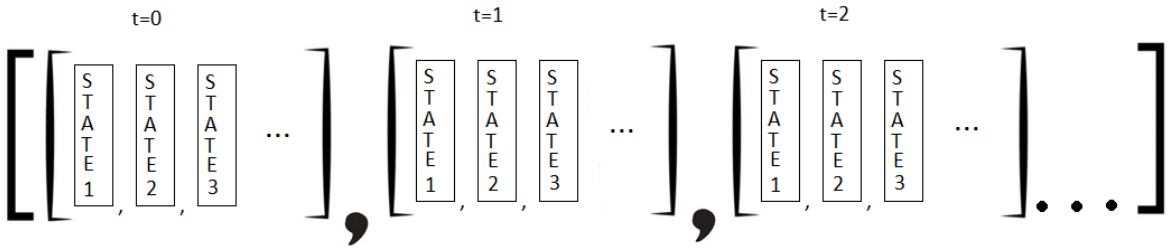
Daljnja obrada: Kombinirani izlaz može se poslati u druge slojeve, kao što su potpuno povezani (eng. *fully connected*) slojevi, za konačnu klasifikaciju ili predikciju.

- **Učenje konteksta u oba smjera:** BiLSTM može koristiti informacije iz cijele sekvence, što je korisno za zadatke gdje je kontekst iz budućnosti ili prošlosti relevantan.
- **Poboljšana točnost:** Korištenjem oba smjera, BiLSTM često postiže bolje rezultate u usporedbi s jednostavnim LSTM-ovima.

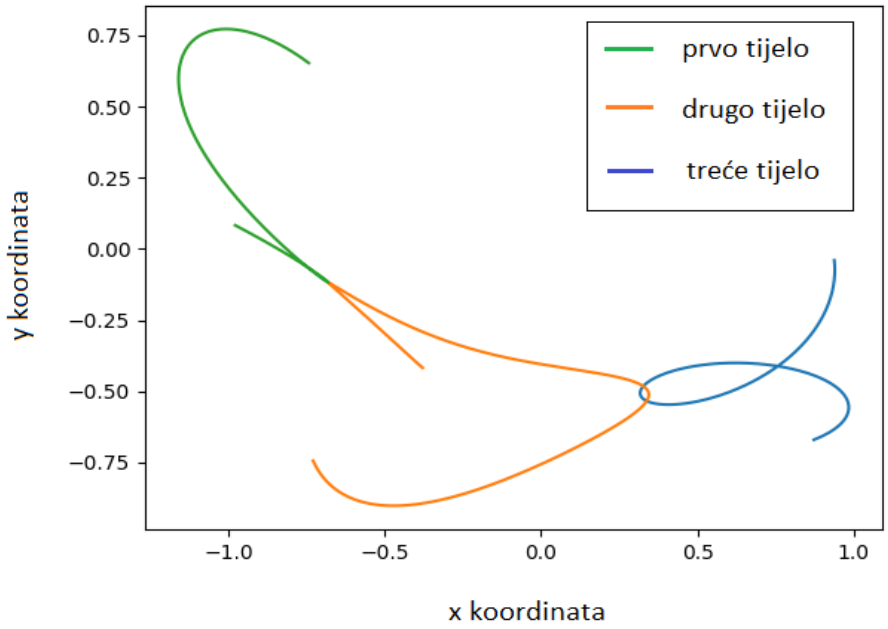
5 Implementacija neuralne mreže na problem triju tijela

5.1 Generiranje simulacija

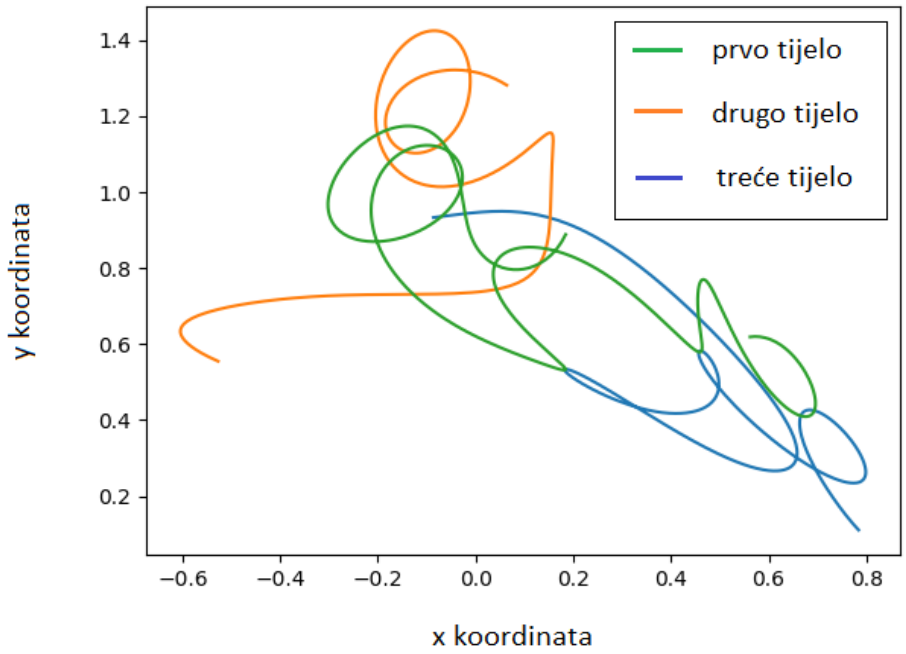
Ideja ovog rada je pokušati analizirati problem triju tijela koristeći neuralne mreže. Bilo je potrebno mnogo simulacija kako bi se prikupio dovoljan broj podataka za treniranje i učenje neuronske mreže (NM). Kod je napisan u Pythonu i simulira kretanje triju tijela u dvije dimenzije radi jednostavnosti. Implementirana je adaptivna Runge-Kutta-Fehlberg (RKF45) metoda numeričke integracije kako bi se svaka simulacija mogla izvršavati kroz duže vremenske intervale. Adaptivna metoda je nužna zbog složenosti simulacija. Veličina koraka u jednadžbama (2.20), (2.21) i (2.22) mora se prilagođavati u svakom novom vremenskom koraku. Ova metoda se pokazala izvanrednom za simulacije. Zbog ograničenja računalnih resursa, nije se mogao prikupiti dovoljan broj simulacija u sekvencama. Da bi se moglo generirati velik broj simulacija u što kraćem vremenu, implementirana je vektorizacija. U jednom procesu se generira 10 simulacija koristeći minimalni korak za Runge-Kutta metodu, što minimizira sve korake tih simulacija. Također je implementirano višejezgreno procesiranje i uključeno 16 procesa istodobno. Implementacija vektorizacije može se vidjeti na Slika 6, gdje su state vektori 'numpy arrays', a svaki sadrži zasebnu simulaciju s 15 parametara. Svaku nezavisnu simulaciju izvršava jedan proces, dok je cijela matrica (Numpy tenzor) skup nezavisnih vektora koji odgovaraju različitim simulacijama. Problem koji se pojavio nakon ubrzavanja generiranja podataka bio je manjak memorije za njihovu pohranu. Svi su podatke spremani u .csv formatu (comma-separated values), što je omogućavalo lakšu manipulaciju i vizualizaciju. Međutim, već na otprilike 800.000 simulacija, veličina sveukupnih podataka narasla je preko 500 GB, pa se moralo komprimirati podatke metodom 'pickle'.



Slika 6: *Skica vektorizacije, gdje se prikazuje glavna lista i u njoj 10 lista, gdje svaka sadrži po nekoliko state vektora. Slika izrađena u paint.*



Slika 7: *Primjer jedne simulacije izrađene u Pythonu.*

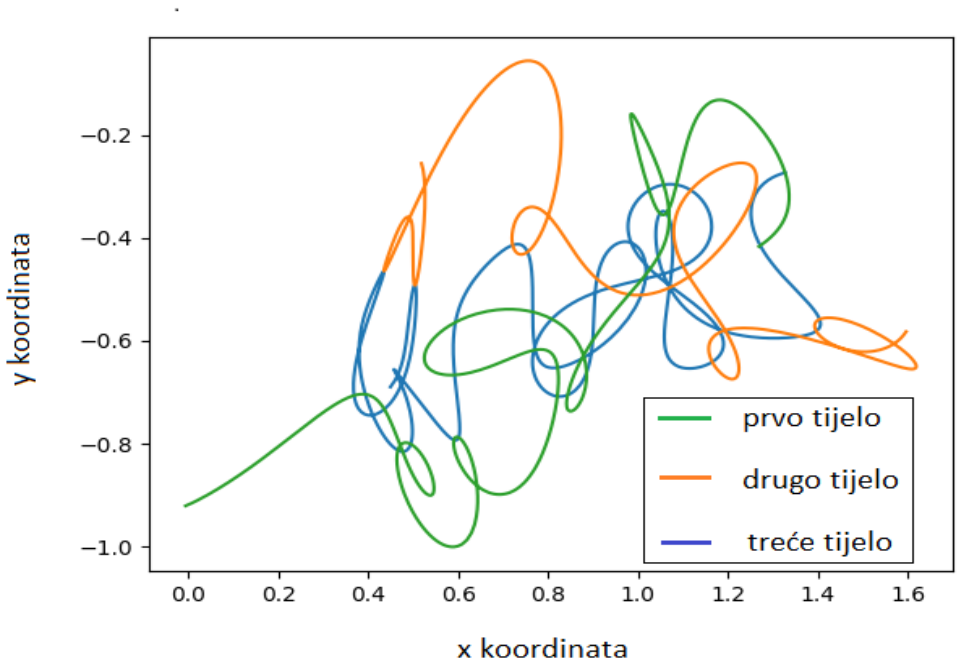


Slika 8: *Primjer jedne simulacije sa složenijim gibanjem izrađene u Pythonu.*

Spremljeni parametri svake simulacije uključuju pozicije, brzine i međusobne udaljenosti triju tijela tijekom cijele simulacije, što ukupno čini 15 parametara. Parametri su prikazani u tablicama: **Tablica 1.** Prikaz parametara kod jedne simulacije, ovdje su prikazani x i y

koordinate kod triju tijela., **Tablica 2.** Prikaz parametara kod jedne simulacije, ovdje su prikazane brzine u x i y smjeru kod triju tijela. i **Tablica 3.** Prikaz parametara kod jedne simulacije, ovdje su prikazane međusobne udaljenosti kod triju tijela.. U prvom retku svake tablice nalaze se početni uvjeti, dok su u preostala dva retka prikazani rezultati za sljedeća dva vremenska koraka u simulaciji.

U konačnici nasumično se odabire 100 000 simulacija, koje se dodatno transformira za ulaz i etikete (eng. *Labels*), potrebni za treniranje NM. Podaci su organizirani tako da svaki korak pune sekvence jedne simulacije služi kao ulaz, a sljedeći korak te iste sekvence, služi kao etiketa. NM pokušava predvidjeti sljedeći korak, ovom metodom u konačnici se dobije oko 73 milijuna podataka za učenje.



Slika 9: *Primjer jedne simulacije sa još složenijim gibanjem izrađene u Pythonu.*

Svi početni položaji su generirani nasumično u rasponu (-1 , 1), početne brzine u rasponu od (-0.5 ,0.5). Brojevi su zapisano kao float32, znači ima 8 decimala, sasvim dovoljno brojeva da svi početni uvjeti budu jedinstveni. Kasnije je smanjen raspon koordinatnog sustava na razmak od 1.5, jer je dosta simulacija bilo takvih da bi se tijela sudarila i odletjela u pravcu i onda bi u nedogled išle samo linije. Za neuralnu mrežu to bi značilo da bi pretpostavljala dosta linearnih gibanja. Izgubila bi bitnu komponentu njihove međusobne interakcije kada su blizu i kada gravitacija najjače djeluje. Primjeri simulacija se mogu vidjeti na slikama 7, 8 i 9.

Simulacije su spremljene u pickle datotekama koje su organizirane u tenzore spremne za grupiranje. Nakon toga su prebačene na grafičku karticu da ih NM može obrađivati.

5.2 Hiperparametri

Hiperparametri su parametri u modelima strojnog učenja koji se ne uče iz podataka tijekom treniranja, već se postavljaju prije početka treninga [17]. Ovi parametri upravljaju procesom obuke i arhitekturom modela, utječući na performanse i učinkovitost algoritma učenja. Odabir pravog skupa hiperparametara je ključan i često zahtijeva eksperimentiranje dok se ne uštimaju najpogodnije brojke. U ovom radu se se hiperparametri dosta isprobavali dok se nisu dobili najbolji uvjeti za učenje i treniranje. Ispod su navedeni i objašnjeni hiperparametri koji su bili korišteni u ovom radu.

5.2.1 Bidirekcionalni LSTM

Bidirekcionalni LSTM obrađuje sekvencijalne podatke u oba smjera, i naprijed (od početka do kraja) i unatrag (od kraja do početka). Ilustracija se može vidjeti na Slika 5. Ovo omogućuje modelu da ima potpuni uvid u cijelu sekvencu prije donošenja odluka ili predviđanja. U problemu tri tijela, položaji i brzine tijela u nekom trenutku u budućnosti mogu ovisiti o njihovim prethodnim stanjima. Bidirekcionalni LSTM može biti korišten za učenje ove zavisnosti iz podataka o prethodnim položajima i brzinama tijela. Bidirekcionalni LSTM, sa svojom sposobnošću da se nosi s kompleksnim sekvencama i nelinearnim odnosima može pomoći predviđanju kaotičnih gibanja triju tijela.

Skrivene ćelije izražene u jednadži (4.6) su postavljene na 1024. Što je njih više to mreža ima veći kapacitet za pamćenje i može predvidjeti kompliciranije putanje. Ne smijemo ni pretjerati sa veličinom skrivenih ćelija jer to može dovesti do pretreniranja (eng. *Overfitting*), onda mreža slabije uči nove putanje.

5.2.2 Broj Epoha

Broj Epoha je stavljen na 100, ona označava jedan prolazak kroz cijeli skup podataka za treniranje. Tijekom jedne epohe, model prolazi kroz svaki podatak iz skupa za treniranje barem jednom. S obzirom na to da model kroz epohu prolazi kroz cijeli skup podataka, očekuje se da će se njegove performanse poboljšati kako stječe bolju sposobnost učenja iz podataka. Međutim, previše epoha može dovesti do pretreniranja.

5.2.3 Batch

Skup (eng. *Batch*) označava broj uzoraka iz skupa podataka koji se koriste za ažuriranje modela u jednom prolazu kroz treniranje. Umjesto da se cijeli skup podataka koristi odjednom za ažuriranje modela, podaci se često dijele na manje skupove kako bi se postigla bolja učinkovitost treniranja i optimizacije. Nakon svake epizode skupa, model ažurira svoje parametre (težine i pomake) kako bi smanjio grešku ili funkciju gubitka.

Rad s cijelim skupom podataka odjednom može zahtijevati previše memorije, osobito kod velikih skupova podataka. Korištenjem skupova, samo jedan dio podataka treba biti u memoriji u svakom trenutku, čime se smanjuje ukupna memorijska potrošnja.

Ažuriranje težina nakon svakog uzorka može biti sporo i neučinkovito. Korištenjem skupova, model se može ažurirati rjeđe, ali s više informacija iz skupa podataka odjednom, što može rezultirati stabilnijim i učinkovitijim procesom učenja. Za ovaj rad smo postavili skup na 2000.

5.2.4 L2 regularizacija

L2 regularizacija, također poznata kao **Ridge regularizacija** ili **Tikhonov regularizacija**, je tehnika koja se koristi u strojnome učenju za smanjenje kompleksnosti modela i sprječavanje pretreniranja. Ova metoda postiže regularizaciju dodavanjem dodatnog termina u funkciju gubitka koji kažnjava velike vrijednosti parametara modela. Matematički, ukupna funkcija gubitka s L2 regularizacijom može se izraziti kao:

$$L_{total} = L_{original} + \lambda \sum_{i=1}^{n} w_i^2 \tag{5.1}$$

Regularizacijski termin $\lambda \sum_{i=1}^{n} w_i^2$ u jednadžbi (5.1) kažnjava velike težine. Time se model prisiljava da drži težine malima, što smanjuje kapacitet modela da se previše prilagodi na podatke za treniranje, odnosno da "zapamti" podatke, umjesto da nauči generalizirati. U radu je postavljena na 0.00001.

5.2.5 Dropout

Isključivanje neurona (eng. *Dropout*) je tehnika regularizacije koja se koristi za sprječavanje prekomjernog učenja (overfittinga) u neuronskim mrežama. Tijekom treniranja, dropout nasumično postavlja dio ulaznih jedinica (neurona) na nulu pri svakom ažuriranju. Ovo

sprječava neurone da se previše zajednički prilagođavaju, potičući mrežu da nauči robusnije i općenitije značajke.

Stopa dropout-a, obično označena kao vrijednost između 0 i 1, predstavlja vjerojatnost postavljanja određene jedinice na nulu tijekom treniranja. Na primjer, stopa dropout-a od 0,5 znači da svaki neuron ima 50% šanse biti isključen tijekom svake iteracije treniranja.

Nasumičnim isključivanjem jedinica, dropout prisiljava mrežu da se ne oslanja na specifične neurone, čime se sprječava prekomjerno učenje, gdje model dobro radi na trening setu, ali loše na neviđenim podacima. U radu je dropout stavljen na 0.15.

5.2.6 Stopa učenja

Stopa učenja je ključni hiperparametar u algoritmima strojnog učenja, posebno onima koji koriste optimizacijske metode poput gradijentnog spusta. Određuje koliko velik korak model poduzima pri ažuriranju svojih parametara (poput težina u neuronskoj mreži) u svakom koraku treniranja na temelju izračunate pogreške.

Tijekom treniranja modela, cilj je minimizirati funkciju gubitka, koja mjeri koliko su predikcije modela udaljene od stvarnih vrijednosti. Algoritmi poput gradijentnog spusta koriste derivaciju funkcije gubitka kako bi izračunali smjer u kojem treba ažurirati parametre kako bi se smanjila pogreška.

Stopa učenja α određuje veličinu tih ažuriranja. Ako je $\nabla L(\theta)$ gradijent funkcije gubitka u odnosu na parametre θ, tada je ažuriranje parametara opisano kao:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t) \tag{5.2}$$

U jednadžbi (5.2), θ_{t+1} su ažurirani parametri za sljedeću iteraciju, θ_t su trenutni parametri u iteraciji t. Visoka stopa učenja znači da će model brže mijenjati parametre, dok niska stopa učenja znači da će promjene biti manje i opreznije. Stopa učenja se za svaki parametar ažurira preko ADAM-a, spomenutog u poglavlju 3.2. Adam prilagođava stopu učenja za svaki parametar pojedinačno. Stopa učenja se skalira prema recipročnoj vrijednosti kvadratnog korijena procjene drugog momenta $\sqrt{v_t}$ u jednadžbi (3.14). To znači da će parametri s većim gradijentima imati manje stope učenja, dok će parametri s manjim gradijentima imati veće stope učenja. U radu smo stopu učenja postavili na 0.0001.

5.2.7 Gradient clipping

Rezanje gradijenata (eng. *Gradient clipping*) je tehnika koja se koristi kako bi se spriječilo da gradijenti postanu pretjerano veliki tijekom treniranja. Veliki gradijenti mogu uzrokovati nestabilnost u treniranju, što može dovesti do eksplodirajućih gradijenata ili uzrokovati divergenciju optimizacijskog algoritma. Rezanje gradijenata pomaže ublažiti ove probleme postavljanjem praga na veličinu gradijenata. Eksplozija gradijenata može uzrokovati da parametri modela postanu jako veliki, što dovodi do nestabilnosti i često onemogućava konvergenciju modela.

Tijekom unazadne propagacije, gradijenti se izračunavaju za svaki parametar u modelu. Rezanje gradijenta radi tako što ograničava veličinu gradijenata tokom ažuriranja težina. Postoje različite metode za implementaciju gradient clippinga:

Clipping by Norm: U ovoj metodi se normom (najčešće L2 normom) svih gradijenata mjeri njihova veličina. Ako je norma veća od određenog praga, gradijenti se skaliraju tako da njihova norma postane jednaka tom pragu.

Clipping by Value: Ovdje se svi gradijenti koji prelaze određeni prag zamjenjuju tim pragom. Na primjer, ako je prag postavljen na 1.0, svi gradijenti veći od 1.0 će biti postavljeni na 1.0. U radu smo se poslužili sa prvom metodom i maksimalni prag je postavljen na vrijednost 8.0.

5.2.8 Validacijsko strpljenje

Validacijsko strpljenje odnosi se na broj epoha koje model može trenirati bez poboljšanja na skupu podataka za validaciju prije nego što se treniranje zaustavi. Ova tehnika se koristi kako bi se spriječila pretreniranost modela, što znači da model počinje učiti specifične uzorke u podacima za obuku koji ne generaliziraju dobro na nove, neviđene podatke.

Praćenje performansi: Tijekom treniranja modela, metrika performansi (npr. točnost, gubitak) se prati na skupu podataka za validaciju nakon svake epohe.

Provjera poboljšanja: Ako se performanse modela na validacijskom skupu ne poboljšavaju nakon određeni broj epoha (definiran kao "strpljenje"), treniranje se zaustavlja.

Kriterij zaustavljanja: Ako je zadano strpljenje, recimo, 5, i model ne pokaže poboljšanje na validacijskom skupu kroz 5 uzastopnih epoha, treniranje se prekida.

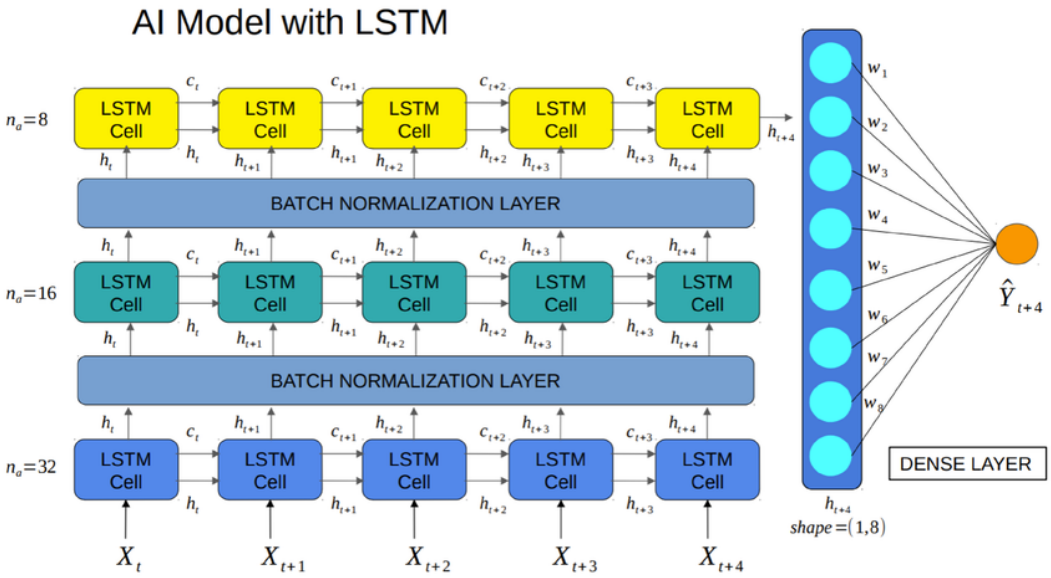
U radu smo se koristili sa validacijskim strpljenjem od 15.

5.2.9 Broj LSTM slojeva

Broj LSTM slojeva u modelu odnosi se na broj LSTM jedinica koje su poredane vertikalno, jedna iznad druge, u modelu. Prikaz slojeva je ilustriran na Slika 10, gdje se *'DENSE LAYER'* znači gusti sloj, '*shape'* označava oblik matrice, *'LSTM Cell'* znači LSTM ćelija, *'Batch Normalization Layer'* je normalizacijski sloj za '*batch*'. Ovi slojevi mogu raditi serijski, gdje izlaz jednog sloja postaje ulaz za sljedeći, ili paralelno, ovisno o arhitekturi. Veći broj slojeva, koristi se za složenije zadatke koji zahtijevaju dubinsko razumijevanje sekvencijalnih obrazaca.

- **Prednosti**: Poboljšana sposobnost modeliranja složenih obrazaca i dugoročnih ovisnosti. Svaki sloj može naučiti različite razine apstrakcije.
- **Ograničenja**: Veća složenost modela, više parametara za treniranje, dulje vrijeme treniranja, veća potreba za računalnim resursima, mogućnost pretreniravanja.

Zbog balansa između složenosti problema i ograničenih resursa, uzeli smo 3 sloja.



Slika 10: *Prikaz LSTM mreže sa 3 sloja (Slika preuzeta sa [18])*

5.2.10 Learning rate decay

Smanjivanje stope učenja (eng. *Learning rate decay*) je tehnika koja se koristi u treniranju neuronskih mreža za postupno smanjenje stope učenja tokom vremena. Stopa učenja je ključni hiperparametar u optimizaciji koji određuje veličinu koraka koje model poduzima prilikom ažuriranja težina na osnovu gradijenata gubitka. Korištenje fiksne stope učenja tokom cijelog

treniranja može biti neefikasno. Visoka stopa učenja može ubrzati treniranje na početku, ali može spriječiti konvergenciju na optimalno rješenje u kasnijim fazama, dok preniska stopa učenja može učiniti treniranje sporim ili zaglaviti model u lokalnim minimumima. Smanjivanje stope učenja omogućava:

1. **Brzi početak učenja**: Veća početna stopa učenja omogućava brzu prilagodbu težina na početku treniranja.
2. **Fina podešavanja kasnije**: Postepeno smanjenje stope učenja omogućava modelu da se preciznije prilagodi optimalnom rješenju kako treniranje napreduje. Postoji nekoliko metoda za implementaciju smanjivanja stope učenja:

1. Step Decay (Korak smanjenja)

Stopa učenja se smanjuje za fiksni faktor nakon određenog broja epoha.

$$\eta = \eta_0 \cdot \text{factor}^{\left(\frac{\text{epoch}}{\text{drop very}}\right)} \tag{5.3}$$

Gdje je η trenutna stopa učenja, η_0 početna stopa učenja, *factor* faktor smanjenja, a *drop every* broj epoha nakon kojih se smanjenje događa.

2. Exponential Decay (Eksponencijalno smanjenje)

Stopa učenja se smanjuje eksponencijalno nakon svake epohe.

$$\eta = \eta_0 \cdot e^{-\lambda \cdot \text{epoch}} \tag{5.4}$$

Gdje je λ stopa smanjenja.

3. 1/t Decay

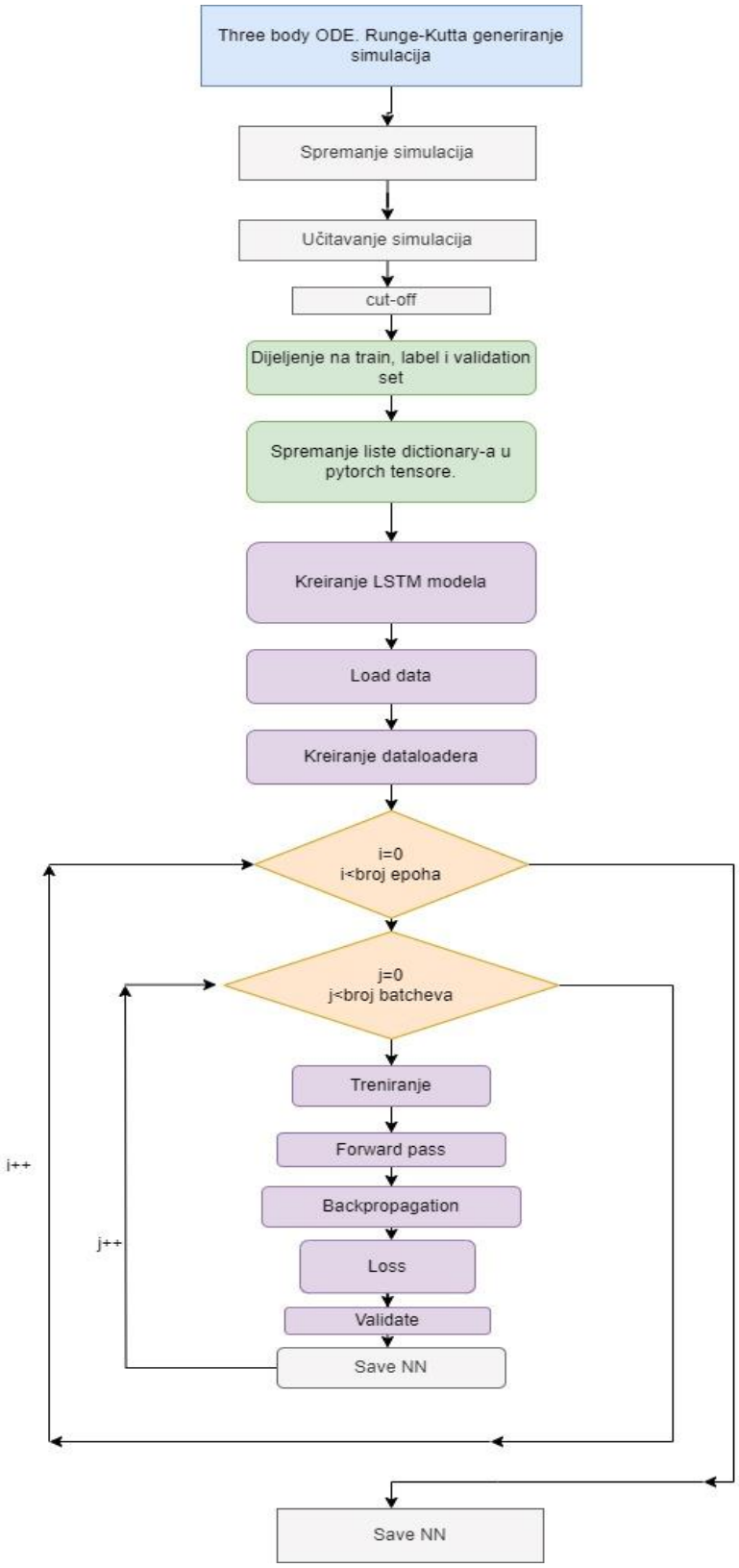
Stopa učenja se smanjuje obrnuto proporcionalno s brojem epoha.

$$\eta = \frac{\eta_0}{1 + \lambda \cdot \text{epoch}} \tag{5.5}$$

5. **Cosine Annealing**

Stopa učenja se smanjuje po kosinusnoj funkciji, što omogućava oscilacije u stopi učenja. Koristi se za bolje istraživanje oblika funkcije gubitaka.

5.3 Algoritam simulacija i NM



Slika 11: *Dijagram koda izrađen u draw.io.*

Predloženi algoritam na Slika 11 počinje sa inicijalizacijom parametara za generiranje simulacija pomoću Runge-Kutta metode. Trebalo je ubrzati proces korištenjem višejezgrenog procesiranja. Simulacije su spremane u csv formatu, komprimirane te su takve spremne za učitavanje.

Nakon toga se za neuralnu mrežu napravio rez (eng. *cut-off*) da ne uči nepotrebne putanje koje bi joj otežale učenje, jer bi učile linearne putanje koje nisu potrebne. Podaci koji su učitani pomoću učitavača podataka (eng. *data loader*) su podijeljeni na trening podatke (eng. *train data*), test podatke (eng. *test data*) i validacijski skup (eng. *validation set*). Trening podaci služe za treniranje modela, ažurirajući parametre. Validacijski skup služi za podešavanje i odabir modela, pomažući u sprječavanju pretreniranja. Test podaci služe za posljednju procjenu, osiguravajući da se model generalizira na nove podatke. Izrađeni su tenzori koji su ulazili u dubinu, jedni su sadržavali ulaze, a drugi etikete. Mogu se vidjeti prikazi tenzora na Slika 12. Pomoću tih tenzora NM je mogla vršiti treniranje. U Pythonu tenzor je osnovna matematička struktura koja se koristi za prikaz podataka. Tenzori su generalizacija skalarnih vrijednosti, vektora i matrica. U najširem smislu, dubina tenzora označava broj dimenzija koje tenzor ima. Namješteno je za NM da čita jedan ulaz, pa pokušava predvidjeti sljedeći korak, onda bi to bio ulaz za sljedeći korak itd.

LSTM model je definiran s mogućnošću više slojeva i mogućnošću bidirekcionalnosti. Model se sastoji od LSTM sloja, sloja normalizacije, proširenog potpuno povezanog sloja (eng. *FCC*), dropout sloja i konačnog izlaznog sloja. Xavier inicializacija se koristi za inicijalizaciju težina, dok se pristranost postavlja na nulu. Uspostavljena je funkcija koja izračunava srednje vrijednosti i standardne devijacije za svaku od značajki u trening setu. One se kasnije koriste za normalizaciju podataka. Ovi izračunati podaci se spremaju u pickle datoteku kako bi se kasnije koristili za normalizaciju u skupu podataka.

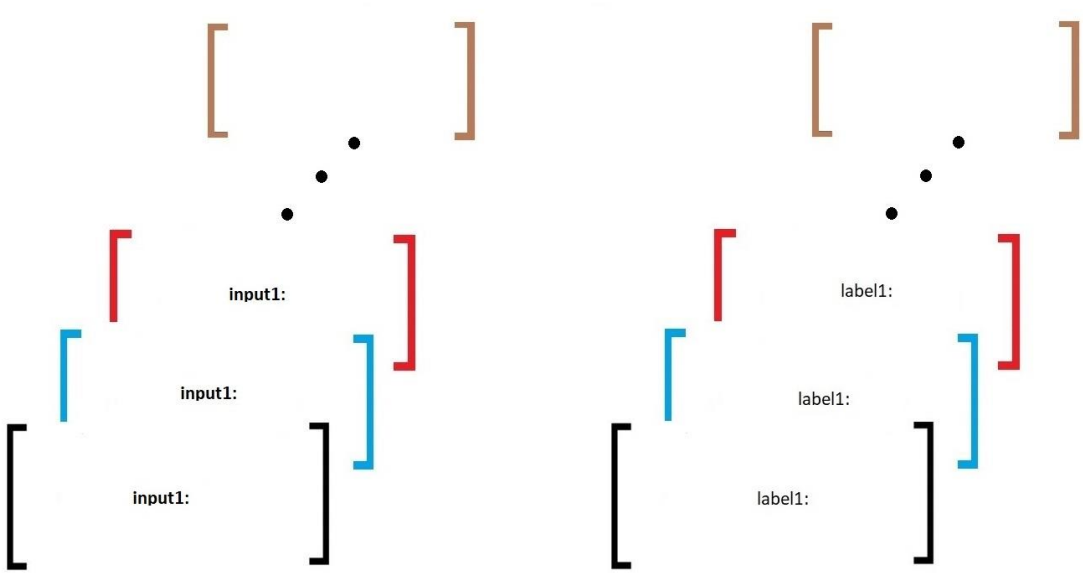
Prilagođavan je učitavač podataka koji učitava podatke iz pickle datoteke i normalizira ih koristeći prethodno izračunate srednje vrijednosti i standardne devijacije. Ovaj učitavač podataka omogućava učitavanje podataka za trening, validaciju i testiranje.

Treniranje mreže u ovoj funkciji se odvija cijeli proces treniranja mreže:

- **Inicijalizacija modela:** Novi model se kreira ili se učitava već prethodno trenirani model.

- **Gubitak (Loss) i Optimizator:** Odabire se gubitak (default je RMSE) i optimizator (default je Adam), zajedno s mogućim raspoređivačima (eng. *Schedulers*) za kontrolu stope učenja (eng. *learning rate decay*).
- **Treniranje:** Za svaku epohu, model prolazi kroz trening skup podataka, izračunava gubitak, propagira gradijente unazad, te optimizator ažurira težine mreže.
- **Validacija:** Nakon svake epohe, model se validira na skupu podataka za validaciju kako bi se pratilo poboljšanje modela.
- **Spremljanje modela:** Nakon svake epohe, trenutni model i njegovi rezultati se spremaju u .pth datoteku.
- **Early Stopping:** Ako validacijski gubitak ne pokazuje poboljšanje nakon određenog broja epoha, treniranje se prekida.

Nakon što je model istreniran, koristi se za testiranje na neviđenim podacima. Model se učitava iz .pth datoteke. Podaci se učitavaju i normaliziraju, te se na kraju generiraju predikcije za test skup. Spremanje rezultata gubitka, ova funkcija sprema rezultate gubitka iz treninga i validacije u csv datoteke za kasniju analizu. RMSELoss klasa je prilagođena funkcija gubitka koja računa korijen srednje kvadratne pogreške (RMSE), što je često korištena mjera za regresijske zadatke.



Slika 12: *Shematski prikaz tenzora, ulijevo koji sadrži ulaze, dok je udesno za etikete, a tenzori idu u dubinu. Slika izrađena u paint.*

x_1	y_1	x_2	y_2	x_3	y_3
0.812	0.552	-0.022	0.672	0.366	-0.136
0.81002	0.557071	-0.01871	0.677751	0.372111	-0.13108
0.808285	0.560787	-0.01602	0.681953	0.376756	-0.12711

Tablica 1. *Prikaz parametara kod jedne simulacije, ovdje su prikazani x i y koordinate kod triju tijela.*

v_{x_1}	v_{y_1}	v_{x_2}	v_{y_2}	v_{x_3}	v_{y_3}
-0.141	0.404	0.245	0.459	0.477	0.37
-0.1692	0.390533	0.26978	0.442128	0.480435	0.400339
-0.1907	0.380229	0.288653	0.429387	0.483007	0.423384

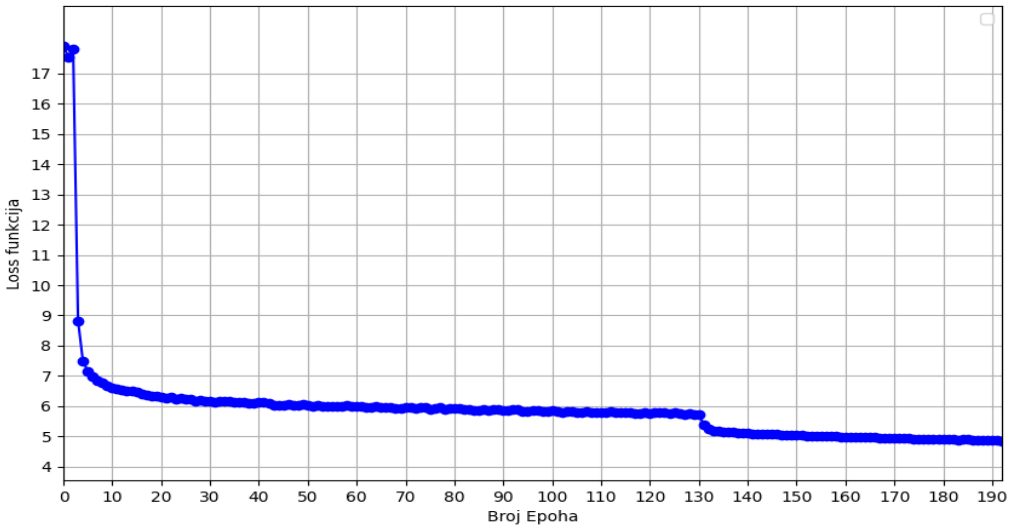
Tablica 2. *Prikaz parametara kod jedne simulacije, ovdje su prikazane brzine u x i y smjeru kod triju tijela.*

r_{12}	r_{13}	r_{23}
0.842588868	0.81991463	0.896330296
0.837475459	0.815672492	0.898308714
0.833165562	0.812049265	0.89936731

Tablica 3. *Prikaz parametara kod jedne simulacije, ovdje su prikazane međusobne udaljenosti kod triju tijela.*

5.4 Rezultati i diskusija

Računalo koje je korišteno imalo je sljedeće specifikacije: CPU - Ryzen 7 3700x s 8 jezgri i 16 niti (eng.*threads*), pojačano (eng. *Overclock*) na 4,5 GHz, 96 GB DDR4 RAM-a na 3200 MHz, te GPU - Nvidia RTX 3080 Ti s 12 GB VRAM-a. Inicijalno je dobiveno oko 73 milijuna sekvencijalnih podataka za NM za treniranje i da uspije minimizirali funkciju gubitka. Pošto je problem dosta kompleksan te što je bilo ograničenje računalnom snagom, dobiveni su rezultati koji nisu bili dovoljno dobri za neuralnu mrežu da može sama izraditi simulaciju od naučenog modela.



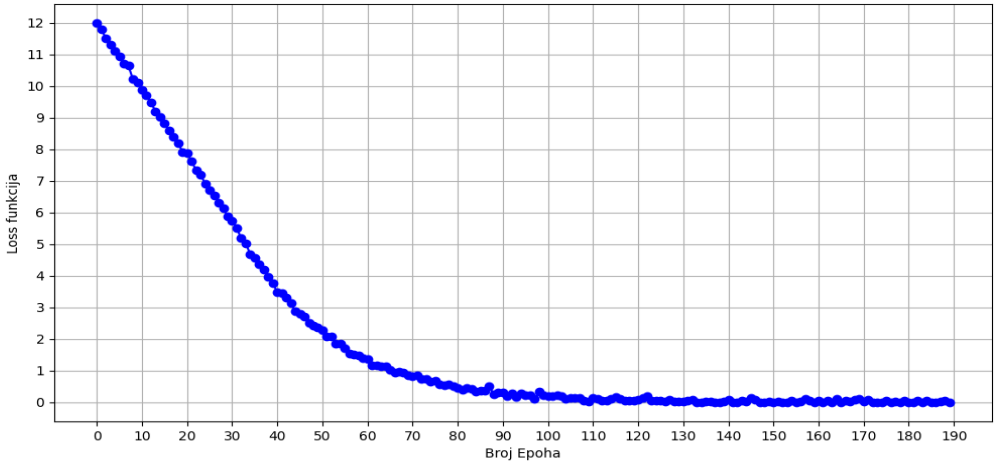
Slika 13: *Prikaz Loss funkcije nakon 190 epoha, koja je padala sa vrijednost 18 i konvergirala prema 4.8. Graf izrađen u Python-u.*

Za ovakvo učenje bi nam trebala puno veća NM, znači veći broj LSTM skrivenih slojeva i veći broj skupova.

Kao što se može vidjeti na Slika 13, funkcija gubitka dobro opada u početku, ali presporo konvergira prema nekakvoj optimalnoj vrijednosti. Pokušalo se sa raznim algoritmima ubrzati proces, ali bilo je bitno otežano i usporeno.

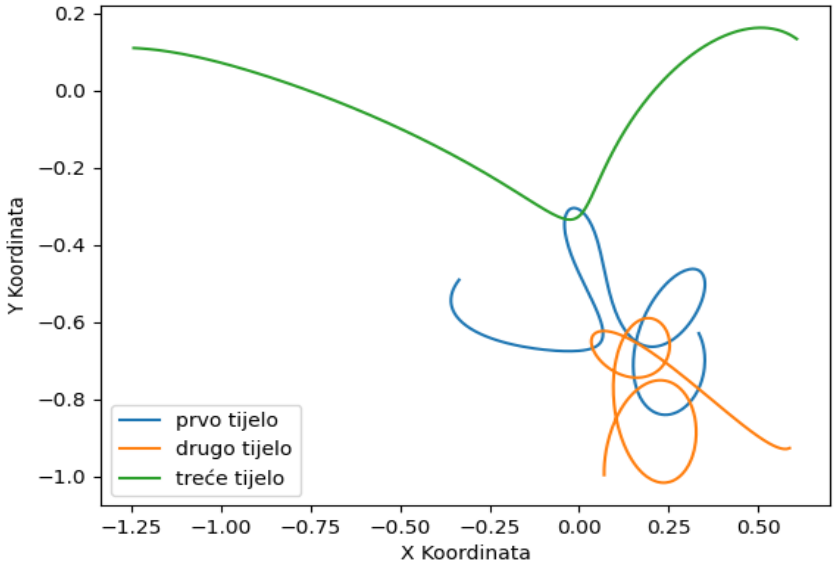
Nakon što je smanjen broj sekvenci, odnosno uzet manji uzorak sekvencijalnih podataka, uspjelo se postići optimalniju funkciju gubitka. Smanjen je prostor početnih uvjeta tako da budu bliži jedni drugima. Izabrane su jako bliske sekvence kod početnih uvjeta naših 15 parametara.

Možemo vidjeti na Slika 14, kako funkcija gubitka puno bolje konvergira prema optimalnijoj vrijednosti.

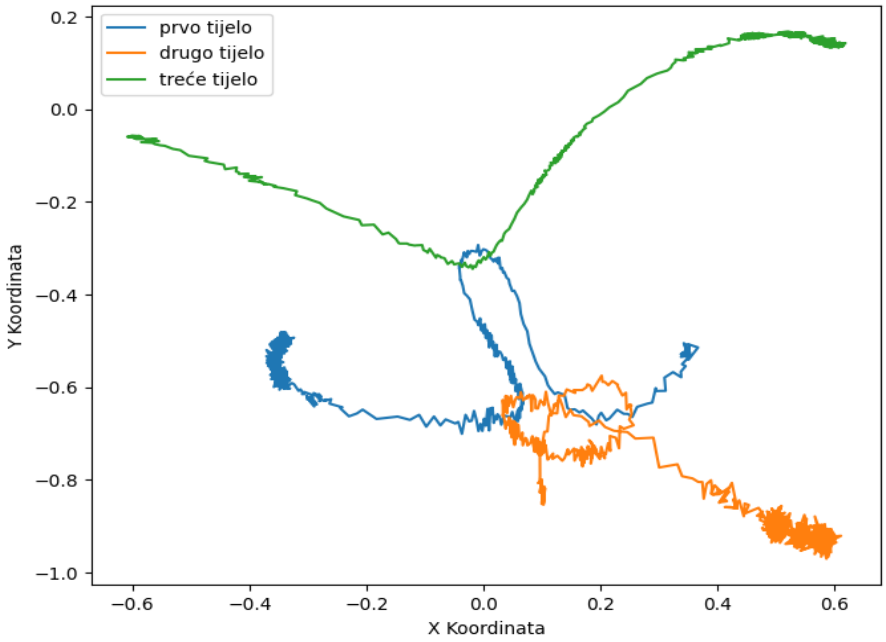


Slika 14: *Prikaz Loss funkcije nakon smanjenog broja sekvenci, gdje Loss funkcija pada s 12 na otprilike 0.0026. Graf izrađen u Python-u.*

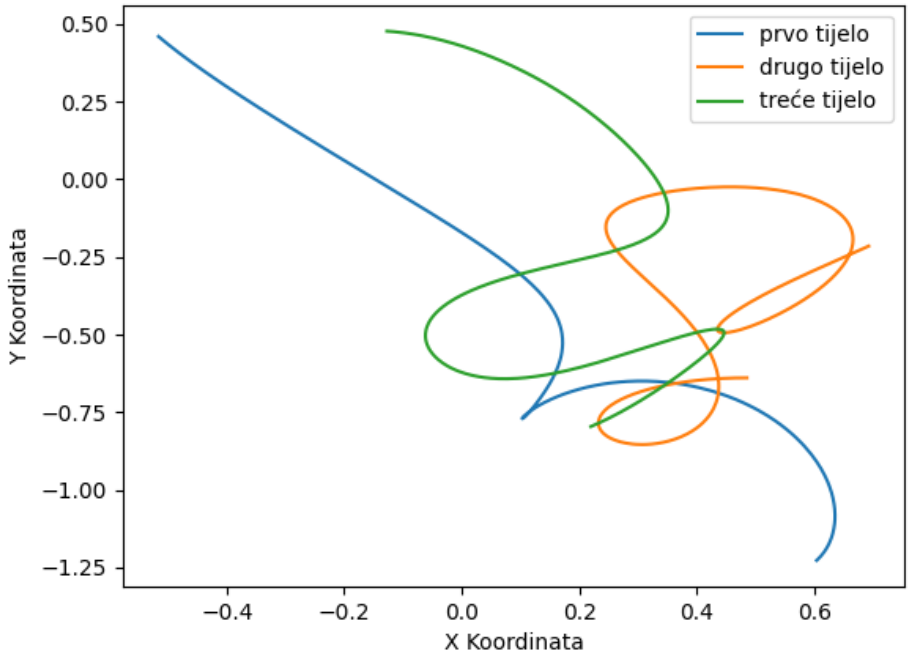
Nakon što je NM uspjela doći do dovoljno niske vrijednosti da bi ona mogla onda sama generirati svoje simulacije koje bi bile približne izvornim. Ispod su prikazane četiri simulacije kako izgledaju pomoću Runge-Kutta metoda. Onda će se usporediti kako ih je generirala sama NM sa istim početnim uvjetima.



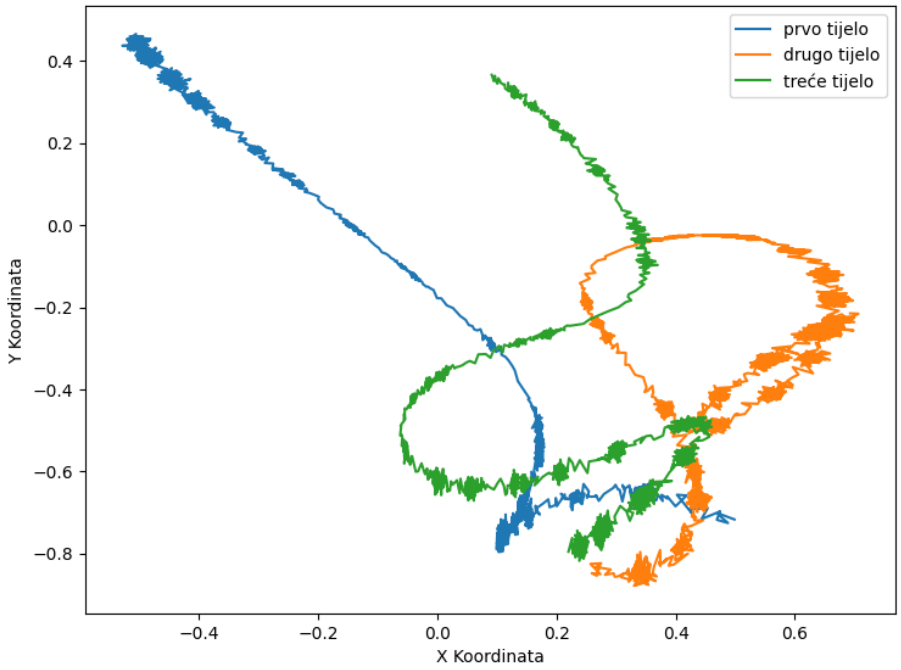
Slika 15: *Prikaz prve Runge-Kutta simulacije. Slika izrađena u Python-u.*



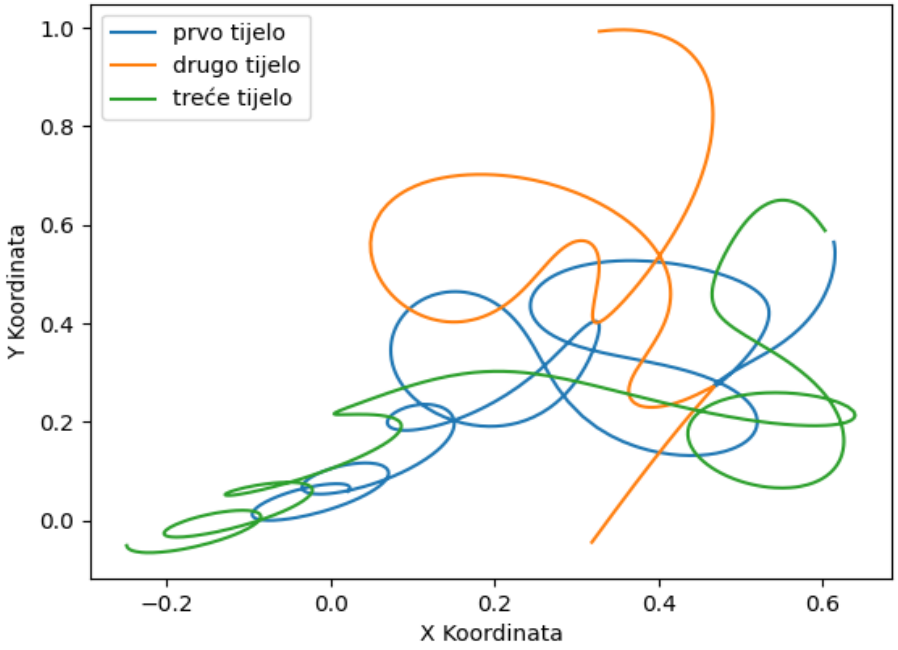
Slika 16: *Prikaz prve simulacije sa početnim uvjetima sa prethodne slike, kako ju je generirala NM. Slika izrađena u Python-u.*



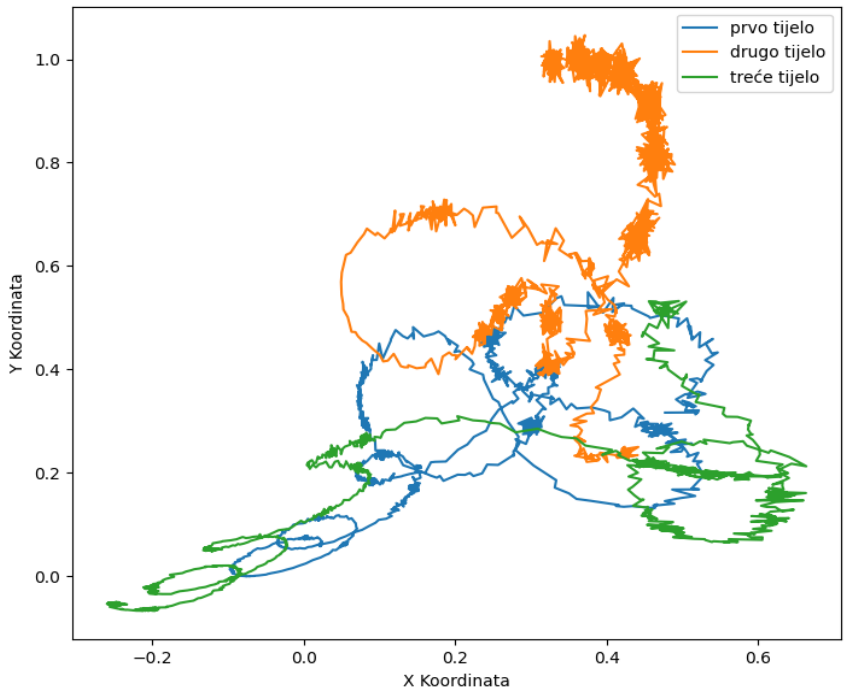
Slika 17: *Prikaz druge Runge-Kutta simulacije. Slika izrađena u Python-u.*



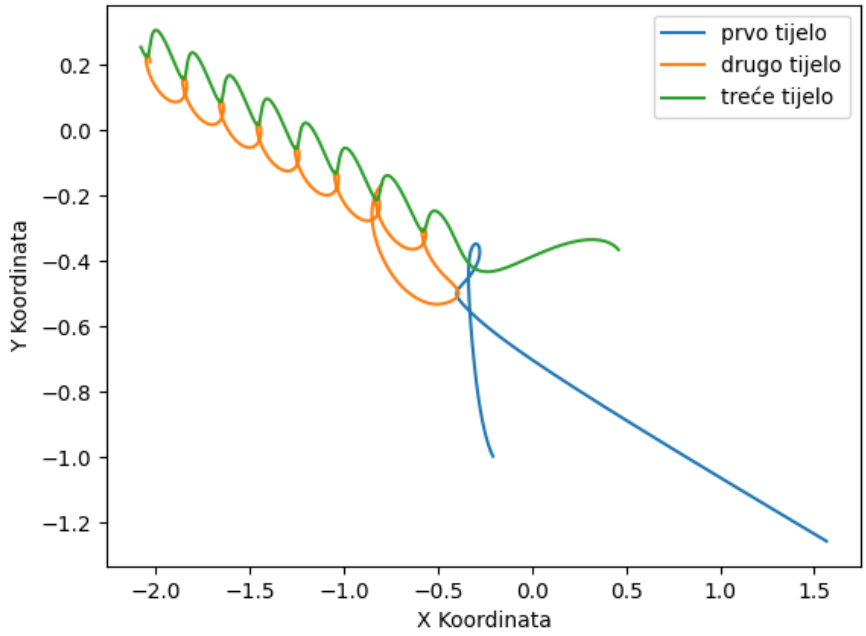
Slika 18: *Prikaz druge simulacije sa početnim uvjetima sa prethodne slike, kako ju je generirala NM. Slika izrađena u Python-u.*



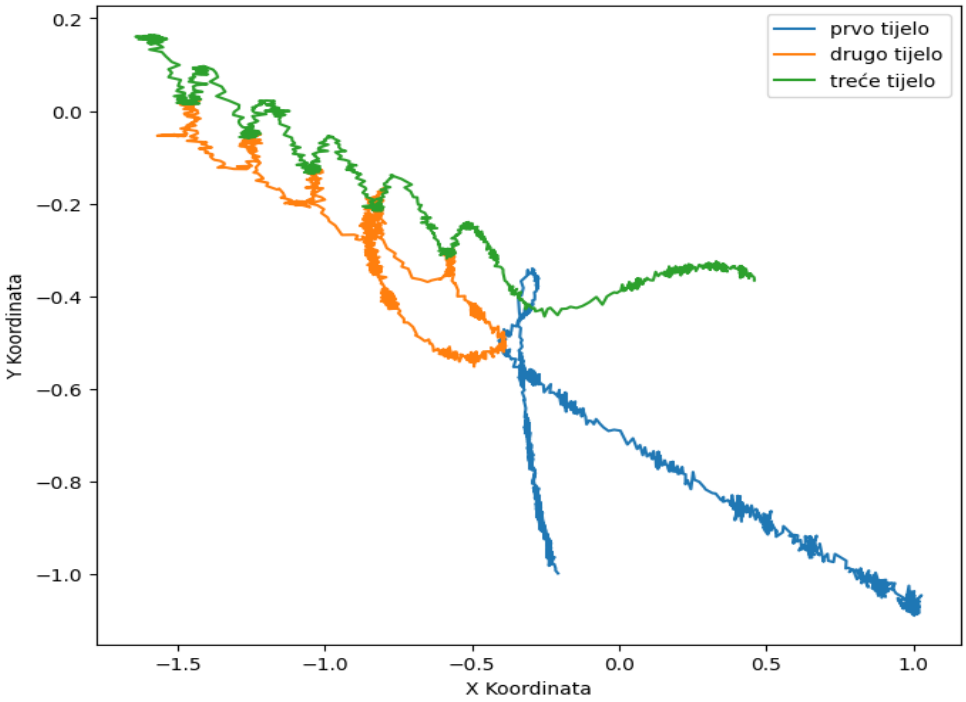
Slika 19: *Prikaz treće Runge-Kutta simulacije. Slika izrađena u Python-u.*



Slika 20: *Prikaz treće simulacije sa početnim uvjetima sa prethodne slike, kako ju je generirala NM. Slika izrađena u Python-u.*



Slika 21: *Prikaz četvrte Runge-Kutta simulacije. Slika izrađena u Python-u.*



Slika 22: *Prikaz četvrte simulacije sa početnim uvjetima sa prethodne slike, kako ju je generirala NM. Slika izrađena u Python-u.*

Na slikama 15, 17, 19 i 21 možemo vidjeti četiri nasumično odabrane simulacije koje su uzete kao primjer. Nakon što je NM uspjela spustiti funkciju gubitka u testiranju na vrijednost otprilike 0.0092, ta funkcija je RMSE u jednadžbi (3.10). RMSE nam govori kolika je greška kada je NM učila predviđati sljedeći korak na račun prošloga koraka. Sveukupna greška od pozicija , udaljenosti i brzina, dakle 15 parametara.

Zbog dobivene vrijednost od 0.0026 za RMSE u treniranju, rezulatati predviđanja NM na slikama 16, 18, 20 i 22 izgledaju kao da imaju buku. Usprkos tome, generalno predviđanja prate trend simulacija. Testirana je mreža na dijelu podskupova koji nije korišten za učenje. Odvojivši od ukupno 27 milijuna sekvenci manji broj za testiranje. Konkretno, odvojeno je 10.000 sekvenci za ovu svrhu. Ovaj problem je izuzetno kompleksan i za pokrivanje većeg prostora početnih uvjeta, s većim razlikama među njima, potrebno je ne samo koristiti veću mrežu, već i znatno veću bazu podataka. Ipak, mreža može konvergirati kada se radi o početnim uvjetima koji su bliski onima na kojima je trenirana.

6 Zaključak

U ovom radu imali smo zamisao generiranja simulacija problema triju tijela. To smo ostvarili pomoću vrlo učinkovite Runge-Kutta-Fehlberg metode, a potrebnu brzinu postigli smo korištenjem višejezgrenog procesuiranja. Problemi su nastali zbog velikog zauzimanja memorije koja nije bila dostupna. Taj smo problem riješili smanjenjem broja simulacija i kompresijom podataka u pickle format. Testiranje same neuralne mreže na manjim podacima trajalo je dugo kako bismo s povjerenjem mogli započeti trening mreže na glavnim podacima. Neuralna mreža je prilično sporo konvergirala. Kroz razne optimizacije postignuto je smanjenje funkcije gubitka na vrlo nisku vrijednost. Međutim, takav rezultat nije bio moguće postići na glavnim podacima zbog ograničenosti računala i nedovoljno velike mreže. Nakon što je smanjen raspon podataka, uspješno smo smanjili funkciju gubitka na vrlo dobru vrijednost, s obzirom na snagu računala. Kada je neuralna mreža puštena da sama pokuša simulirati problem, moglo se vidjeti da, uz malo buke, prilično dobro prati originalne simulacije. Pokazalo se da neuralna mreža može prilično dobro naučiti, ali joj je potreban ogroman broj podataka zbog kompleksnosti problema. Osim toga, trebala bi biti puno veća mreža, što zahtijeva snažnije računalo koje nije bilo na raspolaganju. Važno je napomenuti da umjetna inteligencija može značajno doprinijeti razvoju znanosti. Algoritam kojim se koristi je moćan alat za predviđanje novih spoznaja u raznim područjima fizike.

Ovim istraživanjem pokazano je da neuronska mreža ovog tipa može vrlo uspješno aproksimirati kretanje sustava triju tijela. Međutim, to ne znači da je mreža naučila fizičke zakone koji upravljaju njihovim gibanjima. Umjesto toga, rezultati sugeriraju da je mreža naučila ponašanje specifično za simulacije triju tijela provedene pomoću Runge-Kutta-Fehlberg metode, ali njezino razumijevanje ne ide dalje od toga. Nadalje, smanjenje prostora početnih uvjeta, koji je već bio ograničen tijekom simulacija, upućuje na to da je neuronska mreža vjerojatno naučila samo simulirati ponašanje unutar tog manjeg skupa uvjeta. Izvan tog ograničenog prostora, mreža ne pokazuje sposobnost generalizacije, što ukazuje na potrebu za daljnjim hardverskim resursima za razvoj i poboljšanje modela. Unatoč ovim ograničenjima, ovaj rad potvrđuje da je neuronska mreža izuzetno moćan alat. Uz dovoljno podataka i hardverske podrške, mogla bi postići ne samo bolje rezultate u smislu smanjenja RMSE-a, već i proširiti prostor djelovanja. Prednost neuronske mreže leži u tome što, jednom istrenirana, može brzo pružiti rješenja bez potrebe za dodatnim vremenom učenja. Također, uvijek postoji mogućnost dodatne obuke na novim podacima kako bi se postigli finiji i precizniji rezultati.

7 Literatura

[1] Barrow-Green, J. *Poincaré and the three body problem* 11 (American Mathematical Soc., 1997).

[2] Weinberger P. *Niels Bohr and the dawn of quantum theory* (Philosophical Magazine, 2014).

[3] John Ellis, Mary K. Gaillard and Dimitri V. Nanopoulos *A Historical Profile of the Higgs Boson* URL: https://arxiv.org/abs/1201.6045, 2012.

[4] Charles Petzold, *The Annotated Turing* (Wiley Publishing, Inc, Indianapolis, Indiana 2008).

[5] V. Rajaraman *John McCarthy – Father of Artificial Intelligence* (Resonance of Indian Academy of Sciences, Volume 19, Issue 3 (March 2014))

[6] Tingwu Wang, *Recurrent Neural Network* , Machine Learning Group, University of Toronto for CSC 2541, Sport Analytics

[7] Ralf C. Staudemeyer, Eric Rothstein Morris *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks* URL: https://arxiv.org/abs/1909.09586, 2019

[8] Wikipedia *Newton's Universal law of Gravitation* URL: https://en.wikipedia.org/wiki/Newton%27s_law_of_universal_gravitation

[9] Wikipedia *Three-body Problem* URL: https://en.wikipedia.org/wiki/Three-body_problem

[10] Sundman, K. F. *Theorie der planeten* (1915).

[11] Adam E. Parker *Runge-Kutta 4 (and Other Numerical Methods for ODEs)* (2021).

[12] Kevin Gurney *An introduction to neural networks*, UCL Press (1997)

[13] Kiprono Elijah Koech, *The Basics of Neural Networks (Neural Network Series) — Part 1*, Medium (03.05.2022) URL: https://towardsdatascience.com/the-basics-of-neural-networks-neural-network-series-part-1-4419e343b2b

[14] Hossein Ashtari, *What Is a Neural Network? Definition, Working, Types, and Applications in 2022* URL: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-a-neural-network/

[15] *Architecture of a LSTM Unit* URL: https://d2l.ai/chapter_recurrent-modern/lstm.html)

[16] Arishnama, *Understanding Bidirectional LSTM for Sequential Data Processing*, Medium, 2023.

[17] Isibor Kennedy Ihianle, Augustine O. Nwajana, Richard I Otuka, Solomon Ebenuwa *A Deep Learning Approach for Human Activities Recognition From Multimodal sensing Devices*, (IEEE Access), 2020.

[18] Christian Arnold, Luka Biedebach, Andreas Küpfer, Marcel Neunhoeffer *The role of hyperparameters in machine learning modelsand how to tune them*, Political Science Research and Methods (2024)

[19] URL: https://www.researchgate.net/figure/The-LSTM-model-architecture-which-includes-three-LSTM-layers-two-batch-normalization_fig2_361033469