Troslojna perceptronska mreža s ReLU aktivacijskom funkcijom

Mačečević, Dominik

SVEUČILIŠTE U ZAGREBU

Fakultet strojarstva i brodogradnje

ZAVRŠNI RAD

Dominik Mačečević

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:

prof. dr. sc. Danko Brezak, dip. ing.

Student:

Dominik Mačečević

Zagreb, 2024.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem se svom mentoru **prof. dr. sc. Danku Brezaku** na podršci i savjetima prilikom izrade završnog rada. Također se zahvaljujem obitelji i prijateljima na podršci tijekom studija.

Dominik Mačečević

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Dominik Mačečević** JMBAG: **0035228842**

Naslov rada na hrvatskom jeziku: **Troslojna perceptronska mreža s ReLU aktivacijskom funkcijom**

Naslov rada na engleskom jeziku: **A three-layered perceptron neural network with ReLU activation function**

Opis zadatka:

Za aktivacijsku funkciju neurona skrivenog sloja kod višeslojnih perceptronskih neuronskih mreža najčešće su korištene sigmoidna (logistička) funkcija i hiperbolni tangens, a pojavom struktura dubokih mreža i ReLU (Rectified linear unit) funkcija te njezina aproksimacija u obliku Softplus funkcije. ReLU funkcija uvedena je u cilju rješavanja problema nestajućeg gradijenta koji se javlja kod primjene mreža s više sakrivenih slojeva. Propagacijom gradijenata pogreške kroz niz slojeva dolazi do smanjenja njihova iznosa, što uzrokuje vrlo sporo učenje ili ga u potpunosti onemogućuje.
U ovome je radu provedena usporedna analiza navedenih aktivacijskih funkcija primjenom nekoliko standardnih usporednih testova u cilju određivanja primjene ReLU tipa funkcije i kod mreža s jednim sakrivenim slojem.

U radu je potrebno:

1. Izraditi programsku podršku za četiri varijante unaprijedne perceptronske neuronske mreže učene EBP metodom učenja, čiji neuroni sakrivenog sloja sadrže sigmoidalnu, tangens-hiperbolnu, ReLU i Softplus aktivacijsku funkciju.
2. Generirati odzive mreže za sve varijante mreže primjenom nekoliko standardnih usporednih testova.
3. Usporediti dobivene rezultate različitih struktura mreže.
4. Izvesti zaključke rada.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
30. 11. 2023.	**1. rok:** 22. i 23. 2. 2024. **2. rok (izvanredni):** 11. 7. 2024. **3. rok:** 19. i 20. 9. 2024.	**1. rok:** 26. 2. – 1. 3. 2024. **2. rok (izvanredni):** 15. 7. 2024. **3. rok:** 23. 9. – 27. 9. 2024.

Zadatak zadao: Predsjednik Povjerenstva:

Prof. dr. sc. Danko Brezak Prof. dr. sc. Damir Godec

Sadržaj

Popis slika

Popis kratica

APRBS eng. *Amplitude modulated Pseudo-Random Binary Signal*.

ASCII eng. *American Standard Code for Information Interchange*.

EBP Povratno rasprostiranje greške (eng. *Error backpropagation*).

NRMS Normalizirana srednja kvadratna pogreška eng. *Normalized root mean square error*.

POSIX eng. *Portable Operating System Interface*.

XOR Ekskluzivno ili (eng. *Exclusive or*).

Sažetak

Za aktivaciju neurona u višeslojnoj perceptronskoj neuronskoj mreži se najčešće koriste sigmoidalna aktivacijska funkcija i hiperbolni tangens, a kod dubokih i konvolucijskih neuronskih mreža ReLU i njezina aproksimacija Softplus aktivacijska funkcija. ReLU funkcija je uvedena za rješavanje problema nestajućeg gradijenta koji se javlja u neuronskim mrežama s više saktivenih slojeva. U ovome je radu provedena usporedba navedenih aktivacijskih funkcija na problemima identifikacije dinamičkih sustava i predviđanju Mackey-Glass vremenske serije s ciljem ispitivanja svojstava ReLU aktivacijske funkcije kod neuronskih mreža s jednim krivenim slojem.
Analiza je pokazala kako ReLU aktivacijska funkcija daje bolje odzive od sigmoidalne i tangens-hiperbolne aktivacijske funkcije u rješavanju problema identifikacije identifikacije dinamičkih sustava, a lošije odzive na problemima predviđanja nelinearnog kaotičnog sustava. Jednostavna implementacija i optimizacija ReLU funkcije čini ju dobrim kandidatom za rješavanje problema identifikacije dinamičkih sustava.

Ključne riječi: Neuronske mreže; ReLU; Softplus; Identifikacija dinamičkog sustava

Abstract

For the activation of neurons in multilayer perceptron neural networks, sigmoidal and hyperbolic-tangent activation functions are most commonly used, while for deep and convolutional neural networks, ReLU and its approximation, the Softplus activation functions, are often used. The ReLU function was introduced to solve the vanishing gradient problem that occurs in deep neural networks. In this work, an analysis of activation functions was conducted on the identification of dynamic systems and prediction of the Mackey-Glass time series in order to examine the properties of the ReLU activation function in neural networks with a single hidden layer.
It was shown that the ReLU activation function produces better responses than sigmoidal and hyperbolic-tangent activation functions in the identification of dynamic systems, but it performs worse in predicting a nonlinear chaotic system. The simplicity of implementing and optimizing of the ReLU function makes it a good candidate for solving the identification of dynamic systems.

Keywords: Neural networks; ReLU; Softplus; Dynamic system identification

1 Uvod

1.1 Neuronske mreže

Unaprijedne višeslojne perceptronske neuronske mreže ili eng. *error backpropagation mreže* (EBP mreže) su modeli strojnog učenja. Cilj unaprijedne neuronske mreže je aproksimacija neke funkcije $y = f^*(x)$, pri čemu se ulaz x preslikava u izlaz y.[1]

1.2 Učenje neuronske mreže

Razlikujemo tri načina učenja[2]:

- učenje s učiteljom (eng. *Supervised learning*),
- učenje bez učitelja (eng. *Unupervised learning*),
- kombinacija prethodnih (eng. *Reinforcement learning*).

Za ovaj rad je korišteno samo učenje s učiteljom (eng. *Supervised learning*). Učenje se obavlja u dvije osnovne faze:

- unaprijedna faza,
- povratna faza.

1.2.1 Unaprijedna faza učenja

U unaprijednoj fazi učenja se uzimaju vrijednosti ulaza neuronske mreže Z, potom se izračunavaju izlazne vrijednosti neuronske mreže O. Kako bi u prvom koraku učenja bilo moguće izračunati izlazne vrijednosti, potrebno je inicijalizirati težine skrivenog i izlaznog sloja V, W. Sve težine su nasumično generirane u intervalu između -1 i 1 i zatim su dobiveni iznosti težina korišteni za sve primjere kako bi početak učenja neuronske mreže bio jednak za bilo koji primjer. Svi ulazni podaci su normirane u intervalu između -1 i 1.

Skriveni sloj[2]:
Funkcija sume *net* neurona skrivenog sloja H, dobiva prvi indeks oznake sloja (net_H), te za svaki j-ti neuron dobiva drugi indeks i računa se na slijedeći način:

$$net_{Hj} = \sum_{i=1}^{I} v_{ij} Z_i;\ j = 1, 2, ..., J-1;\ i = 1, 2, ..., I, \tag{1.1}$$

gdje je I broj ulaznih neurona +1, a J predstavlja broj neurona u sakrivenom sloju uvećan za Bias.
Za aktivacijsku funkciju je korištena sigmoidalna funkcija prikazana slikom 1.1a, te se izlaz skrivenog sloja računava prema:

$$y_j = \frac{1}{1 + e^{-net_{Hj}}};\ j = 1, 2, ..., J-1; y_J = 1;\ Bias. \tag{1.2}$$

Izlazni sloj[2]:
Kod izlaznog sloja funkcija sume net neurona izlaznog sloja O dobiva prvi indeks pripadnog sloja (net_O), a može se zapisati izrazom:

$$net_{Ok} = \sum_{j=1}^{J} w_{kj} y_j;\ k = 1, 2, ..., K, \tag{1.3}$$

gdje je K broj neurona izlaznog sloja.
Za aktivacijsku funkciju izlaznog sloja je korištena linearna funkcija, stoga se izlaz neuronske mreže može prikazati kao:

$$O_k = K_p net_{Ok};\ k = 1, 2, ..., K, \tag{1.4}$$

gdje je K_p nagib linearne aktivacijske funkcije. Korišten je nagib $K_p = 1$.

1.2.2 Povratna faza učenja

U povratnoj fazi učenja se izračunava greška učenja iz ostvarenog i željenog izlaza neuronske mreže za svaki ulazno-izlazni par. Promjena težinskih faktora se vrši na osnovu izračunate greške učenja. Korištena je suma kvadrata pogreške kao mjera odstupanja izlaza mreže od željene vrijednosti izlaza:

$$E = \frac{1}{2} \sum_{n=1}^{N} (d_n - O_n), \tag{1.5}$$

gdje je N broj elementa u skupu za učenje. Promjenom težinskih faktora neuronske mreže se nastoji minimizirati funkciju cilja prikazanu izrazom 1.5. Težinski faktori ϑ se mijenjaju prema izrazu[2]:

$$\vartheta(n+1) = \vartheta(n) + \Delta\vartheta(n), \tag{1.6}$$

gdje je n trenutni korak učenja, $\Delta\vartheta(n)$ veličina promjene težinskog faktora, a $\vartheta(n+1)$ je nova vrijednost težinskih faktora.
Kako bi se greška smanjivala u najvećem mogućem iznosu potrebno je izračunati gradijent pogreške:

$$\nabla E(\vartheta) = \frac{\partial E(\vartheta)}{\partial \vartheta}. \tag{1.7}$$

Kako bi se smanjio broj iteracija potrebnih za učenje algoritmom povratnog prostiranja pogreške, uz koeficijent brzine učenja η, dodaje se i momentum α. Veličina promjene težinskih faktora izračunava se prema:

$$\Delta\vartheta(n) = -\eta\nabla E(\vartheta(n)) + \alpha\Delta(n-1), \tag{1.8}$$

gdje je n trenutna, a (n-1) je predhodna promjena težinskog faktora. Vrijednosti koeficijenta η i α bira učitelj. Konačni oblik izraza za promjenu težinskog faktora je:

$$\vartheta(n+1) = \Delta\vartheta(n) - \eta\nabla E(\vartheta(n)) + \alpha\Delta(n-1). \tag{1.9}$$

1.2.2.1 Promjena težina izlaznog sloja

Promjena težinskih faktora koji povezuju izlazni i sakriveni sloj odvija se prema 1.9 na slijedeći način[2]:

$$w_{kj}(n+1) = w_{kj}(n) - \eta\nabla E(n) + \alpha\Delta w_{kj}(n-1). \tag{1.10}$$

Gradijent greške izračunava se prema izrazu 1.7:

$$\nabla E(n) = \frac{\partial E(n)}{\partial w_{kj}}. \tag{1.11}$$

Gradijent greške se može lako odrediti primjenom uzastopnih parcijalnih derivacija:

$$\frac{\partial E(n)}{\partial w_{kj}} = \frac{\partial E(n)}{\partial O_k}\frac{\partial O_k}{\partial net_{Ok}}\frac{\partial net_{Ok}}{\partial w_{kj}}. \tag{1.12}$$

Lako je izračunati pojedine članove izraza 1.12:

$$\frac{\partial E(n)}{\partial O_k} = -(d_k - O_k), \tag{1.13}$$

$$\frac{\partial O_k}{\partial net_{Ok}} = 1, \tag{1.14}$$

$$\delta_{Ok} = d_k - O_k, \tag{1.15}$$

$$\frac{\partial net_{Ok}}{\partial w_{kj}} = y_j. \tag{1.16}$$

Uvrštavanjem izraza 1.13, 1.14, 1.15 i 1.16 u izraz 1.12 dobiven je izraz:

$$\nabla E(n) = \frac{\partial E(n)}{\partial w_{kj}} = -(d_k - O_k)y_j = \delta_{Ok}y_j. \tag{1.17}$$

Uvrštavanjem izraza 1.17 u izraz 1.10 dobiven je konačni algoritam promjene težina izlaznog sloja:

$$w_{kj}(n+1) = w_{kj}(n) - \eta\delta_{Ok}y_j + \alpha\Delta w_{kj}(n-1). \tag{1.18}$$

1.2.2.2 Promjena težina skrivenog sloja

Jednadžba pomoću koje se vrši promjena težina koje povezuju sakriveni i ulazni sloj ne razlikuju se bitno od jednadžbe za promjenu izlaznog sloja 1.10[2]:

$$v_{ji}(n+1) = v_{ji}(n) - \eta \nabla E(n) + \alpha \Delta v_{ji}(n-1). \tag{1.19}$$

Jednako kao i kod izlaznog sloja koristi se uzastopno parcijalno deriviranje za određivanje gradijenta:

$$\frac{\partial E(n)}{\partial v_{ji}} = \frac{\partial E(n)}{\partial y_j} \frac{\partial y_j}{\partial net_{Hj}} \frac{\partial net_{Hj}}{\partial v_{ji}}. \tag{1.20}$$

Lako je izračunati pojedine članove izraza 1.20:

$$\frac{\partial E(n)}{\partial y_{ji}} = -\sum_{k=1}^{K} \delta_{Ok} w_{kj}, \tag{1.21}$$

derivacija aktivacijske funkcije određuje se iz izraza 1.2 i glasi:

$$\frac{\partial y_j}{\partial net_{Hj}} = \frac{e^{-x}}{(1+e^{-x})^2}, \tag{1.22}$$

derivacija skrivenog sloja određuje se iz izraza 1.1 i glasi:

$$\frac{\partial net_{Hj}}{\partial v_{ji}} = Z_i. \tag{1.23}$$

Uvrštavanjem izraza 1.21, 1.22 i 1.23 u izraz 1.19 dobiven je konačni algoritam promjene težina skrivenog sloja:

$$v_{ji}(n+1) = v_{ji}(n) + \eta \frac{e^{-x}}{(1+e^{-x})^2} Z_i \left(\sum_{k=1}^{K} \delta_{Ok} w_{kj} \right) + \alpha \Delta v_{ji}(n-1). \tag{1.24}$$

Numerički postupak za preostale skrivene slojeve je analogan postupku za prvi skriveni sloj, samo se izrazi proširuju. Za druge aktivacijske funkcije mijenjaju se samo pripadajuće derivacije aktivacijske funkcije, dok bi izvodi ostalih parcijalnih derivacija ostali jednaki.

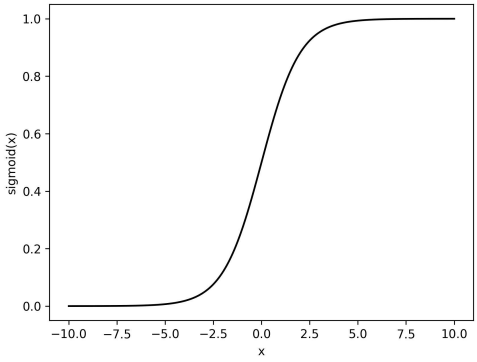
1.3 Uloga aktivacijske funkcije

Neuronska mreža bez aktivacijskih funkcija bi bila ekvivalentna jednoj matrici težina u jednom skrivenom sloju. Stoga uvođenjem aktivacijskih funkcija je ostvarena nelinearnost u modelu strojnog učenja. Nelinearnost je korisna u modelima strojnog učenja jer većina problema iz stvarnog svijeta ima nelinearne karakteristike.[3]

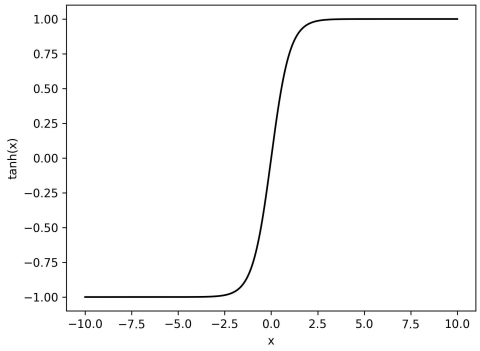
1.3.1 Usporedba aktivacijskih funkcija

Najčešće korištene aktivacijske funkcije u EBP neuronskim mrežama su sigmoidalna funkcija (slika 1.1a) i tangens hiperbolni (slika 1.1b). Kod konvolucijskih neuronskih mreža dolazi do problema nestajućeg gradijenta zbog čega su pogodne ReLU (slika 1.1c) i softplus (slika 1.1d) aktivacijske funckije. Budući da ReLU i softplus funkcije nisu ograničene u nekom intervalu, kao što je sigmoidalna u intervalu $[0, 1]$ i tangens hiperbolni u intervalu $[-1, 1]$, Vrijednosti na izlazu iz njih mogu biti jako velike što narušava numeričku stabilnost modela. Kod višeslojnih perceptronskih mreža nije pogodno stavljati ReLU ili softplus slojeve zaredom jer to omogućuje veliki rast na izlazu iz tih slojeva što može uzrokovati divergiranje od optimalnog odziva neuronske mreže. [1]
Izrazi koji definiraju aktivacijske funckije[4]:

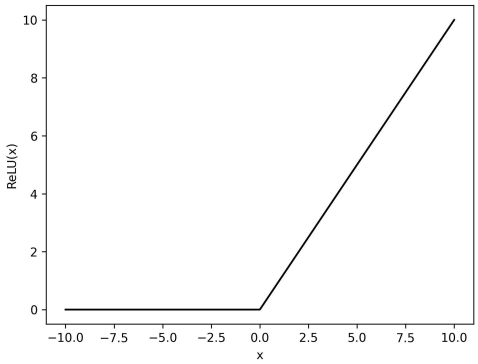
- Sigmoidalna funckije
$$f(x) = \frac{1}{1 + e^{-x}}$$
- Tangens hiperbolni
$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$
- ReLU funckije
$$f(x) = max(0, x)$$
- Softplus funckije
$$f(x) = ln(1 + e^{x})$$



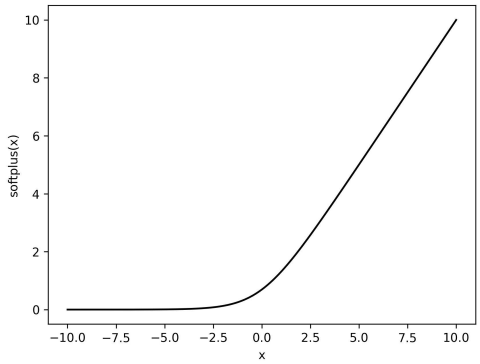
(a) Sigmoidalna funkcija



(b) Tangens hiperbolni



(c) ReLU funkcija



(d) Softplus funkcija

Slika 1.1: Prikaz korištenih aktivacijskih funkcija

1.3.2 Implementacija aktivacijskih funkcija u programskom jeziku C

Prednost ReLU aktivacijske funkcije nad ostalim je njezina izuzentno jednostavna implementacija. Zbog jednostavnosti ReLU aktivacijske funkcije moguće je provesti optimizacije kao što su "inlineing" što smanjuje stack, ubrzava proceduru s "inplace" zapisivanjem rezultata u memoriju i nije potrebno koristiti "math.h" modulom koji ne mora biti dostupan na svim platformama.[5]
Iz slijedećeg ispisa implementacija vidljivo je da je ReLU procedura daleko najjednostavnija, dok ostale procedure pozivaju dodatne procedure sadržane u "math.h" modulu, kao što su "exp" i "log" procedure. Tangens hiperbolni nije prikazan jer je sadržan u "math.h" modulu.

Implementacija sigmoid funkcije:

```
static double
sigmoid(double x)
{
    double y = 1 / (1 + exp(-x));
```

```
    return y;
}
```

Implementacija ReLU funkcije:

```
static double
relu(double x)
{
    double y = x < 0 ? 0 : x;
    return y;
}
```

Implementacija softplus funkcije:

```
static double
softplus(double x)
{
    double y = log(1 + exp(x));
    return y;
}
```

2 Programska podrška za učenje neuronskih mreža

2.1 Opis rada programa

Programska podrška je napravljena u programsku jeziku C bez ikakvih dodatnih modula osim standardnog modula za komunikaciju između programa i Linux kernela. Korištena je verzija C99 bez ekstenzija koji je ISO standard C-a iz 1999. godine (ISO/IEC 9899:1999). Svi moduli su korišteni u POSIX modu kako bi se osigurao rad programa na raznim platformama koje podržavaju POSIX standard.[6]
Opis rada programa je prikazan na XOR problemu zbog njegove jednostavnosti, a to je najjednostavniji problem koji nije linearno separabilan pa ga ima smisla rješavati pomoću EBP neuronskih mreža.
Program radi tako da prvo pročita konfiguracijsku datoteku pod imenom NNfile ili nnfile, alocira memoriju za neuronsku mrežu, podatke potrebne za učenje i za sve ostale parametre potrebne za učenje, prema vrijednostima u konfiguracijskoj datoteci iterativno izvršava algoritam povratnim rasprostiranjem greške (EBP) učenja dok nije postignuti željeni NRMS ili željeni broj iteracija.

Nakon završenog učenja program kreira direktorij _build/ unutar kojega sprema generirane datoteke bytecode.c i ann.h.

U datoteku bytecode.c su spremljeni svi naučeni parametri i topologija mreže, a ann.h datoteka je stb-style modul koji omogućava implementaciju naučene neuronske mreže gotovo na bilo kojoj platformi.

2.2 Konfiguracijska datoteka

Sve linije koje počinju s karakterom "#" su ignorirane kao komentari.
Konfiguracijska datoteka se sastoji od tri dijela:

- config
- topology
- data.

"Config" dio sadrži sve konstante potrebne za učenje napisane u ASCII tekstu kako bi čitanje datoteke bilo jednostavno.
Vrijednost su označene na sljedeći način:

- brzina učenja η: eta
- momentum α: alpha
- broj iteracija: N
- željeni NRMS: NRMS.

"Topology" dio sadrži topologiju neuronske mreže s tipovima slojeva, aktivacijskim funkcijama i brojem neurona. Kako bi se omogućilo daljnje proširenje funkcionalnosti programa i osigurala čitljivost konfiguracijske datoteke aktivacijski dio sloja neuronske mreže je prikazan kao zasebni sloj bez obzira što se gleda zajedno s težinama kao jedan sloj. Sintaksa je inspirirana programskim jezikom Ocaml gdje se takozvani "pipe" operator (|>) koristi za transfer podataka iz izlaza jedne funkcije u ulaz druge.[3] Na sličan način je prikazan tok podataka kroz neuronsku mrežu od ulaznog do izlaznog sloja.

"Data" dio sadrži podatke za učenje napisane u ASCII tekstu za jednostavno i brzo čitanje. Svaki redak predstavlja jednu instancu ulazno-izlaznog para prema kojem se računa gradijent. Ulazni set podataka je odjeljen od izlaznog s dva karaktera "::" koji omogućavaju lako procesiranje podataka kod učitavanja vrijednosti na Heap.
Primjer NNfile datoteke za XOR problem je sadržan u dodatku A.1.

2.3 Alociranje memorije

Kako bi programska podrška podržala neuronske mreže raznih veličina i topologija potrebno je na pravilan način alocirati i osloboditi memoriju kako algoritam učenja nebi imao usko grlo na pristupu memoriji.
Nakon čitanja konfiguracijske datoteke alocirna je struktura podataka u kojoj su sve težine složene u jednostavne liste. Pošto je potrebno računanje umnoška matrica, sve težine su složene u dio memorije koji je u jednom nizu. To osigurava da pokazivač na težinu dolazi do iduće potrebne vrijednosti jednostavnim inkrementom. Ta jednostavna optimizacija omogućava da težine za vrijeme računanja umnoška matrica ostaju u predmemoriji procesora dok se učitava drugi dio težina.
Kompliciranije strukture podataka koje bi trebale biti brže kod većeg broja težina samo narušavaju performanse kod manjih neuronskih mreža. Jednostavnost pristupu podataka

drastično ubrzava proces učenja jer je vrijeme računanja neznačajno naspram vremenu pristupa podacima.

2.4 Algoritam učenja

Učenje se odvija stohastičkim povratnim rasprostiranjem greške. Dvije su osnovne faze učenja, unaprijedna i povratna faza. U stohastičkoj proceduri svaki ulazno-izlazni par mijenja parametre učenja što razultira bržim konvergiranjem neuronske mreže.[2] Učenje se odvija u pet funkcija:

- "feedfoward" za unaprijednu fazu učenja
- "loss" za računanje odstupanja dobivenog izlaza od željenog
- "backpropagation" za računanje veličine promjene težina
- "update_params" za računanje novih parametara
- "get_nrms" za računanje dobivenog NRMS nakon jednog koraka učenja.

"Feedfoward" funkcija uzima kao argument pokazivač na strukturu u kojoj je sadržana cijela neuronska mreža, a vraća jedan cijeli broj koji je nula ako je funkcija uspješna i manji od nule ako nešto nije uspjelo. Taj način pozivanja funkcije omogućava jednostavno provjeravanje grešaka. Budući da se ne kopira memorija nego funkcija uzima samo adresu, svako pozivanje funkcije neznačajno utječe na performanse.
"Feedfoward" funkcija je odgovorna za unaprijednu fazu učenja. Prema topologiji mreže računa se svaki sloj i izlaz sprema u privremeni pokazivač koji slijedeći sloj koristi kao ulaz. Kako bi prolaz kroz topologiju mreže bio što brži koristi se "switch-case" operator koji rezultira "lookup" tablicom u dobivenom assembly programu nakon kompiliranja programa. Na sličan način, svi pokazivači na aktivacijske funkcije su stavljeni u listu, što omućava i pristup aktivacijskoj funkciji u minimalnom mogućem vremenu s obzirom na kompleksnost topologije neuronske mreže. Implementacija "feedfoward" funkcije je prikazana u dodatku B.1.

"Loss" funkcija uzima kao argumente pokazivač na strukturu neuronske mreže i strukturu podataka, a vraća, na jednaki način kao i "feedfoward", funkcija cijeli broj, koji je nula ako je funkcija uspješna i manji od nule ako nešto nije uspjelo.
Greška se računa prema izrazu za sumu kvadrata pogreške prikazanim izrazom 1.5. Implementacija "loss" funkcije je prikazana u dodatku B.2.

"Backpropagation" funkcija uzima kao argumente pokazivač na strukrutu neuronske mreže i pokazivač na klon strukture neuronske mreže u kojoj su sadržane veličine promjene težine, a vraća, na jednaki način kao i "feedfoward" funkcija, cijeli broj, koji je nula ako je funkcija uspješna i manji od nule ako nešto nije uspjelo.
"Backpropagation" funkcija je odgovorna za povratnu fazu učenja. Implementacija te funkcije je gotovo identična kao i "feedfoward" funkcije, razlika je što funkcija prolazi kroz topologiju mreže u suprotnom smjeru (od izlaznog sloja do ulaznog sloja) i računa veličine promjene težine koje sprema u klon strukture neuronske mreže. Implementacija "backpropagation" funkcije je prikazana u dodatku B.3.

"Update_parms" funkcija uzima kao argumente pokazivač na strukrutu neuronske mreže i pokazivač na klon strukture neuronske mreže u kojoj su sadržane veličine promjene težine, a vraća, na jednaki način kao i "feedfoward" funkcija, cijeli broj, koji je nula ako je funkcija uspješna i manji od nule ako nešto nije uspjelo.
"Update_parms" računa nove težine neuronske mreže. Ta procedura je odvojena u zasebnu funkciju kako se ne bi miješala s računanjem gradijenta. Pošto se sve težine mijenjaju u memoriji koja je napočetku alocirana za njih, bez kopiranja memorije, potrebno je računanje novih težina izvršiti nakon izračunatog gradijenta cijele neuronske mreže. Implementacija "update_parms" funkcije je prikazana u dodatku B.4.

"Get_nrms" funkcija uzima kao argumente pokazivač na strukturu neuronske mreže i pokazivač na strukturu podataka, a vraća iznos NRMS-a. Ako je željeni NRMS 0 nije potrebno računati NRMS do zadnje iteracije učenja. U tom slučaju se ne prekida učenje do zadnje iteracije definirane u konfiguracijskoj datoteci, stoga se ne poziva funkcija za računanje NRMS do kraja učenja. Implementacija "get_nrms" funkcije je prikazana u dodatku B.5.

2.5 Generiranje datoteke bytecode.c

Datoteka "bytecode.c" je generirana nakon procesa učenja. Cijela topologija mreže zajedno sa svim parametrima je upisana u jednu datoteku u ANSI C standardu koja je potpuno neovisna o bilo kakvim modulima. To omogućava da se ta jedna datoteka prenosi na sve platforme i naučena neuronska mreža je primjenjiva na bilo kojem računalu za koje postoji C compiler.
Datoteka se sastoji od dvije enumeracije, koje definiraju korištene aktivacijske funkcije i korištene tipove slojeva, od jednodimezionalne liste u kojoj su sve težine po redu upi-

sane što osigurava statički alocirani dio memorije i od dvodimenzionalne liste u kojoj su sadržani podaci o topologiji mreže. Zbog jednostavnosti tipova podataka jendostavno je implementirati razne funkcije koje mogu učitati simbole iz bytecode.c datoteke i koristiti naučeni model. Jedan od mogućih načina korištenja tako pohranjenih parametara je prikazan u "ann.h" datoteci.
Primjer generirane bytecode.c datoteke za XOR problem sadržan je u dodatku A.2.

2.6 Rad ann.h modula

Modul "ann.h" sadržan od samo jedne datoteke izveden je u tzv. "stb-style" načinu. Svi simboli su definirani u prvom dijelu datoteke, a implementacija je definirana u drugom dijelu ispod "#ifdef IMPLEMENTATION". Taj način omogućava da se, pri korištenju modula, definira "IMPLEMENTATION" macro koji učita zadanu implementaciju funkcija. U protivnom je moguće definirati svoju specifičnu implementaciju u zasebnoj .c datoteci ako je potrebno zbog nekih posebnih zahtjeva ili posebnog hardware-a.
U ovom primjeru postoji jedna funkcija pod nazivom "snn" koja uzima kao argumente adresu liste alocirane za izlaz i liste alocirane za ulaz, a vraća cijeli broj koji je nula ako je funkcija uspješna. Ostale funkcije su potrebne za rad "snn" funkcije i nije ih potrebno pozivati iz drugih datoteka tako da su definirane kao statičke. Modul "ann.h" dobiva simbole iz datoteke "bytecode.c" prilikom procesa linking-a programa pa nema potrebe za dodatnu .h datoteku za unos simbola iz datoteke "bytecode.c" u modul "ann.h". Ostatak programa može pozvati funkciju neuronske mreže "snn" bez ikakvog znanja o implementaciji neuronske mreže i koristiti gotov model za rješavanje nekog problema gdje samo za željeni ulaz dobije željeni izlaz. Primjer ann.h datoteke je sadržan u dodatku A.3.

2.7 Korištenje naučene mreže u produkciji

U dodatku A.4 prikazano je rješenje XOR problema s prije prikazanim datotekama "bytecode.c" i "ann.h". Ovaj jednostavan problem pokazuje kako je, nakon učenja, jednostavno iskoristiti naučeni model u bilo kojem sustavu jer nema nikakvih posebnih formata datoteka ili dodatnih modula, samo ANSI C koji je podržan na svakoj platformi.

3 Opis provedenih testova

Za određeni problem sve analizirane varijante algoritma neuronske mreže u ovome radu učene su s istim paramerima. Nasumično generirane vrijednost težina na početku učenja su jednake na početku svakog učenja kako bi se osigurali jednaki početni uvjeti od kojih kreće učenje. Željeni NRMS je svugdje postavljen kao nula da bi se osigurao jednak broj iteracija za svaku aktivacijsku funkciju.
Provođenje testova je automatizirano pomoću alata "GNU Make" koji je korišten kao ekspertni sustav. Obrada podataka prije učenja, generiranje NNfile datoteka, učenje i testiranje neuronskih mreža i grafički prikaz rezultata je izveden pomoću rekurzivnih Makefile datoteka. U Makefile datotekama su navedena sva pravila kojim redoslijedom se moraju izvršavati i kod promjene koje datoteke koji dijelovi testova se moraju izvršiti. Rekurzivni "make" ima poznati problem što nema potpuni usmjereni aciklički graf, što može rezultirati propustom nekih pravila koje bi potencijalno trebalo nužno provesti.[7] Poznavajući taj problem ekspertni sustav je napravljen tako da taj problem ne utječe na provođenje testova.

3.1 Odabir problema

Odabrani problemi za testiranje su standardni problemi na kojima se često provodi testiranje raznih vrsti neuronskih mreža. Skupovi podataka za učenje se lako generiraju bez potrebe za mjerenjem što smanjuje grešku uzrokovanu lošim podacima. Budući da se uspoređuju aktivacijske funkcije koje se koriste u kombinaciji s konvolucijom (ReLU i softplus) i funkcije koje se najčešće koriste za višeslojne perceptronske neuronske mreže (sigmoidalna funkcija i hiperbolni tangens) važno je ispitati kakva svojstva obrade signala imaju te funkcije bez utjecaja konvolucije.

3.2 P1 član

Dinamika P1 člana opisuje se diferencijalnom jednadnžbom prvog reda[2]:

$$Tx(t) + x(t) = K_p u(t), \tag{3.1}$$

gdje je T vremenska konstanta, a K_p konstanta pojačanja sustava.
Budući da je teško koristiti kontinuiranu jednadžbu kao izvor informacija za neuronske

mreže, jednandžba je diskretizirana na sljedeći način[2]:

$$x(n) = \frac{T_0}{T + T_0}\left[\frac{T}{T_0}x(n-1) + K_p u(n)\right], \tag{3.2}$$

gdje $x(n)$ i $u(n)$ označavaju trenutne ulazne i izlazne vrijednosti dok $x(n-1)$ je prošla vrijednost izlazne veličine, a T_0 predstavlja vrijednost perioda uzrokovanja.
Svi podaci su generirani uz parametre[2]:

- $T = 1$ s,
- $T_0 = 0.2$ s,
- $K_p = 1$.

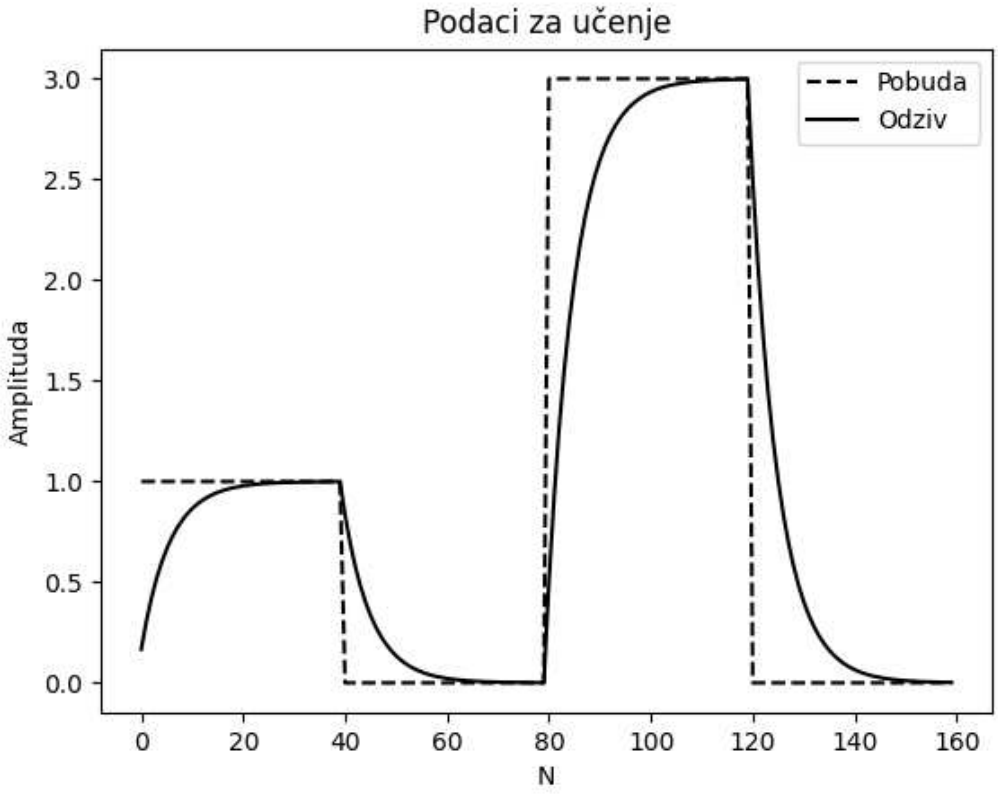
Za generiranje podataka P1 član je pobuđen funkcijom definiranom u 160 koraka (n) u formi[2]:

$$\begin{aligned} u(n) &= 1, n = 0, 2, ..., 39 \\ u(n) &= 0, n = 40, 41, ..., 79 \\ u(n) &= 3, n = 80, 81, ..., 119 \\ u(n) &= 0, n = 120, 121, ..., 159. \end{aligned} \tag{3.3}$$

Za sve primjere aktivacijskih funkcija korištena je topologija mreže s dva neurona u ulaznom sloju i s dva neurona u jednom skrivenom sloju (2-2-1). Parametri učenja su:

- $\eta = 0.01$
- $\alpha = 0.8$
- $N = 500000$
- $NRMS = 0$

Odziv P1 dinamičkog člana na pobudnu funkciju 3.3 prikazan je slikom 3.1. Taj skup podataka je korišten za učenje kao ulazno-izlazni par.



Slika 3.1: Prikaz podataka korištenih za učenje dinamike P1 dinamičkog člana.

3.3 Nelinearni sustav

Identifikacija dinamičkog nelinearnog sustava je provedena za poznatu nelinearnu jednadžbu diferencije prvog reda uz period uzorkovanja od 1 sekunde i vremensku konstantu sustava od 10 sekundi[2]:

$$x(n+1) = (0.9 - 0.003x(n))x(n) + 0.2(u). \tag{3.4}$$

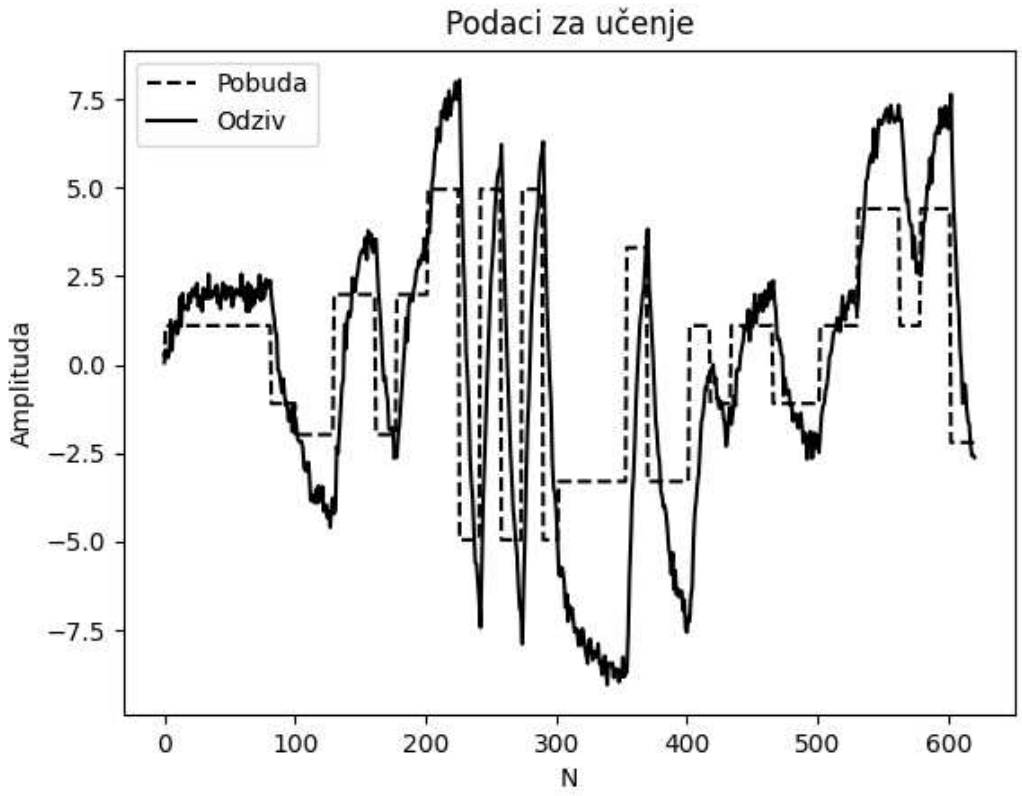
Kako bi se generiralo doboljno podataka o procesu sustav mora biti pobuđen u svim modovima procesa. Da bi se to ostvarilo sustav je pobuđen s pseudo-binarnim signalom slučajne frekvencije i slučajne amplitude (APRBS, eng. *Amplitude modulated Pseudo-Random Binary Signal*).[2]

Za sve primjer aktivacijskih funkcija korištena je topologija mreže s dva neurona u ulaznom sloju i s deset neurona u jednom skrivenom sloju (2-10-1). Parametri učenja su:

- $\eta = 0.01$
- $\alpha = 0.8$

- $N = 500000$
- $NRMS = 0.$

Skup podataka korištenih za učenje je prikazan slikom 3.2. Ulaz u mrežu je APRBS signal, a izlaz je odziv nelinearnog sustava čija je dinamika definirana izrazom 3.4.



Slika 3.2: Prikaz podataka korištenih za učenje dinamike nelinearnog dinamičkog sustava.

3.4 Predikcija Mackey-Glass sustava

Često se novi algoritmi učenja neuronskih mreža testiraju na predviđanju nelinearnih kaotičnih sustava. Jedan od takvih sustava koji ima široku primjenu je Mackey-Glass sustav. Sustav je opisan nelinearnom diferencijalnom jednadžbom s kašnjenjem τ dana izrazom[2]:

$$\dot{x} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t), \tag{3.5}$$

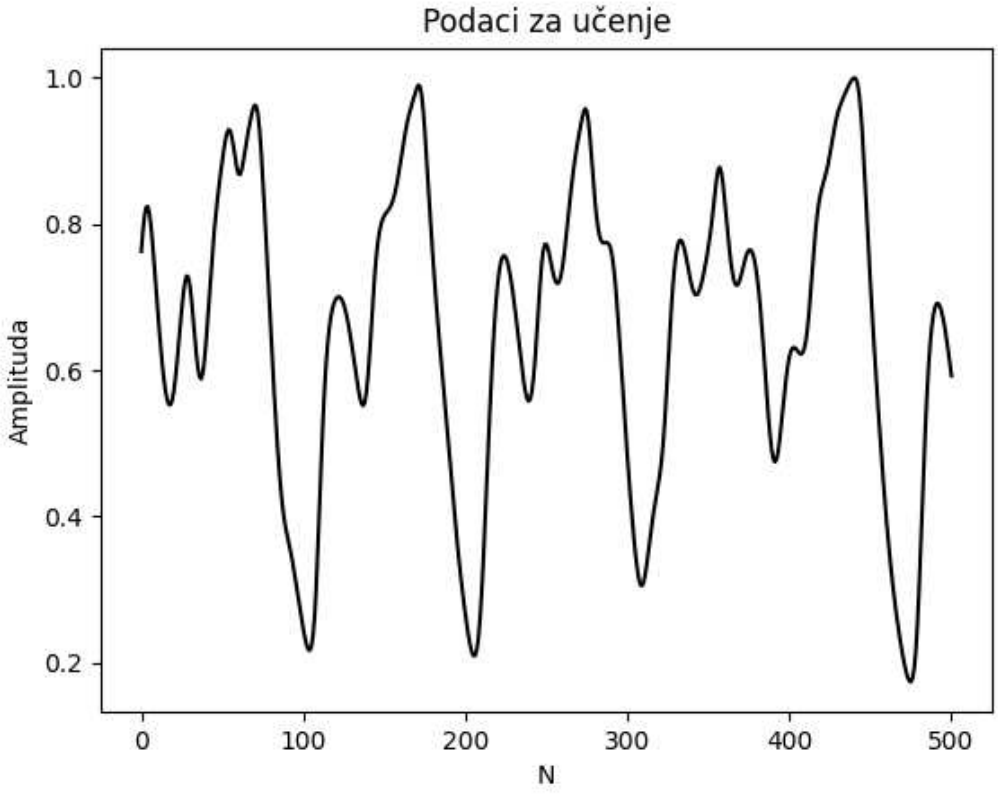
gdje su a i b parametri sustava, a t oznaka vremena. Vremenska serija od 1000 točaka za $\tau = 30$ dobivena je diskretnom jednadžbom[2]:

$$x(n) = \frac{1}{1+b}\left[x(n-1) + \frac{ax(n-\tau)}{1+x^{10}(n-\tau)}\right]. \tag{3.6}$$

Za sve primjere aktivacijskih funkcija korištena je topologija mreže s pet neurona u ulaznom sloju i s pet neurona u jednom skrivenom sloju (5-5-1). Parametri učenja su:

- $\eta = 0.01$
- $\alpha = 0.4$
- $N = 500000$
- $NRMS = 0$

Skup podataka korištenih za učenje Mackey-Glass vremenske serije je prikazan slikom 3.3.



Slika 3.3: Prikaz Mackey-Glass vremenske serije korištene za učenje.

4 Rezultati testova

4.1 Rezultati identifikacije dinamike P1 člana

Testiranje naučene mreže je provedeno pomoću skokovitih pobudnih funkcija diskretiziranih s 80 točaka. Funkcije su definirane na sljedeći način:

$$\begin{aligned} u(n) &= 0.5, n = 0, 2, ..., 39 \\ u(n) &= 0, n = 40, 41, ..., 79 \end{aligned} \tag{4.1}$$

$$\begin{aligned} u(n) &= 1.0, n = 0, 2, ..., 39 \\ u(n) &= 0, n = 40, 41, ..., 79 \end{aligned} \tag{4.2}$$

$$\begin{aligned} u(n) &= 1.5, n = 0, 2, ..., 39 \\ u(n) &= 0, n = 40, 41, ..., 79 \end{aligned} \tag{4.3}$$

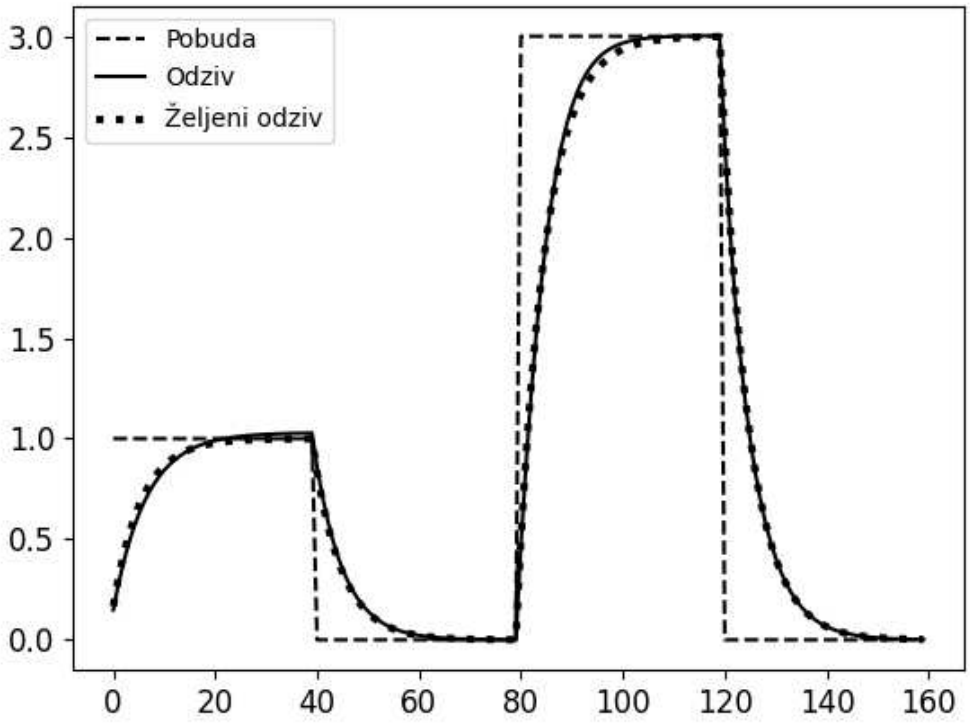
$$\begin{aligned} u(n) &= 2.0, n = 0, 2, ..., 39 \\ u(n) &= 0, n = 40, 41, ..., 79 \end{aligned} \tag{4.4}$$

$$\begin{aligned} u(n) &= 2.5, n = 0, 2, ..., 39 \\ u(n) &= 0, n = 40, 41, ..., 79 \end{aligned} \tag{4.5}$$

$$\begin{aligned} u(n) &= 3.0, n = 0, 2, ..., 39 \\ u(n) &= 0, n = 40, 41, ..., 79 \end{aligned} \tag{4.6}$$

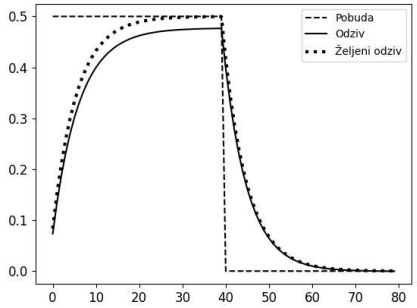
4.1.1 Sigmoidalna aktivacijska funkcija

Iz slike 4.1 je vidljivo da naučena neuronska mreža sa sigmoidalnom aktivacijskom funkcijom dobro aproksimira P1 član, ali postoji prebačaj za željenim odzivom. NRMS nakon učenja je 0.436230 .

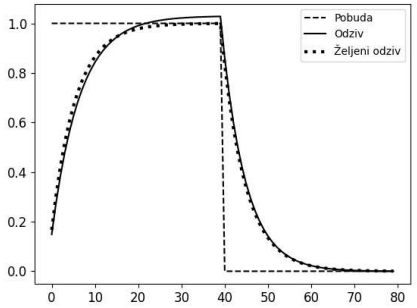


Slika 4.1: Odziv na pobudu iz podataka za učenje dinamike P1 član sa sigmoidalnom aktivacijskom funkcijom

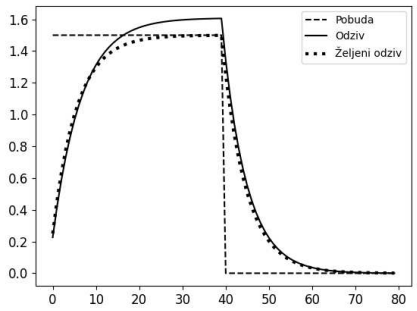
Iz slike 4.2 vidljivo je da su generalizacijska svojstva neuronske mreže sa sigmoidalnom aktivacijskom funkcijom dobra, ali ne i idealna. Aproksimacija je prihvatljiva jer nema velikog odstupanja od željenog odziva.



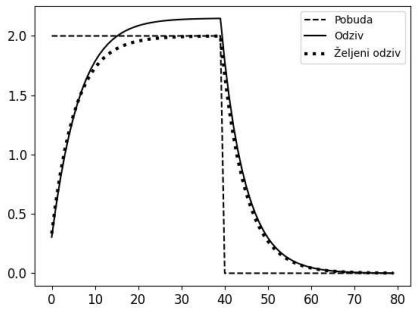
(a) Amplitude 0.5, NRMS = 0.498225



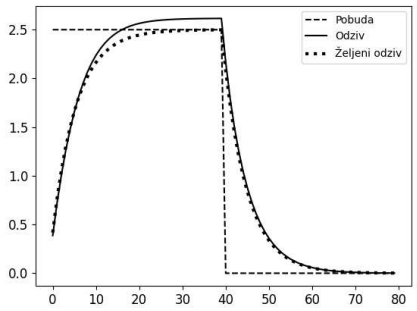
(b) Amplitude 1.0, NRMS = 0.501722



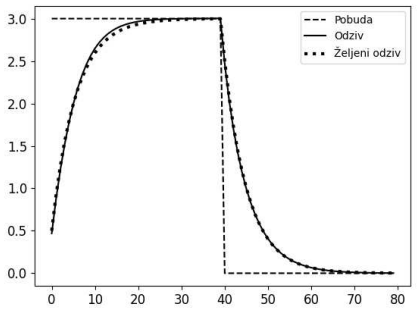
(c) Amplitude 1.5, NRMS = 0.508032



(d) Amplitude 2.0, NRMS = 0.503964



(e) Amplitude 2.5, NRMS = 0.490110

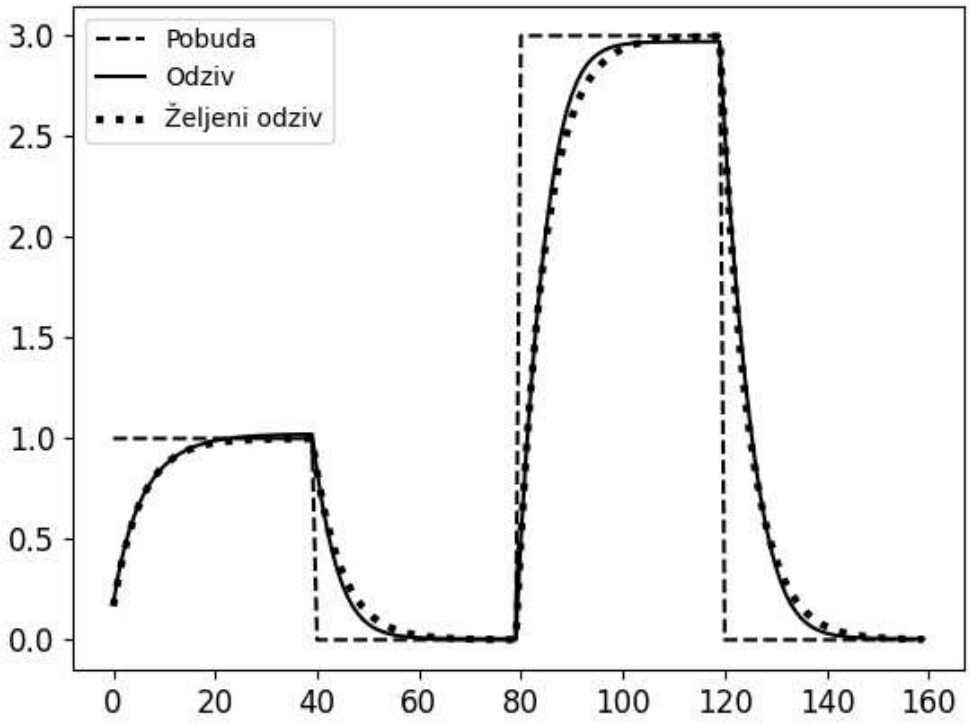


(f) Amplitude 3.0, NRMS = 0.475152

Slika 4.2: Odzivi neuronske mreže sa sigmoidalnom aktivacijskom funkcijom na testne podatke kod problema identifikacije P1 člana

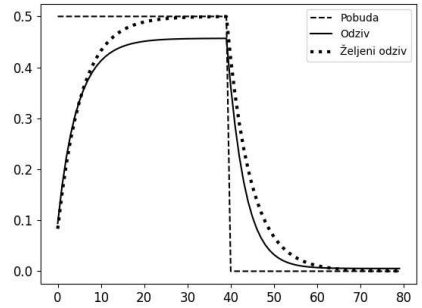
4.1.2 Hiperbolni tangens aktivacijska funkcija

Iz slike 4.3 je vidljivo da naučena neuronska mreža s aktivacijskom funkcijom u formi hiperbolnog tangensa aproksimira P1 član jednako dobro kao i neuronska mreža sa sigmoidalnom aktivacijskom funkcijom. Jednako kao i kod sigmoidalne aktivacijske funkcije postoji prebačaj za željenim odzivom. NRMS nakon učenja je 0.434490 .

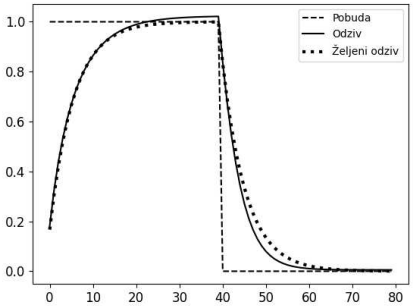


Slika 4.3: Odziv na pobudu iz podataka za učenje P1 član s hiperbolnim tangensom kao aktivacijskom funkcijom

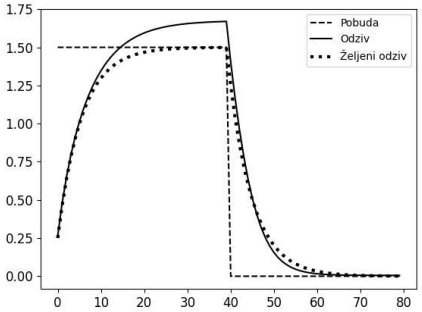
Iz slike 4.4 vidljivo je da su genralizacijska svojstva neuronske mreže s hiperbolnim tangensom kao aktivacijskom funkcijom lošija od sigmoidalne aktivacijske funkcije. Odzivi su slični, ali su odstupanja od željenog odziva veća kod aktivacijske funkcije u formi hiperbolnog tangensa.



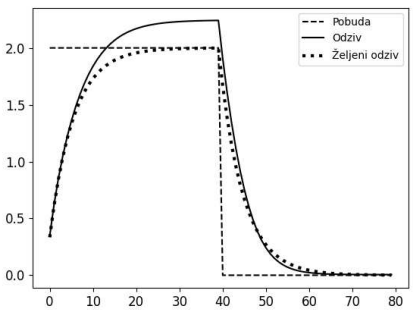
(a) Amplitude 0.5, NRMS = 0.436941



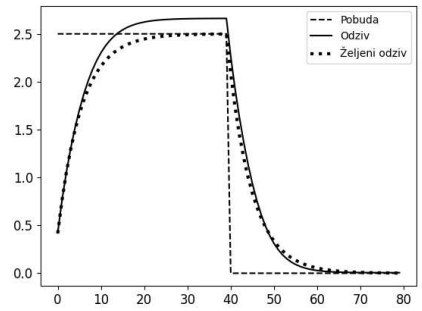
(b) Amplitude 1.0, NRMS = 0.454818



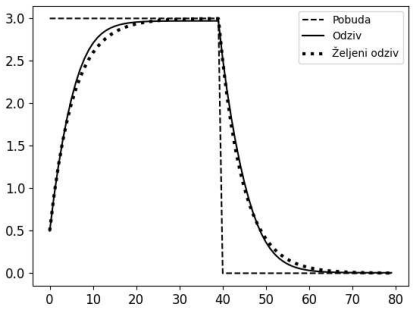
(c) Amplitude 1.5, NRMS = 0.500284



(d) Amplitude 2.0, NRMS = 0.516741



(e) Amplitude 2.5, NRMS = 0.498538

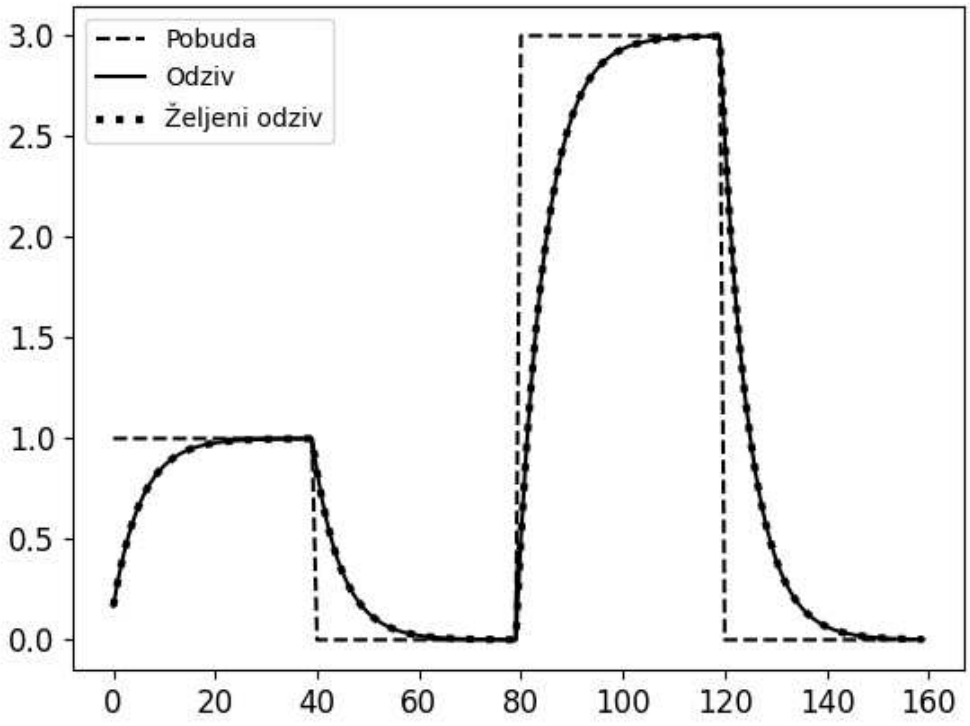


(f) Amplitude 3.0, NRMS = 0.478664

Slika 4.4: Odzivi neuronske mreže s hiperbolnim tangensom kao aktivacijskom funkcijom na testne podatke kod problema identifikacije P1 člana

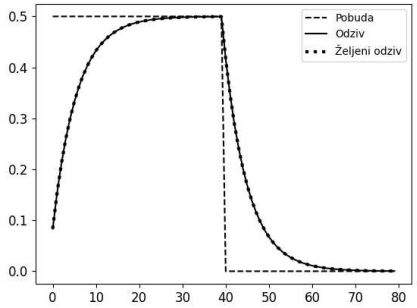
4.1.3 ReLU aktivacijska funkcija

Iz slike 4.5 je vidljivo da naučena neuronska mreža s ReLU aktivacijskom funkcijom aproksimira dinamiku P1 člana bolje i od neuronske mreže sa sigmoidalnom i s aktivacijskom funkcijom u formi hiperbolnog tangensa. NRMS nakon učenja je 0.435003 .

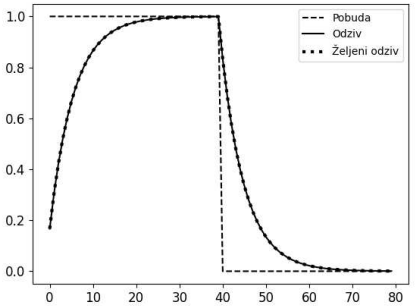


Slika 4.5: Odziv na pobudu iz podataka za učenje dinamike P1 član s ReLU aktivacijskom funkcijom

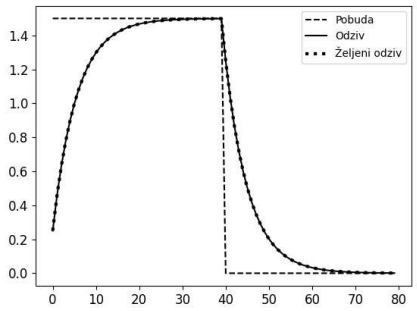
Iz slike 4.6 je vidljivo da je generalizacija neuronske mreže s ReLU aktivacijskom funkcijom daleko najbolja. Dobiven odziv neuronske mreže na svim testovima praktički ne odstupa od željenog odziva.



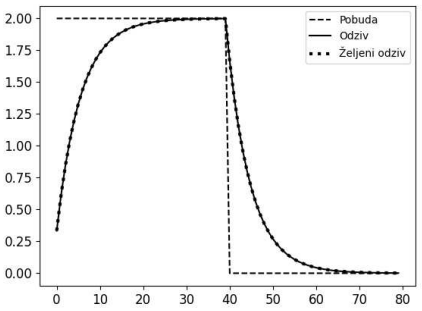
(a) Amplitude 0.5, NRMS = 0.476565



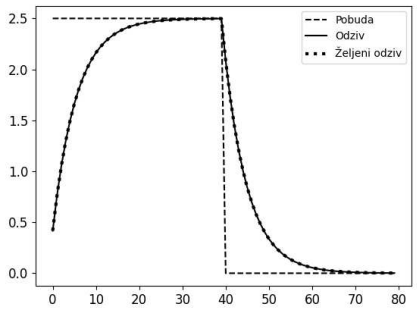
(b) Amplitude 1.0, NRMS = 0.476567



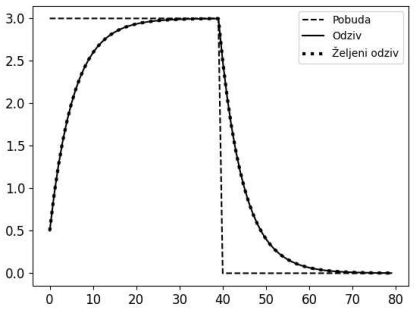
(c) Amplitude 1.5, NRMS = 0.476569



(d) Amplitude 2.0, NRMS = 0.476569



(e) Amplitude 2.5, NRMS = 0.476570

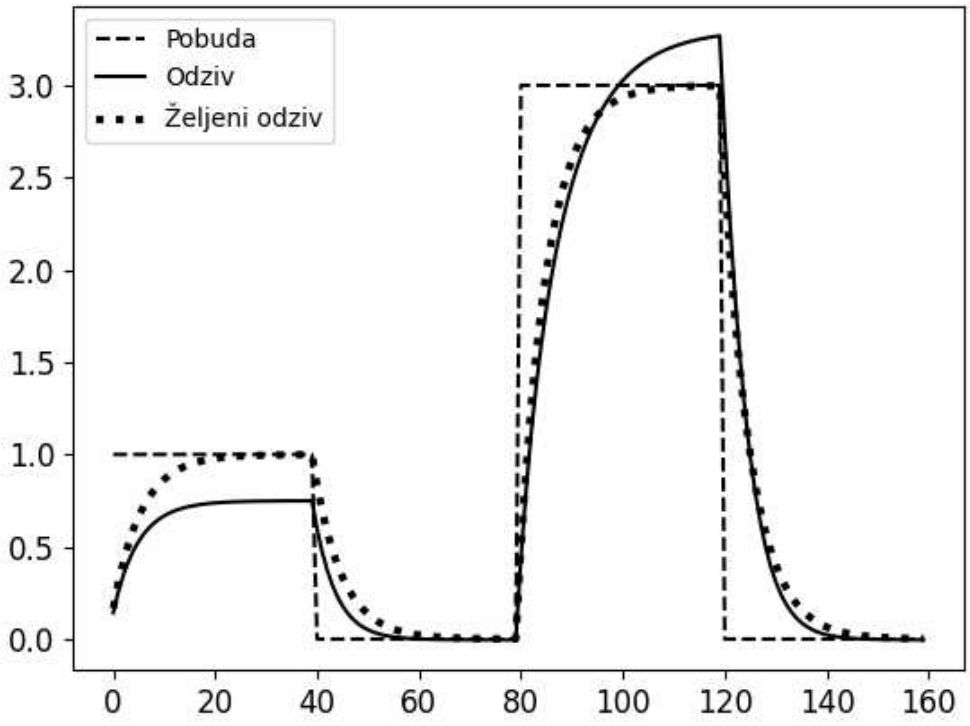


(f) Amplitude 3.0, NRMS = 0.476570

Slika 4.6: Odzivi neuronske mreže s ReLU aktivacijskom funkcijom na testne podatke kod problema identifikacije P1 člana

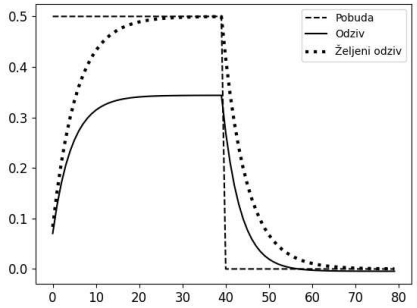
4.1.4 Softplus aktivacijska funkcija

Iz slike 4.7 je vidljivo da naučena neuronska mreža sa softplus aktivacijskom funkcijom aproksimira P1 član najlošije. Pri rastu i padu funkcije aproksimacija je bolja od sigmoidalne i aktivacijske funkcije u formi hiperbolnog tangensa, ali pri ekstremima prebačaj je neprihvatljiv. Sa parametrima učenja korištenim za ovaj test neuronska mreža ostane u lokalnom minimumu iz kojeg ne može izaći za zadani broj iteracija, dok ostale neuronske mreže uspješno konvergiraju. NRMS nakon učenja je 0.475614 .

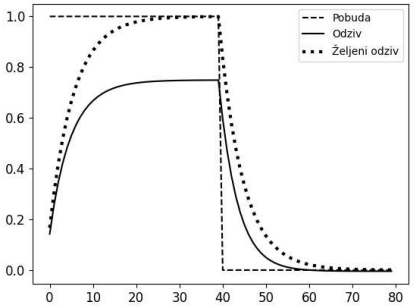


Slika 4.7: Odziv na pobudu iz podataka za učenje dinamike P1 član sa softplus aktivacijskom funkcijom

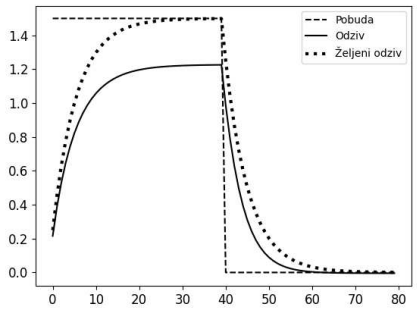
Iz slike 4.8 je vidljivo da odziv neuronske mreže lošiji od svih testova.



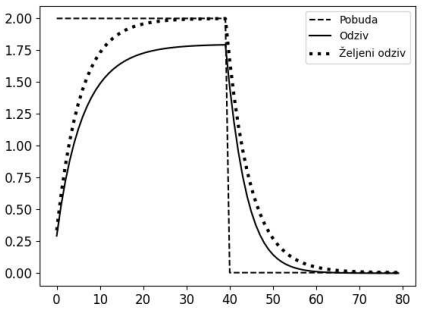
(a) Amplitude 0.5, NRMS = 0.598824



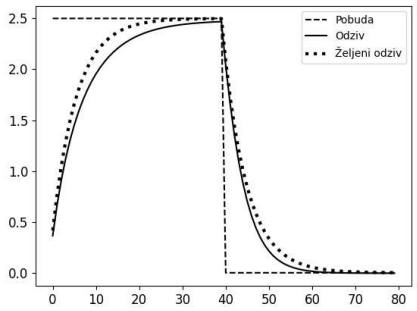
(b) Amplitude 1.0, NRMS = 0.557387



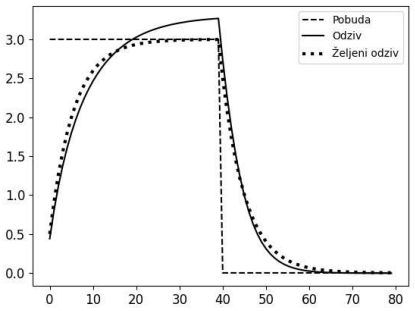
(c) Amplitude 1.5, NRMS = 0.523075



(d) Amplitude 2.0, NRMS = 0.499743



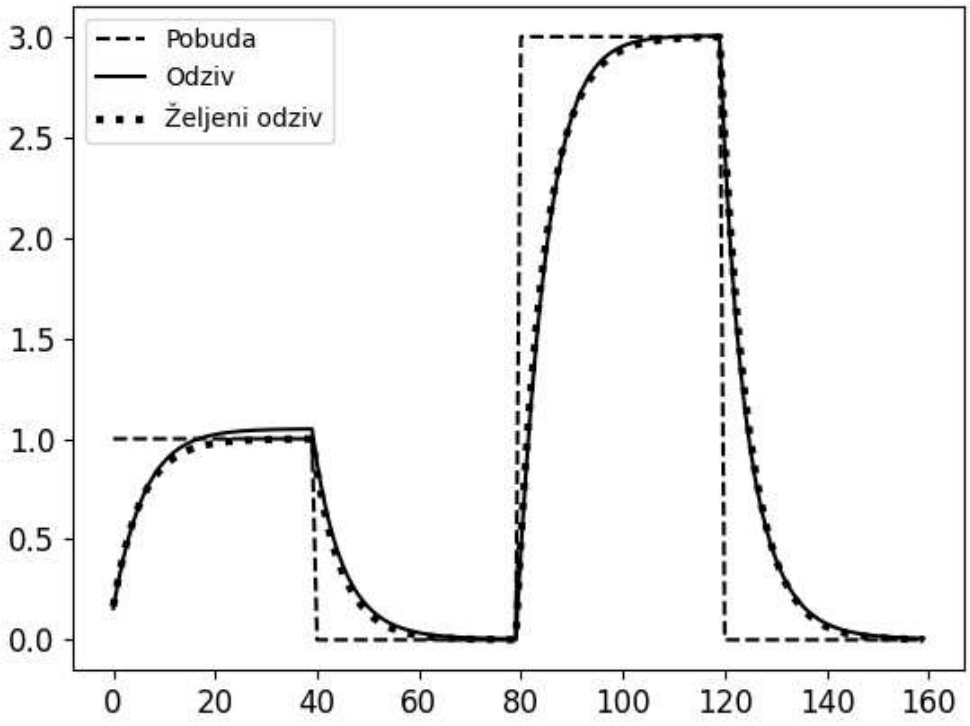
(e) Amplitude 2.5, NRMS = 0.495155



(f) Amplitude 3.0, NRMS = 0.516438

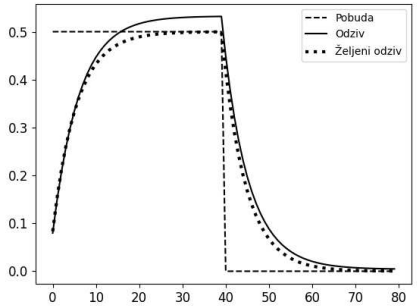
Slika 4.8: Odzivi neuronske mreže sa softplus aktivacijskom funkcijom na testne podatke kod problema identifikacije P1 člana

Promjenom broja neurona u skrivenom sloju neuronske mreže s dva na pet, dobiveni odziv je gotovo jednako dobar kao i kod neuronske mreže s ReLU aktivacijskom funkcijom s dva neurona u skrivenom sloju. NRMS nakon učenja je 0.437279 .

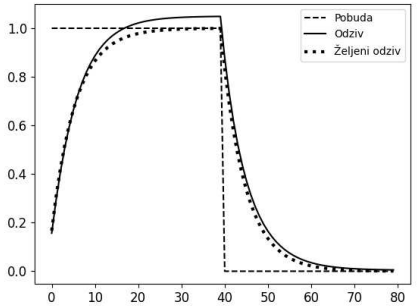


Slika 4.9: Odziv na pobudu iz podataka za učenje P1 član sa softplus aktivacijskom funkcijom

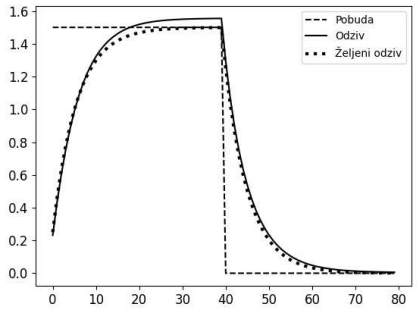
Iz slike 4.10 vidljivo je da su generalizacijska svojstva nešto lošija od ReLU aktivacijske funkcije, a bolja od sigmoidalne aktivacijske funkcije i od aktivacijske funkcije u formi hiperbolnog tangensa.



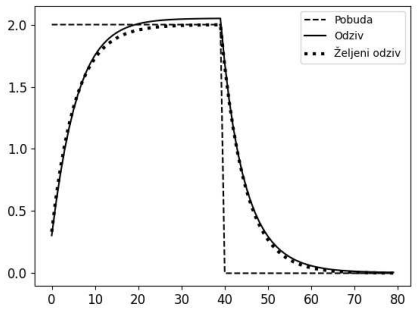
(a) Amplitude 0.5, NRMS = 0.509527



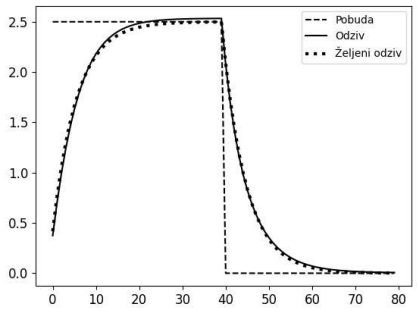
(b) Amplitude 1.0, NRMS = 0.500051



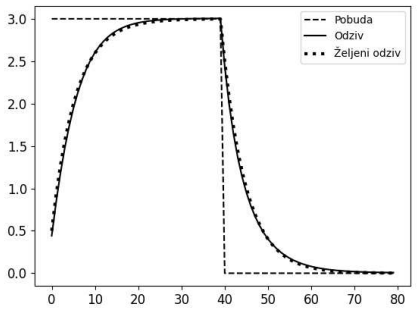
(c) Amplitude 1.5, NRMS = 0.492874



(d) Amplitude 2.0, NRMS = 0.486795



(e) Amplitude 2.5, NRMS = 0.481570

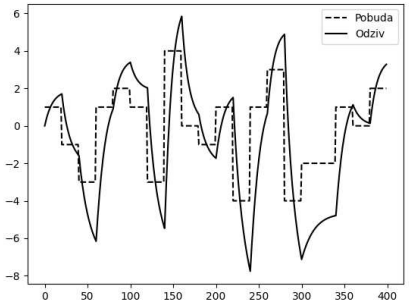


(f) Amplitude 3.0, NRMS = 0.477141

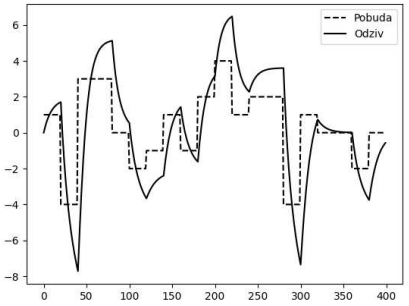
Slika 4.10: Odzivi P1 člana na testne podatke sa softplus aktivacijskom funkcijom s pet neurona u skrivenom sloju.

4.2 Rezultati identifikacije dinamike nelinearnog člana

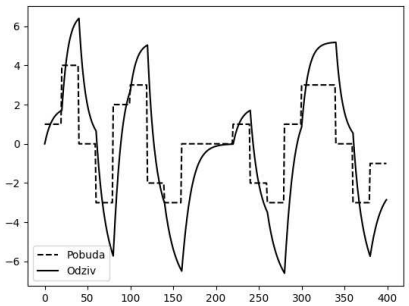
Kako bi se testirao odziv neuronske mreže i na signale koji nisu obuhvaćeni učenjem generirani su novi APRBS signali. Generirani signali za testiranje i njihovi željeni odzivi prikazani su u slici 4.11



(a) APRBS signal 1



(b) APRBS signal 2

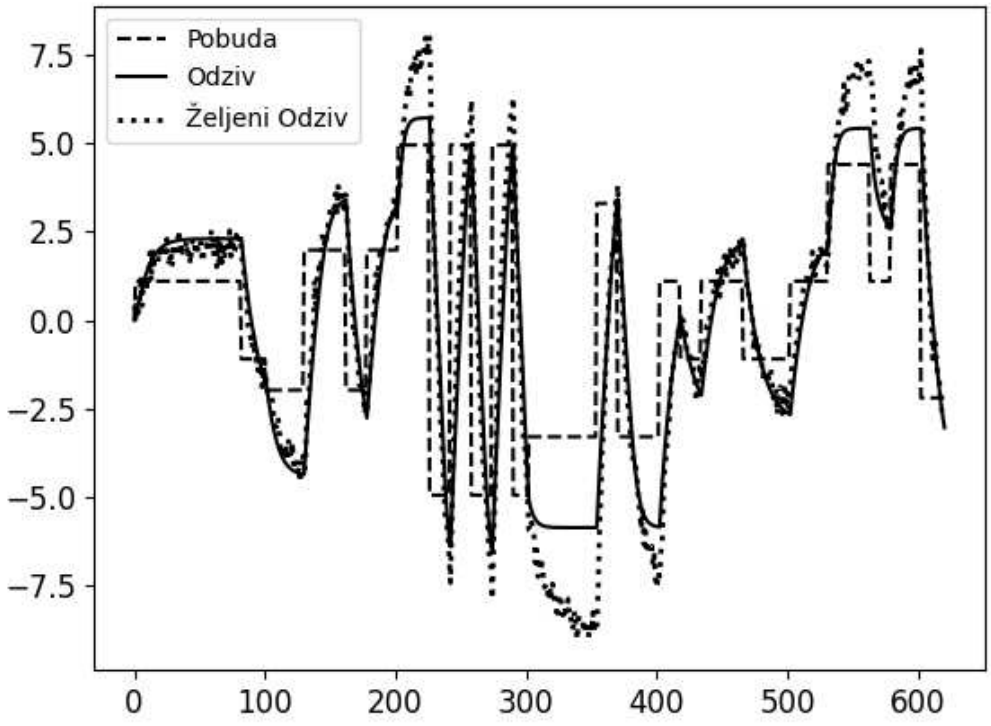


(c) APRBS signal 3

Slika 4.11: Testni podaci za identifikaciju nelinearnog člana

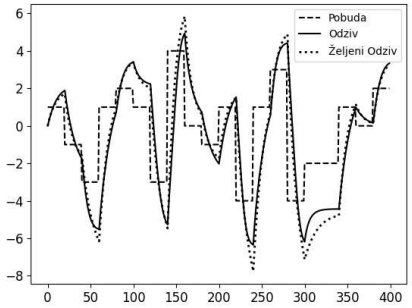
4.2.1 Sigmoidalna aktivacijska funkcija

Iz slike 4.12 je vidljivo da naučena neuronska mreža sa sigmoidalnom aktivacijskom funkcijom dobro aproksimira nelinearnu jednadžbu diferencije svugdje osim u ekstremima. NRMS nakon učenja je 0.886648 .

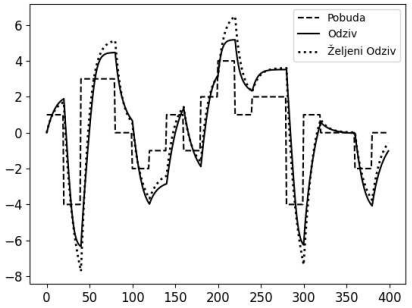


Slika 4.12: Odziv neuronske mreže sa sigmoidalnom aktivacijskom funkcijom kod identifikacije dinamike nelinearnog člana.

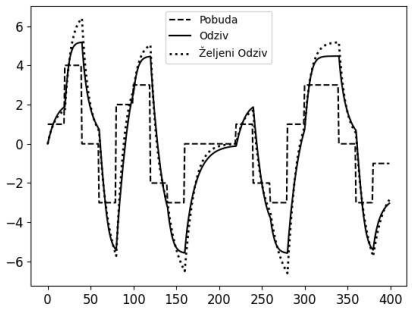
Iz slike 4.13 je vidljivo da su generalizacijska svojstva neuronske mreže sa sigmoidalnom aktivacijskom funkcijom dosta dobra, ali jednako kao i na podacima na kojima je izvršeno učenje, greška je najveća u ekstremima funkcije, dok se rast i pad funkcije prati gotovo potpuno točno.



(a) 4.11a signal, NRMS = 0.996446



(b) 4.11b signal, NRMS = 0.936104

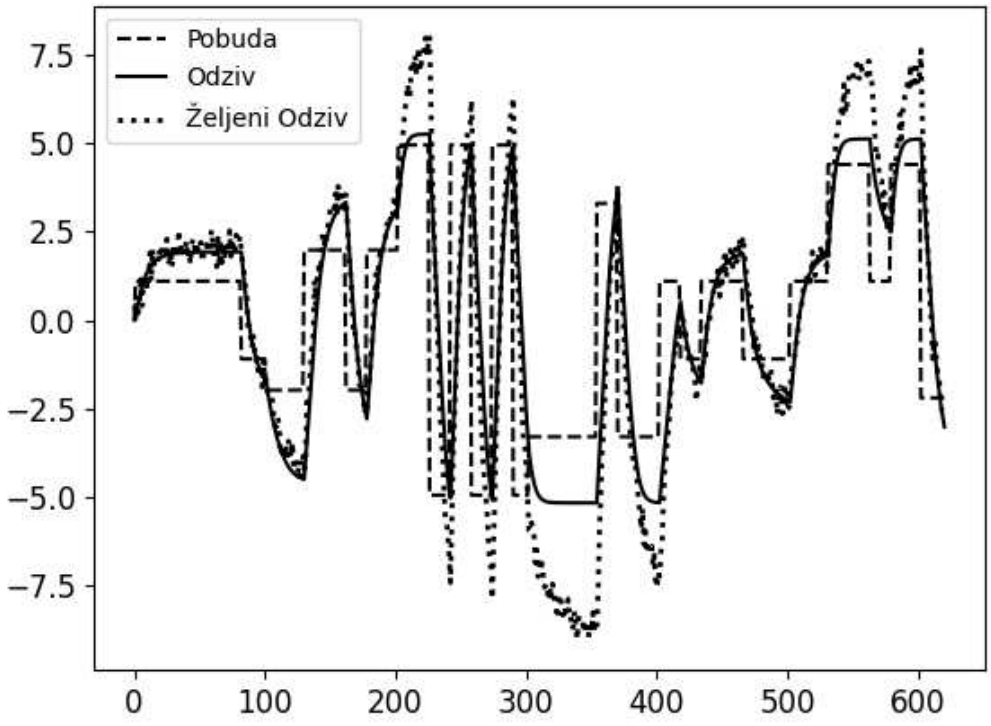


(c) 4.11c signal, NRMS = 0.908039

Slika 4.13: Odziv neuronske mreže sa sigmoidalnom aktivacijskom funkcijom na testne podatke kod identifikacije dinamike nelinearnog člana.

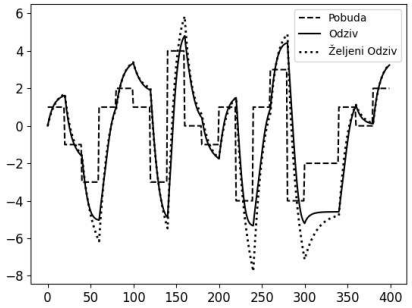
4.2.2 Hiperbolni tangens aktivacijska funkcija

Iz slike 4.14 je vidljivo da naučena neuronska mreža s aktivacijskom funkcijom u formi hiperbolnog tangensa gotovo identično aproksimira nelinearnu jednadžbu diferencije, kao i neuronske mreže sa sigmoidalnom aktivacijskom funkcijom. NRMS nakon učenja je 0.805356 .

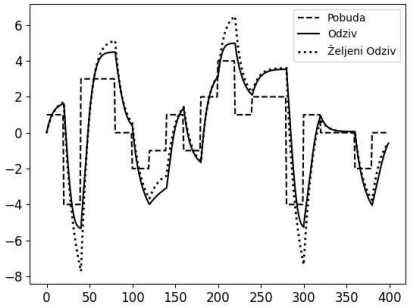


Slika 4.14: Odziv neuronske mreže s hiperbolnim tangensom kao aktivacijskom funkcijom kod identifikacije dinamike nelinearnog člana.

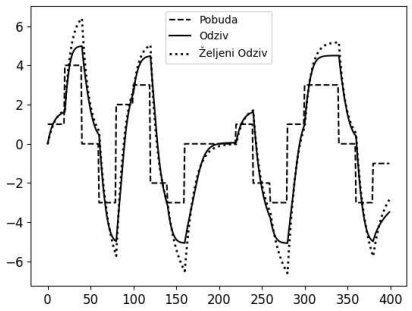
Usporedbom slika 4.13 i 4.15 vidljivo je su generilazijska svojstva gotovo identična kod praćenja rasta i pada funkcije, ali neuronska mreža s aktivacijskom funkcijom u formi hiperbolnog tangensa više griješi u ekstremima odziva.



(a) 4.11a signal, NRMS = 0.945119



(b) 4.11b signal, NRMS = 0.885636

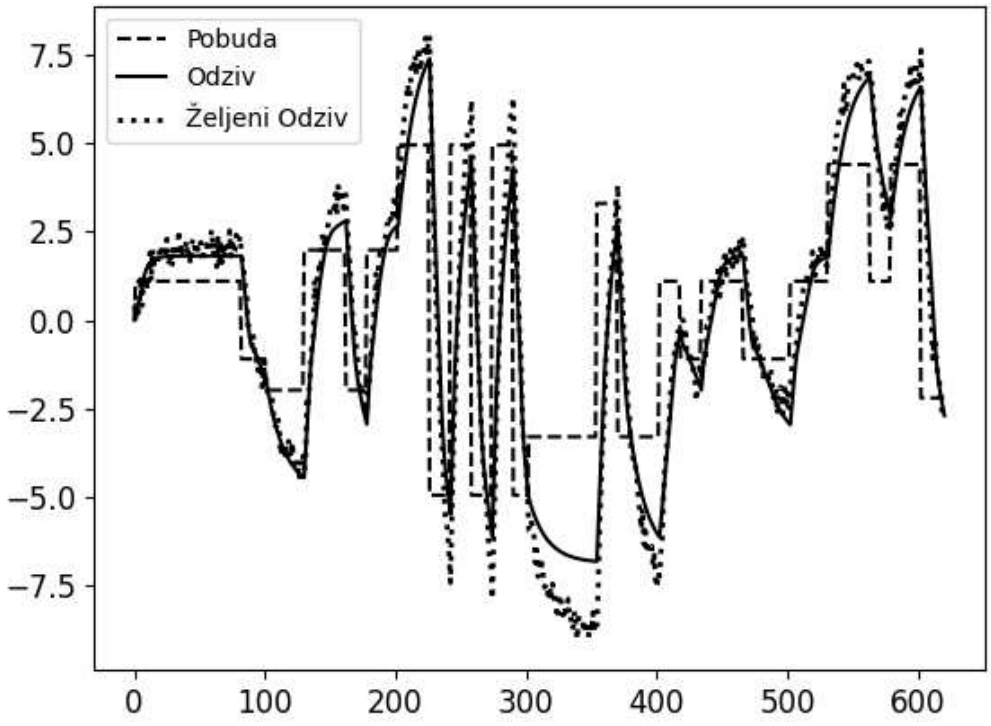


(c) 4.11c signal, NRMS = 0.875886

Slika 4.15: Odziv neuronske mreže s hiperbolnim tangensom kao aktivacijskom funkcijom na testne podatke kod identifikacije dinamike nelinearnog člana.

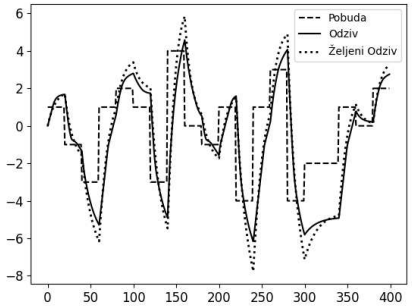
4.2.3 ReLU aktivacijska funkcija

Iz slike 4.16 je vidljivo da naučena neuronska mreža s ReLU aktivacijskom funkcijom najbolje aproksimira nelinearnu jednadžbu diferencije s najmanjom greškom u ekstremima funkcije. NRMS nakon učenja je 0.879255 .

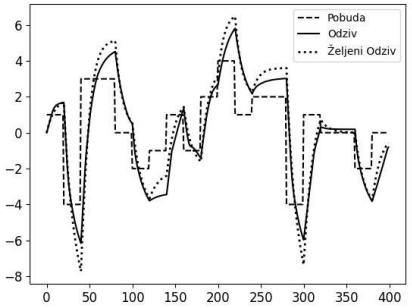


Slika 4.16: Odziv neuronske mreže s ReLU aktivacijskom funkcijom kod identifikacije dinamike nelinearnog člana.

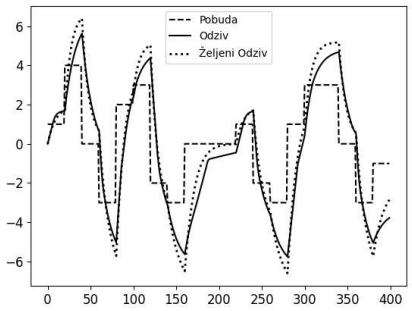
U usporedbi s sigmoidalnom aktivacijskom funkcijom i aktivacijskom funkcijom u formi hiperbolnog tangensa neuronska mreža s ReLU aktivacijskom funkcijom ima najbolja generalizacijska svojstva što je vidljivo usporedbom 4.13, 4.15 i 4.17 slika. Greška u ekstremima kod ReLU aktivacijske funkcije je puno manja od grešaka kod sigmoidalne aktivacijske funkcije i aktivacijskie funkcije u formi hiperbolnog tangensa.



(a) 4.11a signal, NRMS = 0.964899



(b) 4.11b signal, NRMS = 0.907838

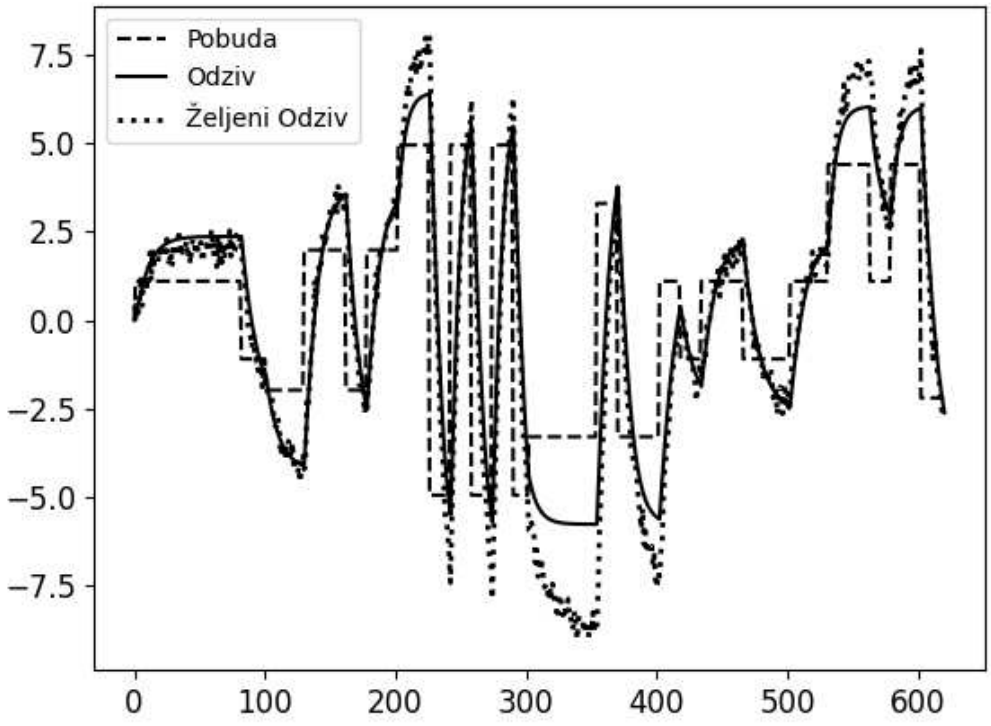


(c) 4.11c signal, NRMS = 0.947352

Slika 4.17: Odziv neuronske mreže s ReLU aktivacijskom funkcijom na testne podatke kod identifikacije dinamike nelinearnog člana.

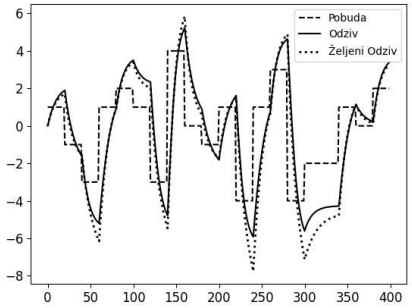
4.2.4 Softplus aktivacijska funkcija

Iz slike 4.18 je vidljivo da naučena neuronska mreža sa softplus aktivacijskom funkcijom aproksimira nelinearnu jednadžbu diferencije gotovo identično kao i mreža sa sigmoidalnom aktivacijskom funkcijom. NRMS nakon učenja je 0.835355 .

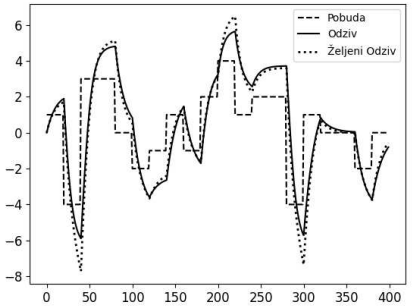


Slika 4.18: Odziv neuronske mreže sa softplus aktivacijskom funkcijom kod identifikacije dinamike nelinearnog člana.

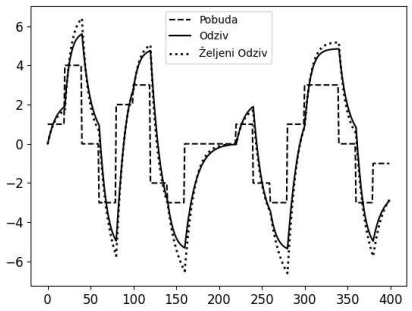
U usporedbi sa sigmoidalnom aktivacijskom funkcijom i aktivacijskom funkcijom u formi hiperbolnog tangensa, neuronska mreža sa softplus aktivacijskom funkcijom ima slična generalizacijska svojstva što je vidljivo usporedbom 4.13, 4.15 i 4.19 slika. Greška u ekstremima je malo manja od sigmoidalne aktivacijske funkcije i od aktivacijske funkcije u formi hiperbolnog tangensa, ali veća od obične ReLU funkcije.



(a) 4.11a signal, NRMS = 0.932173



(b) 4.11b signal, NRMS = 0.898182

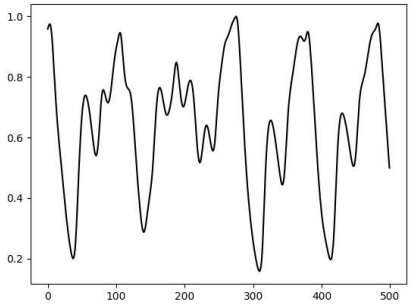


(c) 4.11c signal, NRMS = 0.887712

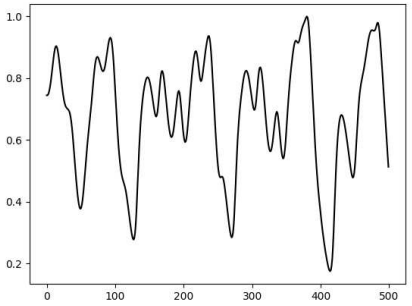
Slika 4.19: Odziv neuronske mreže sa softplus aktivacijskom funkcijom na testne podatke kod identifikacije dinamike nelinearnog člana.

4.3 Rezultati predikcije Mackey-Glass vremenske serije

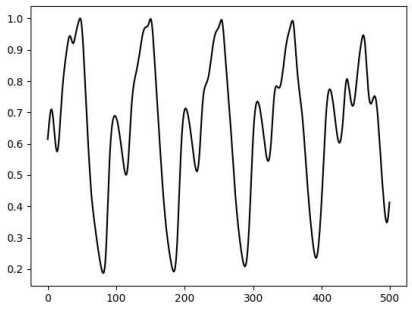
Kako bi se testirala generalizacijska svojstva neuronske mreže, generirani su podaci Mackey-Glass vremenske serije koji su različiti od podatka korištenih za učenje. Slika 4.20 prikazuje vremenske serije generirane za testiranje naučenih neuronskih mreža.



(a) Mackey-Glass vremenska serija 1



(b) Mackey-Glass vremenska serija 2

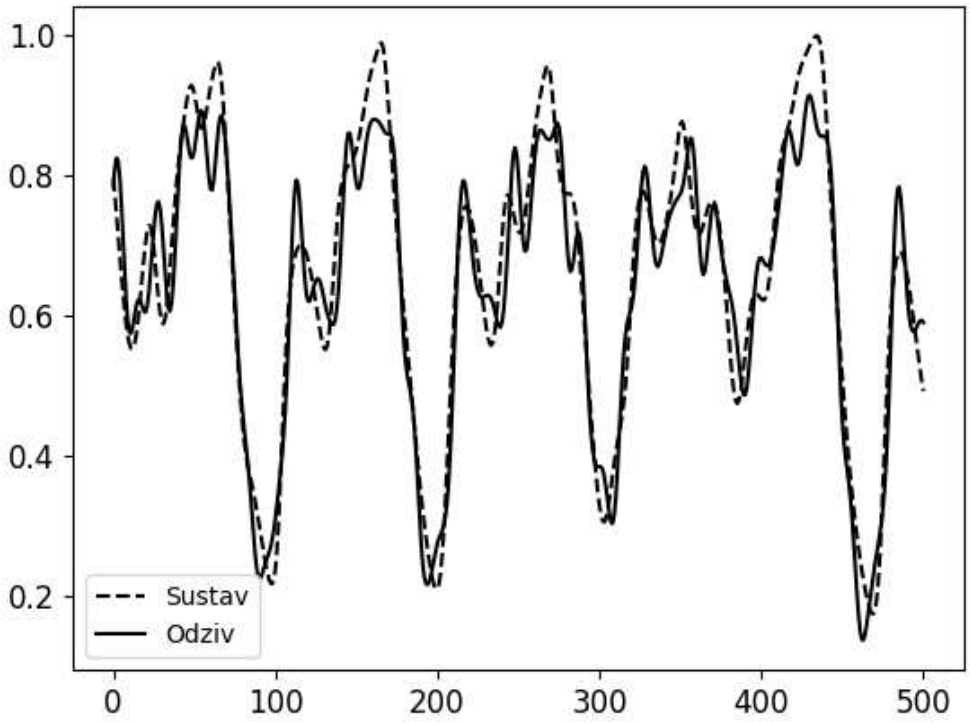


(c) Mackey-Glass vremenska serija 3

Slika 4.20: Testni podaci Mackey-Glass vremenske serije

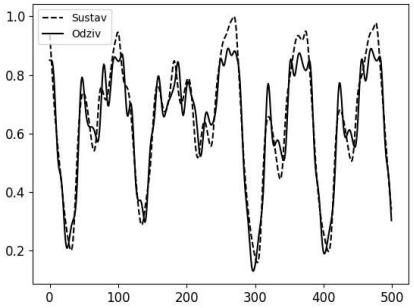
4.3.1 Sigmoidalna aktivacijska funkcija

Iz slike 4.21 je vidljivo da odziv naučene neuronske mreže sa sigmoidalnom aktivacijskom funkcijom prati vremensku seriju s greškom koja je veća u ekstremima željenog odziva. NRMS nakon učenja je 0.590568 .

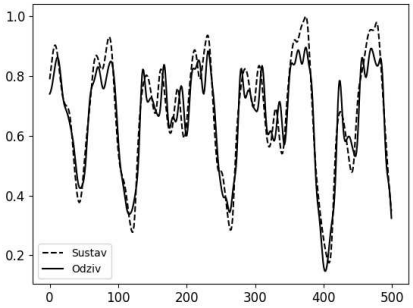


Slika 4.21: Odziv neuronske mreže sa sigmoidalnom aktivacijskom funkcijom za vremensku seriju korištenu za učenje

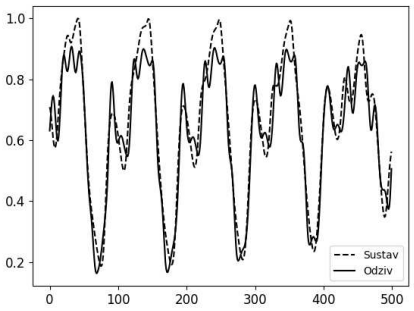
Usporedbom slika 4.21 i 4.22 vidljivo je da su generalizacijska svojstva dobra jer greška nije veća od greške dobivene na podacima za učenje. Jednako kao i na podacima za učenje vidljivo je da je najveća greška u ekstremima željenog odziva.



(a) 4.20a vremenska serija, NRMS = 0.583857



(b) 4.20b vremenska serija, NRMS = 0.633066

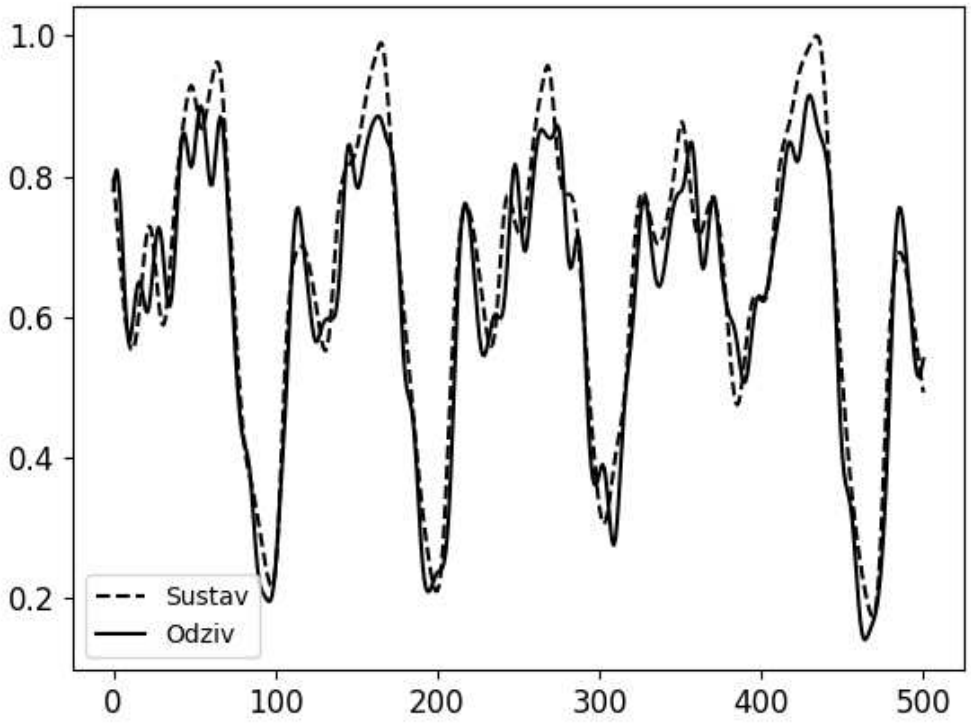


(c) 4.20c vremenska serija, NRMS = 0.574998

Slika 4.22: Odziv neuronske mreže sa sigmoidalnom aktivacijskom funkcijom za testne vremenske serije

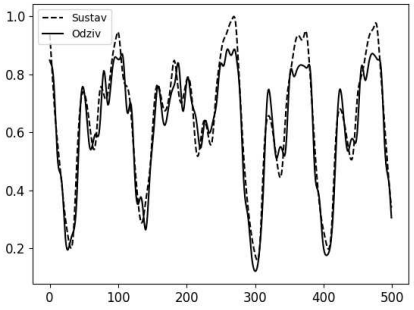
4.3.2 Tangens hiperbolni aktivacijska funkcija

Iz slike 4.23 je vidljivo da odziv naučene neuronske mreže s aktivacijskom funkcijom u formi hiperbolnog tangensa nije znatno različit od odziva neuronske mreže sa sigmoidalnom aktivacijskom funkcijom. NRMS nakon učenja je 0.590568 .

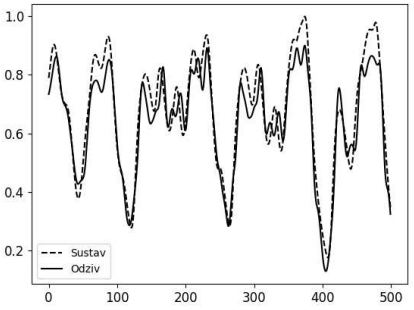


Slika 4.23: Odziv neuronske mreže s hiperbolnim tangensom kao aktivacijskom funkcijom za vremensku seriju korištenu za učenje

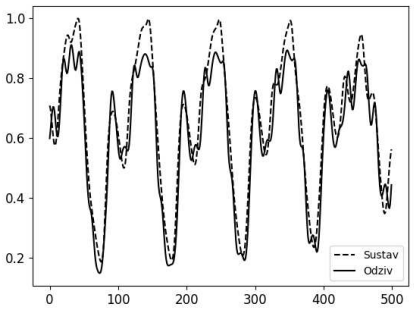
Jednako kao i kod sigmoidalne akivacijske funkcije usporedbom slika 4.23 i 4.24 vidljivo je da su generalizacijska svojstva dobra jer greška nije veća od greške na podacima za učenje. Jednako kao i na podacima za učenje vidljivo je da je najveća greška u ekstremima željenog odziva.



(a) 4.20a vremenska serija, NRMS = 0.583857



(b) 4.20b vremenska serija, NRMS = 0.633066

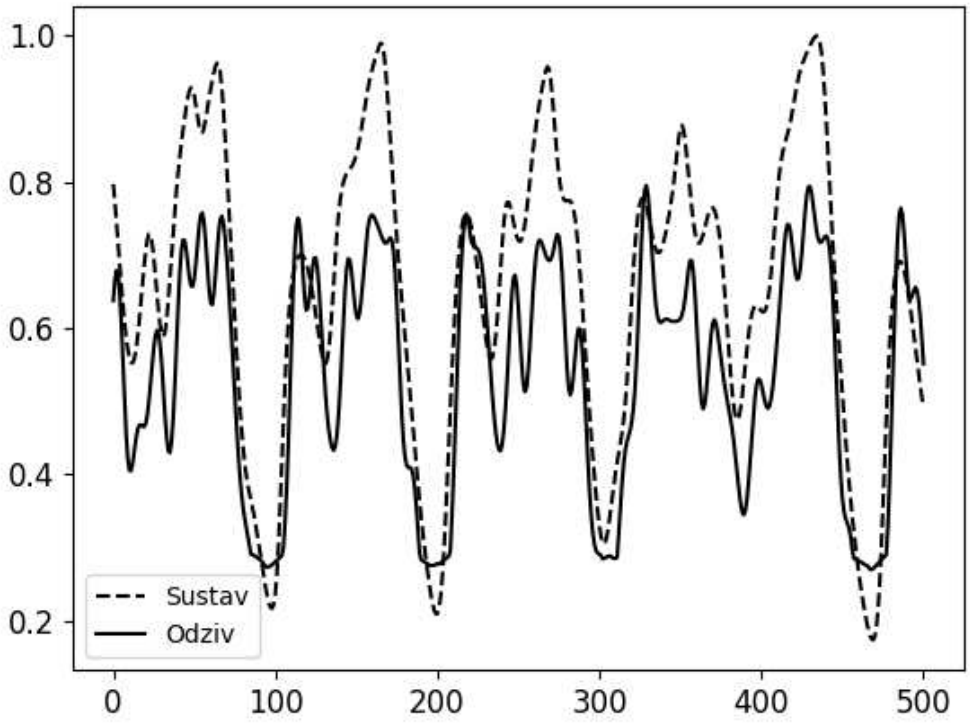


(c) 4.20c vremenska serija, NRMS = 0.574998

Slika 4.24: Odziv neuronske mreže s hiperbolnim tangensom kao aktivacijskom funkcijom za testne vremenske serije

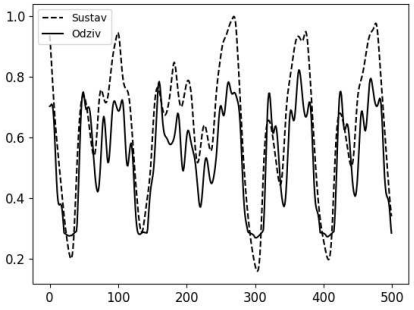
4.3.3 ReLU aktivacijska funkcija

Iz slike 4.25 je vidljivo da je odziv naučene neuronske mreže s ReLU aktivacijskom funkcijom je znatno lošiji od odziva neuronske mreže sa sigmoidalnom aktivacijskom funkcijom ili aktivacijskom funkcijom u formi hiperbolnog tangensa. NRMS nakon učenja je 0.590568 .

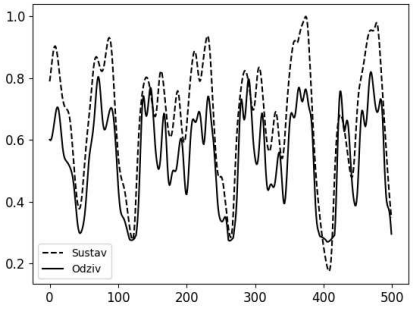


Slika 4.25: Odziv neuronske mreže s ReLU aktivacijskom funkcijom za vremensku seriju korištenu za učenje

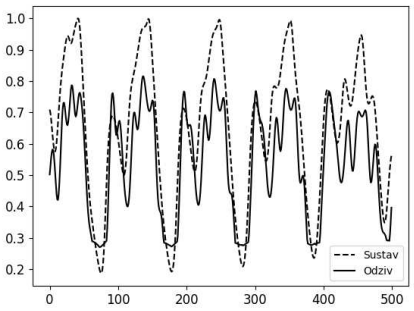
Usporedbom slika 4.25 i 4.26 vidljivo je da odziv neuronske mreže loše prati vremensku seriju s značajnim greškama u ekstremima vremenske serije.



(a) 4.20a vremenska serija, NRMS = 0.583857



(b) 4.20b vremenska serija, NRMS = 0.633066

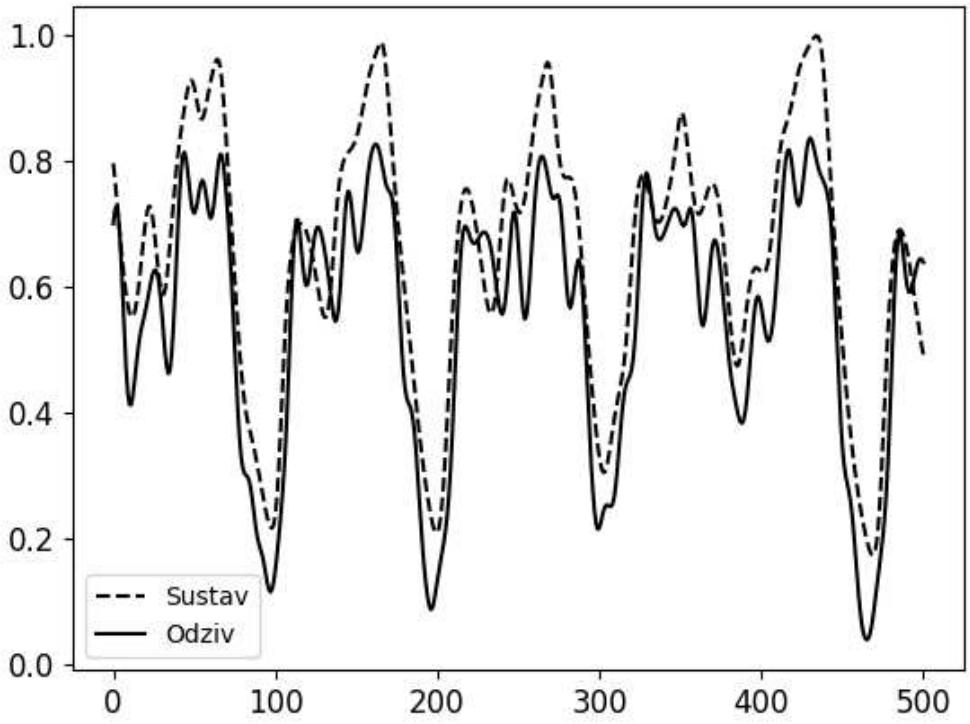


(c) 4.20c vremenska serija, NRMS = 0.574998

Slika 4.26: Odziv neuronske mreže s ReLU aktivacijskom funkcijom za testne vremenske serije

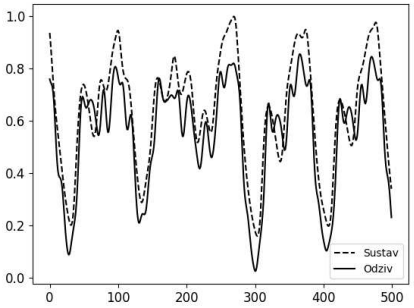
4.3.4 Softplus aktivacijska funkcija

Iz slike 4.27 je vidljivo da je odziv naučene neuronske mreže s softplus aktivacijskom funkcijom je bolji od odziva neuronske mreže s ReLU aktivacijskom funkcijom, ali lošiji od neuronske mreže sa sigmoidalnom aktivacijskom funkcijom ili aktivacijskom funkcijom u formi hiperbolnog tangensa. NRMS nakon učenja je 0.590568 .

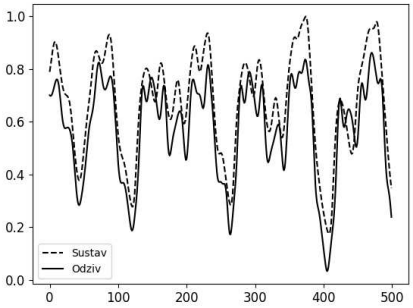


Slika 4.27: Odziv neuronske mreže sa softplus aktivacijskom funkcijom za vremensku seriju korištenu za učenje

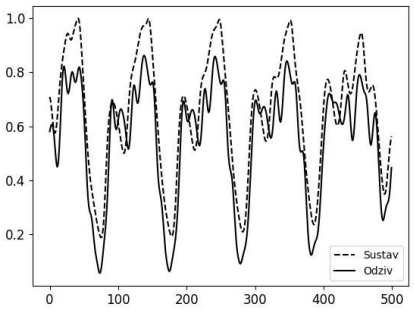
Usporedbom slika 4.22, 4.24, 4.26 i 4.28 vidljivo je da su generalizacijska svojstva najbolja kod sigmoidalne aktivacijske funkcije i aktivacijske funckije u formi hiperbolnog tangensa, najlošija kod ReLU aktivacijske funkcije, a softplus aktivacijska funkcija daje rezultat između navedenih aktivacijskih funkcija.



(a) 4.20a vremenska serija, NRMS = 0.583857



(b) 4.20b vremenska serija, NRMS = 0.633066



(c) 4.20c vremenska serija, NRMS = 0.574998

Slika 4.28: Odziv neuronske mreže sa softplus aktivacijskom funkcijom za testne vremenske serije

5 Zaključak

U ovome je radu ukratko opisana struktura i princip funkcioniranja višeslojne perceptronske neuronske mreže, razvijena je programska podrška za učenje neuronskih mreža s raznim aktivacijskim funkcijama i provedena je usporedna analiza aktivacijskih funkcija na nekoliko standardnih testova.
Programska podrška je izvedena na način da podržava razne topologije i aktivacijske funkcije neuronskih mreža s raznim parametrima učenja koji su lako podesivi u konfiguracijskim datotekama. Aplikacija je napravljena bez ikakvog korištenja modula za strojno učenje ili linearnu algebru što omogućava potpunu kontrolu nad modelom strojnog učenja. Provedeni su testovi identifikacije linearnog P1 dinamičkog člana, identifikacije nelinearnog dinamičkog sustava i predviđanje Mackey-Glass vremenske serije.

Kod identifikacije linearnog P1 dinamičkog člana neuronska mreža s ReLU aktivacijskom funkcijom daje najbolje rezultate. Primjena sigmoidalne aktivacijske funkcije i hiperbolnog-tangensa daje podjednako lošije rezultate, a neuronska mreža sa Softplus aktivacijskom funkcijom daje najlošiji rezultat, osim ako se ne poveća broj neurona u skrivenom sloju. Kod povećanja broja neurona skrivenog sloja neuronske mreže sa Softplus aktivacijskom funkcijom rezultati su približni rezultatima neuronske mreže s ReLU aktivacijskom funkcijom.

Kod identifikacije nelinearnog dinamičkog sustava ReLU neuronska mreža daje bolje rezultate od neuronskih mreža sa sigmoidalnom i tangens-hiperbolnom aktivacijskom funkcijom. Neuronska mreža sa Softplus aktivacijskom funkcijom daje rezultate bolje od neuronskih mreža sa sigmoidalnom i tangens-hiperbolni aktivacijskom funkcijom, a lošije od neuronske mreže s ReLU aktivacijskom funkcije.

Kod predviđanja Mackey-Glass vremenske serije najbolji odziv daje neuronska mreža sa sigmoidalnom aktivacijskom funkcijom, zatim mreža koja za aktivacijsku funkciju koristi hiperbolni-tangens, pa Softplus funkciju. Najlošiji rezultat postignut je s neuronskom mrežom s ReLU aktivacijskom funkcijom.

Svojstva ReLU i Softplus aktivacijskih funkcija da domena funkcije nije u konačnom intervalu loše utječe na numeričku stabilnost učenja kod problema predviđanja vremenskih serija i potrebno je smanjiti momentum kako bi se dobilo numerički moguće rješenje.

Perceptronska neuronska mreža s ReLU aktivacijskom funkcijom može rješavati pro-

bleme identifikacije dinamičkih sustava bolje od neuronske mreže sa sigmoidalnom funkcijom i hiperbolnim-tangensom. Jednostavna implementacija ReLU funkcije i točniji odzivi čine tu aktivacijsku funkciju dobrim kandidatom za probleme identifikacije dinamičkih sustava, dok za problem predikcije kaotične vremenske serije ne daje dovoljno dobar rezultat.

Literatura

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] Branko Novaković, Dubravko Majetić, and Mladen Široki. Umjetne neuronske mreže, 1998.

[3] Liang Wang, Jianxin Zhao, and Richard Mortier. Ocaml scientific computing. *Cham, Switzerland: Springer*, 2022.

[4] Sebastian Raschka. Activation Functions for Artificial Neural Networks. https://sebastianraschka.com/faq/docs/activation-functions.html.

[5] GNU Project. An Inline Function is As Fast As a Macro. https://gcc.gnu.org/onlinedocs/gcc/Inline.html.

[6] GNU Project. C99 Status for GCC. https://gcc.gnu.org/c99status.html.

[7] Peter Miller. Recursive make considered harmful. *AUUGN Journal of AUUG Inc*, 19(1):14–25, 1998.

A Primjer rada programa na XOR problemu

A.1 NNfile datoteka

```
config = {
    eta = 0.1
    alpha = 0.8
    N = 5000
    NRMS = 0.01
}

topology = {
    |> (2) input
    |> (2) dense
    |> (2) activation :: tanh
    |> (1) dense
    |> (1) output
}

data = [
0 0 :: 0
0 1 :: 1
1 0 :: 1
1 1 :: 0
]
```

A.2 bytecode.c datoteka

```
/*
* Bytecode enums
*/
typedef enum {
Input    = 0,
```

```
Output  = 1,
Dense   = 2,
Activation      = 3,
} layers;

typedef enum {
None    = 0,
Sigmoid = 1,
Tanh    = 2,
ReLu    = 3,
Softplus        = 4,
} activations;

/*
* Parametars
*/
double w[] = {
        /* Input layer */
        0,
        0,
        /* Dense layer */
        -0.954470,      /* b */
        0.225736,       /* b */
        1.865377,       /* w[0][0] */
        -1.060038,      /* w[0][1] */
        1.639017,       /* w[1][0] */
        -0.874662,      /* w[1][1] */
        0.000000,       /* result buffer */
        0.000000,       /* result buffer */
        /* Activation layer */
        0.000000,       /* result buffer */
        0.000000,       /* result buffer */
        /* Dense layer */
        1.489708,       /* b */
        1.498447,       /* w[0][0] */
        -1.661651,      /* w[0][1] */
        0.000000,       /* result buffer */
```

```
        /* Output layer */
};

/*
* Metadata
*/
#define NO_LAYERS  5

int metadata[][5] = {
        {Input, 2},
        {Dense, 2, 8, 2, 2},
        {Activation, Tanh, 10, 2},
        {Dense, 12, 15, 2, 1},
        {Output, 15, 1},
};
```

A.3 ann.h datoteka

```
#define NO_LAYERS 5

#ifndef ANN_H
#define ANN_H

#define DEBUG
#ifdef DEBUG
#include <math.h>
#endif // !DEBUG

/*
 * INPUT = I
 */
#define I_NO_INPUTS             1

/*
 * OUTPUT = O
 */
#define O_START_INDEX           1
```

```
#define O_NO_OUTPUTS            2

/*
 * WEIGHTS = W
 */
#define W_START_INDEX           1
#define W_END_INDEX             2
#define W_NO_INPUTS             3
#define W_NO_OUTPUTS            4

/*
 * ACTIVATIONS = A
 */
#define A_FUNCTION              1
#define A_START_INDEX           2
#define A_NO_INPUTS             3

// misc
#define NO_LAYER_TYPES          4
#define NO_ACTIVATION_TYPES     5

/*
 * Bytecode enums
 */
typedef enum {
    Input      = 0,
    Output     = 1,
    Dense      = 2,
    Activation = 3,

} layers;

typedef enum {
    None    = 0,
    Sigmoid = 1,
    Tanh    = 2,
    ReLu    = 3,
```

```
    Softplus = 4,

} activations;

/*
 * Simple feedfoward neural network
 */
int snn(double *, double *);

extern double w[];
extern int metadata[][NO_LAYERS];


#ifdef ANN_IMPLEMENTATION

/*
 * Functions for lookup layer types
 */
static void
input_layer(double *w, int *layer_data, double *y, double *x)
{
    for (int i = 0; i < layer_data[I_NO_INPUTS]; i++) {
        w[i] = x[i];
    }
}

static void
output_layer(double *w, int *layer_data, double *y, double *x)
{
    for (int i = 0; i < layer_data[O_NO_OUTPUTS]; i++) {
        y[i] = w[layer_data[O_START_INDEX] + i];
    }
}

static void
dense_layer(double *w, int *layer_data, double *y, double *x)
{
```

```
/*
 *        y = x * w + b
 *        y_i = sum( x_j * w_i_j ) + b_i
 *
 */
    int w_no_inputs = layer_data[W_NO_INPUTS];
    int w_no_outputs = layer_data[W_NO_OUTPUTS];
    int w_start_index = layer_data[W_START_INDEX];
    int w_end_index = layer_data[W_END_INDEX];

    for (int i = 0; i < w_no_outputs; i++) {
        w[w_end_index + i] = 0;
        for (int j = 0; j < w_no_inputs; j++) {
            w[w_end_index + i] += w[(w_start_index + w_no_outputs) + i * w_
        }
        w[w_end_index + i] += w[w_start_index + i];
    }
}


double (*activation_lookup[NO_ACTIVATION_TYPES]) (double x);

static double
none(double x)
{
    return x;
}

static double
sigmoid(double x)
{
    double y = 1 / (1 + exp(-x));
    return y;
}

static double
relu(double x)
```

```
{
    double y = x < 0 ? 0 : x;
    return y;
}

static double
softplus(double x)
{
    double y = log(1 + exp(x));
    return y;
}

static void
activation_layer(double *w, int *layer_data, double *y, double *x)
{
    activation_lookup[None]     = &none;
    activation_lookup[Sigmoid]  = &sigmoid;
    activation_lookup[Tanh]     = &tanh;
    activation_lookup[ReLu]     = &relu;
    activation_lookup[Softplus] = &softplus;

    int a_no_inputs = layer_data[A_NO_INPUTS];
    int a_start_index = layer_data[A_START_INDEX];
    int a_function = layer_data[A_FUNCTION];

    for (int i = 0; i < a_no_inputs; i++) {
        w[a_start_index + i] = activation_lookup[a_function](w[a_start_inde
    }

}


void (*layer_lookup[NO_LAYER_TYPES]) (double *w, int *layer_data, double *y
int
snn(double *y, double *x)
{
    /*
```

```
     * Layers lookup table
     */
    layer_lookup[Input]      = &input_layer;
    layer_lookup[Output]     = &output_layer;
    layer_lookup[Dense]      = &dense_layer;
    layer_lookup[Activation] = &activation_layer;

    for (int i = 0; i < NO_LAYERS; i++) {
        layer_lookup[metadata[i][0]](w, metadata[i], y, x);
    }
    return 0;
}

#endif // ANN_IMPLEMENTATION

#endif // !ANN_H
```

A.4 Primjer korištenja naučene neuronske mreže

```
#!/usr/local/bin/tcc -run bytecode.c
#include <stdio.h>
#define ANN_IMPLEMENTATION
#include "ann.h"

/*
 * Output data
 */
double y[] = {0};

/*
 * Input data
 */
double x[][2] = {
    {0, 0},
    {0, 1},
    {1, 0},
    {1, 1},
```

```
};

int
main(void)
{
    int rc = 0;
    for (int i = 0; i < 4; i++) {
        rc = snn(y, x[i]);
        printf("------------------------------\n");
        printf("input  = %lf %lf\n", x[i][0], x[i][1]);
        printf("output = %lf\n", y[0]);
    }
    return 0;
}
```

B Ispis procedura C koda za učenje neuronskih mreža

B.1 feedfoward funkcija

```
static int
feedfoward(network *nn)
{
    unsigned int dense_counter = 0;
    unsigned int activation_counter = 0;
    int rc = 0;

    for (int layer = 0; layer < nn->topol.n_layers; layer++) {
        double *temp_y;
        //printf("Layer number: %d\n", layer);
        switch (nn->topol.layers[layer]) {
        case (Input):
            temp_y = nn->input_layer;
            break;
        case (Output):
            nn->output_layer = temp_y;
            break;
        case (Dense):
            rc = dense_result(&nn->dense_layer[dense_counter], temp_y);
            temp_y = nn->dense_layer[dense_counter].y;
            if (rc) {
                printf("ERROR dense layer matrix failed\n");
                return -1;
            }
            dense_counter++;
            break;
        case (Activation):
            rc = activation_result(&nn->activation_layer[activation_counter
            temp_y = nn->activation_layer[activation_counter].y;
```

```
            if (rc) {
                printf("ERROR activation layer failed\n");
                return -1;
            }
            activation_counter++;
            break;
        default:
            printf("Wrong layer\n");
        }
    }
    return 0;
}
```

B.2 loss funkcija

```
static int
loss(network *nn, double *data)
{
    nn->loss = 0;
    for (unsigned int i = 0; i < nn->topol.n_neurons[nn->topol.n_layers - 1
        nn->loss += pow((nn->output_layer[i] - data[i]), 2);
    }
    nn->loss = nn->loss / 2;
    return 0;
}
```

B.3 backpropagation funkcija

```
static int
backpropagation(network *nn, network *dnn)
{
    unsigned int dense_counter = nn->topol.no_dense;
    unsigned int activation_counter = nn->topol.no_activation;
    int rc = 0;
    for (int layer = nn->topol.n_layers - 1; layer >= 0; layer--) {
        double *temp_grad;
        switch (nn->topol.layers[layer]) {
```

```
        case (Input):
            break;
        case (Output):
            rc = grad_loss(nn, dnn);
            temp_grad = dnn->output_layer;
            if (rc) {
                printf("ERROR: loss function gradient failed.");
            }
            break;
        case (Dense):
            dense_counter--;
            rc = grad_dense(nn, dnn, dense_counter, temp_grad);
            temp_grad = dnn->dense_layer[dense_counter].x;
            if (rc) {
                printf("ERROR: dense layer gradient failed.");
            }
           break;
        case (Activation):
            activation_counter--;
            rc = grad_activation(nn, dnn, activation_counter, temp_grad);
            temp_grad = dnn->activation_layer[activation_counter].x;
            if (rc) {
                printf("ERROR: dense layer gradient failed.");
            }
            break;
        default:
            printf("Wrong layer\n");
        }
    }
    return rc;
}
```

B.4 update_parms funkcija

```
static int
update_params(network *nn, network *dnn)
{
```

```
    unsigned int dense_counter = 0;
    unsigned int activation_counter = 0;
    int rc = 0;
    for (int layer = 0; layer < nn->topol.n_layers; layer++) {
        switch (nn->topol.layers[layer]) {
        case (Input):
            break;
        case (Output):
            break;
        case (Dense):
            rc = update_dense(&nn->dense_layer[dense_counter], &dnn->dense_
            dense_counter++;
            if (rc) {
                printf("ERROR dense layer matrix update failed\n");
                return -1;
            }
            break;
        case (Activation):
            activation_counter++;
            break;
        default:
            printf("Wrong layer\n");

        }
    }
    return 0;
}
```

B.5 get_nrms funkcija

```
double
get_nrms(network *nn, data *train_data)
{
    #define NO_DI   (train_data->no_data * nn->topol.no_output_neurons)
    double di[train_data->no_data][nn->topol.no_output_neurons];
    double d_ = 0;
    double sigma_dn = 0;
```

```
double rms = 0;
double nrms = 0;
double temp_sum = 0;

int count = 0;

/*d_*/
for (unsigned int i = 0; i < train_data->no_data; i++) {
    nn->input_layer = train_data->input[i];
    nn->target = train_data->output[i];
    int rc = feedfoward(nn);
    if (rc){
        printf("ERROR: feedfoward failed\n");
        exit(1);
    }
    for (int j = 0; j < nn->topol.no_output_neurons; j++) {
        di[i][j] = nn->output_layer[j];
        d_ += di[i][j];
        count++;
    }
}
d_ = d_ / count;

/*sigma_dn*/
temp_sum = 0;
for (unsigned int i = 0; i < train_data->no_data; i++) {
    for (int j = 0; j < nn->topol.no_output_neurons; j++) {
        temp_sum += pow((di[i][j] - d_), 2);
    }
}
sigma_dn = sqrt(temp_sum / NO_DI);


/*rms*/
temp_sum = 0;
for (unsigned int i = 0; i < train_data->no_data; i++) {
    for (int j = 0; j < nn->topol.no_output_neurons; j++) {
```

```
                temp_sum += pow((di[i][j] - train_data->output[i][j]), 2);
            }
        }
        rms = sqrt(temp_sum / NO_DI);

        /*nrms*/
        nrms = rms / sigma_dn;

        return nrms;
}
```